



InfaMatura



# Teoria Maturalna Informatyka



WIEDZA, ZADANIA MATURALNE I ODPOWIEDZI  
Wy tłumaczone i opakowane w przystępny sposób



# Autor



## KILKA SŁÓW WSTĘPU

eBook został stworzony, aby zainspirować jak najwięcej młodych ludzi do zgłębiania wiedzy informatycznej. Jako maturzysta, sam chciałbym mieć dostęp do kompleksowego kompendium, z którego mógłbym się uczyć. Ebook zawiera moje notatki, informacje zebrane z internetu oraz zadania wraz z szczegółowymi odpowiedziami. Mam nadzieję, że okaże się dla Ciebie przydatny i pomoże Ci dostać się na wymarzone studia.

~Olaf Grykatowski

Po więcej materiałów do nauki sprawdź moje sociale !



**InfaMatura**

# Legenda

Podzieliłem ebook na rozdziały, oznaczając je według prawdopodobieństwa wystąpienia na maturze, w trzech kategoriach.



**Wysoki priorytet:** Najważniejsze zagadnienia i zadania, niemal na pewno pojawią się na maturze.



**Średni priorytet:** Powinniście je dobrze zrozumieć i biegle stosować, choć rzadziej pojawiają się na maturze.



**Niski priorytet:** Przeczytajcie, zrozumcie, ale nie musicie ich zapamiętywać szczegółowo. Mają pomóc w szerszym zrozumieniu tematu.



## Infamatura

# Spis Treści

|                                                                                                                           |     |
|---------------------------------------------------------------------------------------------------------------------------|-----|
| 1. <u>Systemy Liczbowe i Konwersje</u>   | 6   |
| 2. <u>System U2</u>                      | 17  |
| 3. <u>Bramki Logiczne</u>                | 21  |
| 4. <u>Architektura Komputerowa</u>       | 26  |
| 5. <u>Architektura Oprogramowania</u>    | 33  |
| 6. <u>Stos, Słota i Kolejka</u>          | 38  |
| 7. <u>Złożoność Obliczeniowa</u>         | 47  |
| 8. <u>Pseudokod</u>                      | 53  |
| 9. <u>Iteracja</u>                      | 62  |
| 10. <u>Rekurencja</u>                  | 66  |
| 11. <u>Algorytmy</u>                   | 71  |
| a. <u>Sortowania</u>                                                                                                      | 72  |
| b. <u>Przeszukiwania</u>                                                                                                  | 87  |
| c. <u>Kryptograficzne</u>                                                                                                 | 91  |
| 12. <u>Języki Programowania</u>        | 95  |
| 13. <u>Sieci Komputerowe</u>           | 101 |
| 14. <u>Protokoły Sieciowe</u>          | 113 |
| 15. <u>Grafika Komputerowa</u>         | 120 |
| 16. <u>Prawa Autorskie i Licencje</u>  | 129 |
| 17. <u>Wirusy Komputerowe</u>          | 134 |
| 18. <u>Odpowiedzi</u>                                                                                                     | 138 |





CHAPTER N.1

# Systemy liczbowe i konwersje



Systemy liczbowe, konwersje, operacje,  
typy zadań maturalnych



# Bit i Bajt

BIT:

Najmniejsza jednostka informacji w komputerze. Może przyjmować wartość **0** lub **1**

BAJT:

Zbiór 8 bitów.

**10101100**

## Systemy Liczbowe

### SYSTEM DZIESIĘTNY (DECYMALNY):

- Podstawa: 10
- Cyfry: 0-9
- Przykład: 123

$$\begin{array}{ccc} 1 \times 10^2 & 2 \times 10^1 & 3 \times 10^0 \\ \swarrow & \swarrow & \swarrow \\ \mathbf{1} & \mathbf{2} & \mathbf{3} \\ 10^2 & 10^1 & 10^0 \end{array}$$

### SYSTEM BINARNY (DWÓJKOWY):

- Podstawa: 2
- Cyfry: 0, 1
- Przykład: 1101

$$\begin{array}{cccc} 1 \times 2^3 & 1 \times 2^2 & 0 \times 2^1 & 1 \times 2^0 \\ \swarrow & \swarrow & \swarrow & \swarrow \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$

### SYSTEM SZESNASTKOWY (HEXADECYMALNY):

- Podstawa: 16
- Cyfry: 0-9, A-F (gdzie A=10, B=11, ..., F=15)
- Przykład: 1A3

$$\begin{array}{ccc} 1 \times 16^2 & \overset{10}{A} \times 16^1 & 3 \times 16^0 \\ \swarrow & \swarrow & \swarrow \\ \mathbf{1} & \mathbf{A} & \mathbf{3} \\ 16^2 & 16^1 & 16^0 \end{array}$$

### SYSTEM ÓSEMKOWY (OKTALNY)

- Podstawa: 8
- Cyfry: 0-7
- Przykład: 157

$$\begin{array}{ccc} 1 \times 8^2 & 5 \times 8^1 & 7 \times 8^0 \\ \swarrow & \swarrow & \swarrow \\ \mathbf{1} & \mathbf{5} & \mathbf{7} \\ 8^2 & 8^1 & 8^0 \end{array}$$

### SYSTEM DZIEWIĄTKOWY:

- Podstawa: 9
- Cyfry: 0-8
- Przykład: 168

$$\begin{array}{ccc} 1 \times 9^2 & 6 \times 9^1 & 8 \times 9^0 \\ \text{---} & \text{---} & \text{---} \\ \mathbf{1} & \mathbf{6} & \mathbf{8} \\ 9^2 & 9^1 & 9^0 \end{array}$$

### SYSTEM TRÓJKOWY:

- Podstawa: 3
- Cyfry: 0, 1, 2
- Przykład: 2120

$$\begin{array}{cccc} 2 \times 3^3 & 1 \times 3^2 & 2 \times 3^1 & 0 \times 3^0 \\ \text{---} & \text{---} & \text{---} & \text{---} \\ \mathbf{2} & \mathbf{1} & \mathbf{2} & \mathbf{0} \\ 3^3 & 3^2 & 3^1 & 3^0 \end{array}$$

## Wzór

System liczbowy o podstawie  $n$

$$\begin{array}{ccccccc} & & & m & & & \\ & & & \text{---} & & & \\ \mathbf{X} & \dots & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \\ n^{m-1} & \dots & n^3 & n^2 & n^1 & n^0 & \end{array}$$

Gdzie:  $X < n$

# Konwersje Między Systemami

## Dziesiętny na dowolny system

Metoda polega na dzieleniu z resztą liczby przez podstawę systemu, na który jest konwertowana, aż do 0, następnie odczytujemy nową liczbę z reszt od dołu do góry

Przykład:

Zamieniamy 13 na system binarny (dwójkowy)

$$\begin{array}{l} 13 : 2 = 6 \text{ } \sim 1 \\ 6 : 2 = 3 \text{ } \sim 0 \\ 3 : 2 = 1 \text{ } \sim 1 \\ 1 : 2 = 0 \text{ } \sim 1 \end{array} \uparrow \quad \begin{array}{l} \text{Wynik:} \\ \underline{\underline{1101}} \end{array}$$

Przykład 2:

Zamieniamy 761 na system szesnastkowy (hexodecymalny)

$$\begin{array}{l} 761 : 16 = 47 \text{ } \sim 9 \\ 47 : 16 = 2 \text{ } \sim 15 \\ 2 : 16 = 0 \text{ } \sim 2 \end{array} \uparrow \quad \begin{array}{l} \text{Wynik:} \\ \underline{\underline{2F9}} \end{array}$$

Przykład 3:

Zamieniamy 234 na system dziewiątkowy

$$\begin{array}{l} 234 : 9 = 26 \text{ } \sim 0 \\ 26 : 9 = 2 \text{ } \sim 8 \\ 2 : 9 = 0 \text{ } \sim 2 \end{array} \uparrow \quad \begin{array}{l} \text{Wynik:} \\ \underline{\underline{280}} \end{array}$$

## Ćwiczenie 1.1

- |                                |                                |
|--------------------------------|--------------------------------|
| a) 4380 na System piątkowy     | e) 789 na System binarny       |
| b) 345 na System binarny       | f) 9999 na System dziewiątkowy |
| c) 2048 na System szesnastkowy | g) 1234 na System trójkowy     |
| d) 5678 na System ósemkowy     | h) 4096 na System szóstkowy    |

# Dowolny system na dziesiętny

Metoda polega na sumowaniu iloczynu każdej cyfry z odpowiadającą jej potęgą.

Przykład:  
Zamieniamy  $1101_2$  na system dziesiętny

1101  
3 2 1 0

$$\begin{aligned} & 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 = \\ & = 1 \cdot 1 + 0 \cdot 2 + 1 \cdot 4 + 1 \cdot 8 = \\ & = 1 + 0 + 4 + 8 = \underline{\underline{13}} \end{aligned}$$

Przykład 2:  
Zamieniamy  $3FA_{16}$  na system dziesiętny

3FA  
2 1 0

$$\begin{aligned} & 3 \cdot 16^2 + F \cdot 16^1 + A \cdot 16^0 = \\ & = 3 \cdot 256 + 15 \cdot 16 + 10 \cdot 1 = \\ & = 768 + 240 + 10 = \underline{\underline{1018}} \end{aligned}$$

Przykład 3:  
Zamieniamy  $1324_5$  na system dziesiętny

1324  
3 2 1 0

$$\begin{aligned} & 1 \times 5^3 + 3 \times 5^2 + 2 \times 5^1 + 4 \times 5^0 = \\ & = 1 \times 125 + 3 \times 25 + 2 \times 5 + 4 \times 1 = \\ & = 125 + 75 + 10 + 4 = \underline{\underline{214}} \end{aligned}$$

Przykład 4:  
Zamieniamy  $754_8$  na system dziesiętny

754  
2 1 0

$$\begin{aligned} & 7 \times 8^2 + 5 \times 8^1 + 4 \times 8^0 = \\ & = 7 \times 64 + 5 \times 8 + 4 \times 1 = \\ & = 448 + 40 + 4 = \underline{\underline{492}} \end{aligned}$$

## Wskazówka

Podczas obliczeń na kalkulatorze prostym używaj jego pamięci:

- mr - wyświetla
- mc - usuwa
- m+ - dodaje

## Ćwiczenie 1.2

Zamień na system dziesiętny:

- a)  $110101_2$
- b)  $FFF_{16}$
- c)  $1267_8$
- d)  $432_5$

- e)  $B7_{12}$
- f)  $632_7$
- g)  $12G_{20}$
- h)  $1110100101_2$

# Konwersja Szybkim Sposobem

Metoda polega na **podziale liczby na grupy bitów**, gdzie długość grupy w jednym systemie odpowiada długości grupy bitów w drugim.

$$\underline{54}_{16} \rightarrow \underline{01010100}_2$$

Kiedy możemy stosować?

- Możemy zastosować szybką konwersję między dwoma systemami liczbowymi, jeśli istnieje **równość** między ich **podstawami podniesionymi do odpowiednich potęg**

$$X^n = Y^m$$

X - podstawa systemu nr 1

Y - podstawa systemu nr 2

$n, m \geq 1$

Przykład:

$$16^n = 2^m \rightarrow 16^1 = 2^4$$

Ustalanie długości grup bitowych:

- Na podstawie równania możemy łatwo określić długość grup bitowych. W powyższym przykładzie **jeden** znak w systemie **szesnastkowym** odpowiada **czterem** bitom w systemie **binarnym**.

Przedstawmy sobie **najczęściej występujące zamiany** (są one oczywiście odwracalne):

- **Potęgi dwójki:**

- Czwórkowy → Binarny

$$4^1 = 2^2 \rightarrow Y_4 = XX_2$$

- Ósemkowy → Binarny

$$8^1 = 2^3 \rightarrow Y_8 = XXX_2$$

- Szesnastkowy → Binarny

$$16^1 = 2^4 \rightarrow Y_{16} = XXXX_2$$

- **Potęga czwórki**

- Szesnastkowy → Czwórkowy

$$16^1 = 4^2 \rightarrow Y_{16} = XX_4$$

- **Potęga trójki**

- Dziewiątkowy → Trójkowy

$$9^1 = 3^2 \rightarrow Y_9 = XX_3$$

### Wyjaśnienie

Grupa bitów po lewej może zakodować tyle samo informacji co grupa bitów po prawej, mimo że należą do różnych systemów liczbowych.

Oczywiście istnieją inne konwersje, jednak **nie da** się ich przeprowadzić tak szybko w pamięci, a kluczowa jest dla nas prędkość. Przykładowo:

- **Ósemkowy → Czwórkowy**

$$8^2 = 4^3 \rightarrow YY_8 = XXX_4$$

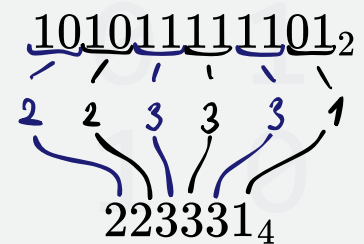
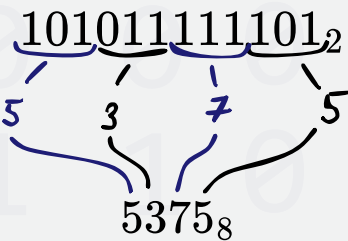
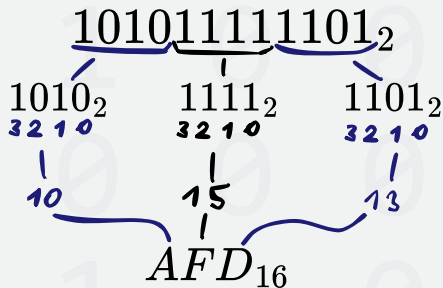
- **Szesnastkowy → Ósemkowy**

$$16^3 = 8^4 \rightarrow YYY_{16} = XXXX_8$$

# Konwersja Szybkim Sposobem

Przykład:

Zamieniamy  $101011111101_2$  na systemy 16, 8 i 4-owy.

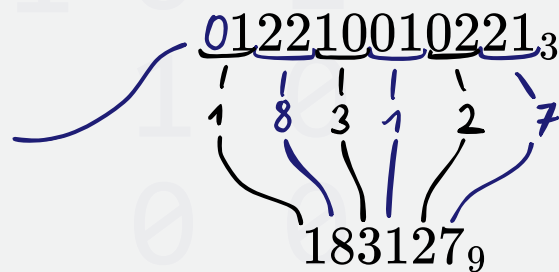


Przykład 2:

Zamieniamy  $12210010221_3$  na system 9

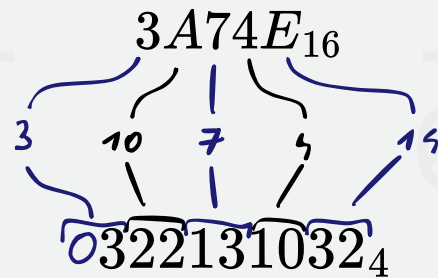
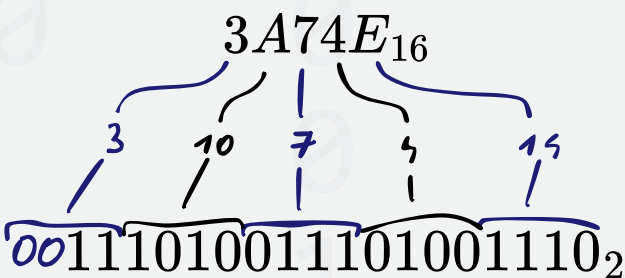
## Wskazówka

W sytuacji, gdy nie mamy pełnej grupy zawsze możemy dopisać z przodu zera, one nic nie zmieniają



Przykład 3:

Zamieniamy  $3A74E_{16}$  na system 2, 4 owy.



## Ćwiczenie 1.3

- a)  $101011_2$  zamień na systemy 16, 8 i 4
- b)  $2011_3$  zamień na system 9
- c)  $ADA_{16}$  zamień na systemy 2 i 4
- d)  $11010111_2$  zamień na systemy 16, 8 i 4
- e)  $3E7_{16}$  zamień na systemy 2 i 4
- f)  $512_8$  zamień na systemy 2
- g)  $1042_9$  zamień na system 3
- h)  $321_4$  zamień na system 16



# Operacje na tych samych systemach liczbowych

Skupimy się na operacjach takich jak **dodawanie** i **odejmowanie**. Najprościej, tak jak w systemie dziesiętnym, robi się to pod kreską. Tutaj również wykorzystamy ten sposób jednakże z małą różnicą. Będziemy mieć nowy zakres cyfr.

## Operacje na Systemie Binarnym

### Zasada:

Dodajemy cyfry w kolumnach od prawej do lewej, przenosząc nadmiar ( $1 + 1 = 10$ , czyli zapisujemy 0 i przenosimy 1) do następnej kolumny.

### Przykład:

•  $1011_2 + 1101_2$

1.

$$\begin{array}{r} 1011 \\ + 1101 \\ \hline \end{array}$$

2.

$$\begin{array}{r} 10\overset{1}{1}1 \\ + 1101 \\ \hline 0 \end{array}$$

3.

$$\begin{array}{r} 1\overset{1}{0}\overset{1}{1}1 \\ + 1101 \\ \hline 00 \end{array}$$

4.

$$\begin{array}{r} 1\overset{1}{0}\overset{1}{1}1 \\ + 1101 \\ \hline 000 \end{array}$$

5.

$$\begin{array}{r} 1\overset{1}{0}\overset{1}{1}1 \\ + 1101 \\ \hline 11000 \end{array}$$

### Zasada:

Odejmujemy cyfry w kolumnach od prawej do lewej, pożyczając jeśli trzeba ( $1 - 1 = 0$ ,  $0 - 1 = 1$  z pożyczaniem).

### Przykład:

•  $1011_2 - 110_2$

1.

$$\begin{array}{r} 1011 \\ - 110 \\ \hline \end{array}$$

2.

$$\begin{array}{r} 1011 \\ - 110 \\ \hline 1 \end{array}$$

3.

$$\begin{array}{r} 1011 \\ - 110 \\ \hline 01 \end{array}$$

4.

$$\begin{array}{r} 0\overset{2}{1}011 \\ - 110 \\ \hline 01 \end{array}$$

5.

$$\begin{array}{r} 0\overset{2}{1}011 \\ - 110 \\ \hline 101 \end{array}$$

## Ćwiczenie 1.4

a)  $100101_2 + 1001011_2$

b)  $1111_2 + 1111_2$

c)  $100110_2 + 110101_2$

d)  $100110_2 - 100001_2$

e)  $1111_2 - 111_2$

f)  $101010_2 - 101_2$

# Operacje na Systemie Szesnastkowym

## Zasada:

Dodajemy cyfry w kolumnach od prawej do lewej, przenosząc nadmiar do następnej kolumny (np.  $A + 5 = F$ ,  $9 + 1 = 10 \rightarrow$  zapisujemy 0 i przenosimy 1).

## Przykład:

•  $1A3_{16} + B2_{16}$

1. 
$$\begin{array}{r} 1A3 \\ + B2 \\ \hline \end{array}$$

2. 
$$\begin{array}{r} 1A3 \\ + B2 \\ \hline 5 \end{array}$$

3. 
$$\begin{array}{r} 10+11=21 \\ 21/16=1 \text{ r } 5 \\ \uparrow \\ 1A3 \\ + B2 \\ \hline 5 \end{array}$$

4. 
$$\begin{array}{r} 1A3 \\ + B2 \\ \hline 55 \end{array}$$

5. 
$$\begin{array}{r} 1A3 \\ + B2 \\ \hline 255 \end{array}$$

## Zasada:

Odejmujemy cyfry w kolumnach od prawej do lewej, pożyczając jeśli trzeba z następnej kolumny.

## Przykład:

•  $1A3_{16} - B2_{16}$

1. 
$$\begin{array}{r} 1A3 \\ - B2 \\ \hline \end{array}$$

2. 
$$\begin{array}{r} 1A3 \\ - B2 \\ \hline 1 \end{array}$$

3. 
$$\begin{array}{r} 016 \\ \swarrow \\ 1A3 \\ - B2 \\ \hline 1 \end{array}$$

4. 
$$\begin{array}{r} 16+10-11= \\ = 26-11=15=F \\ \uparrow \\ 016 \\ \swarrow \\ 1A3 \\ - B2 \\ \hline 1 \end{array}$$

5. 
$$\begin{array}{r} 016 \\ \swarrow \\ 1A3 \\ - B2 \\ \hline F1 \end{array}$$

# Operacje na Różnych Systemach Liczbowych

a)  $567_8 + 245_8$

$$\begin{array}{r} 11 \\ 567 \\ + 245 \\ \hline 1034 \end{array}$$

b)  $251_7 + 121_7$

$$\begin{array}{r} 1 \\ 251 \\ + 121 \\ \hline 402 \end{array}$$

c)  $545_8 - 267_8$

$$\begin{array}{r} 1378 \\ 545 \\ - 267 \\ \hline 236 \end{array}$$

d)  $1212_3 - 222_3$

$$\begin{array}{r} 123 \\ 1212 \\ - 222 \\ \hline 220 \end{array}$$

## Ćwiczenie 1.5

a)  $101_3 + 202_3$

b)  $123_4 + 321_4$

c)  $234_5 + 321_5$

d)  $543_6 + 345_6$

e)  $726_8 + 652_8$

f)  $312_9 + 231_9$

g)  $3E4_{16} + 1C5_{16}$

h)  $210202_3 - 102_3$

i)  $1321_4 - 123_4$

j)  $432_5 - 234_5$

k)  $654_6 - 453_6$

l)  $1565_8 - 767_8$

m)  $43112_9 - 234_9$

n)  $5A9_{16} - 1BF_{16}$

# OCEŃ PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/LJjE49gBC4svDxQ48>



## CHAPTER N.2

# System U2



Reprezentacja liczb, Właściwości,  
Typy zadań maturalnych

10011010  
00101101

# U2

## Definicja:

Kod U2, czyli uzupełnienie do dwóch, jest sposobem reprezentacji liczb całkowitych (zarówno dodatnich, jak i ujemnych) w systemie binarnym.

Jest powszechnie stosowany w komputerach do przechowywania i przetwarzania liczb całkowitych, ponieważ umożliwia łatwe wykonywanie operacji arytmetycznych.



# Reprezentacja liczb

## LICZBY DODATNIE

- Reprezentacja jest taka sama jak w zwykłym systemie binarnym.
- Przykładowo liczba **+5 w 8-bitowym** systemie binarnym to:  $00000101_2$

## LICZBY UJEMNE

- Reprezentacja liczb ujemnych w U2 polega na **odwróceniu** wszystkich bitów liczby dodatniej i **dodaniu 1**.
- Przykładowo liczba **-5 w 8-bitowym** systemie binarnym:
  - Najpierw reprezentujemy +5:

$00000101_2$

- Następnie **odwracamy** bity:

$11111010$

- Ostatnim krokiem jest **dodanie 1**:

$$11111010 + 1 = 11111011_{U2}$$

# Właściwości Systemu U2

## Jednoznaczność reprezentacji:

- Każda liczba ma dokładnie jedną reprezentację w systemie U2.

## Zakres liczb:

- Dla systemu n-bitowego zakres reprezentowanych liczb wynosi

$$\text{od } -2^{n-1} \text{ do } 2^{n-1} - 1$$

- Przykładowo, dla 8-bitowego systemu:

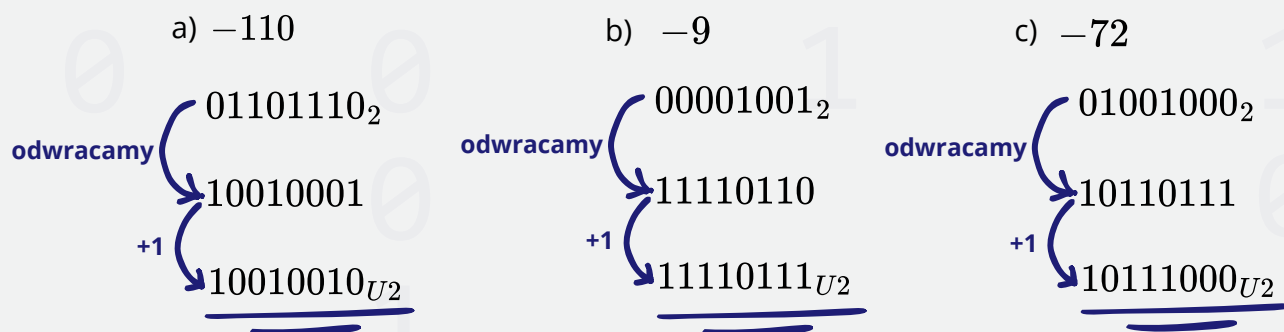
$$\text{od } -128 \text{ do } +127$$

- Dla 16-bitowego systemu:

$$\text{od } -32768 \text{ do } +32767$$

# Zamiana Liczb

## PRZEDSTAWINIE UJEMNYCH LICZB



## Ćwiczenie 2.1

Zamień na U2:

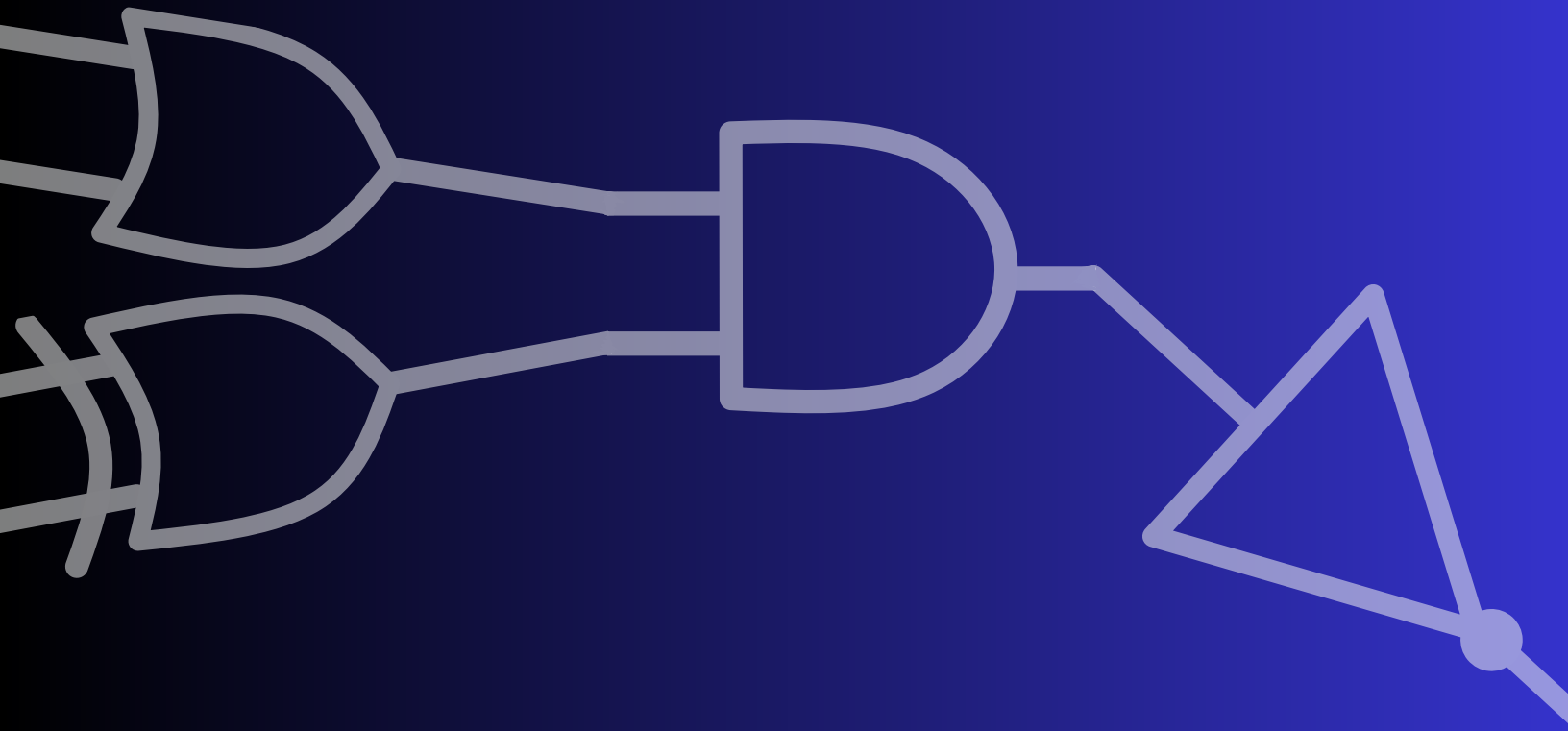
- a)  $-46$
- b)  $-123$
- c)  $-14$

- d)  $-201$
- e)  $-56$
- f)  $-128$

# OCEŃ PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/aNvBjVzfbevtgoJPA>



CHAPTER N.3

# Bramki Logiczne



Definicja, Podstawowe i Złożone Bramki Logiczne, Typy zadań maturalnych

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



# Bramka Logiczna

## Definicje:

**Bramka logiczna** to element elektroniczny w układach cyfrowych, który realizuje operacje logiczne (AND, OR, NOT) na sygnałach wejściowych, dając sygnał wyjściowy. Bramki logiczne są fundamentalnymi blokami budującymi układy cyfrowe.

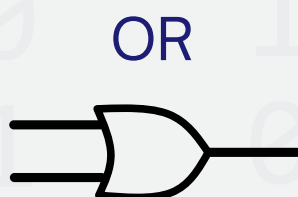
**Tabela prawdy** to narzędzie używane do opisu funkcji bramki, przedstawiające wszystkie możliwe kombinacje sygnałów wejściowych i odpowiadających im sygnałów wyjściowych.

## Podstawowe Typy Bramek



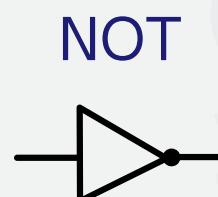
**Wysoki** wyjściowy, tylko gdy **wszystkie** wejścia są **wysokie**.

| A | B | y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |



**Wysoki** wyjściowy, gdy **jakiegokolwiek** wejście jest **wysokie**.

| A | B | y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |



**Odwraca** sygnał wejściowy

| A | y |
|---|---|
| 1 | 0 |
| 0 | 1 |

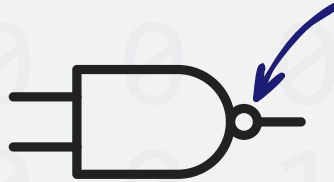
1 oznacza stan **wysoki**  
0 oznacza stan **niski**

Reszta bramek logicznych są stworzone z kombinacji tych 3 powyższych struktur.

Na szczęście nie musisz uczyć się tabel prawdy każdej bramki na pamięć. Na egzaminie są one podane, chce po prostu byś miał/miała wiedzę o nich i by ten temat nie był ci obcy.

# Złożone Typy Bramek

NAND



**Niski** wyjściowy,  
tylko gdy **wszystkie**  
wejścia są **wysokie**.

| A | B | Y |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

NOR



**Niski** wyjściowy, gdy  
**jakikolwiek** wejście  
jest **wysokie**.

| A | B | Y |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

XOR



**Wysoki** wyjściowy,  
gdy wejścia są **różne**.

| A | B | Y |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

XNOR



**Wysoki** wyjściowy,  
gdy wejścia są **takie**  
**same**.

| A | B | Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

# Zadania na Bramkach Logicznych

Zadania maturalne polegają na stosowaniu bramki logicznej między bitami dwóch liczb binarnych. W przypadku większej liczby liczb, operacje wykonuje się po kolei od lewej. Tabela bramek logicznych zawsze będzie podana.

Wyrażenie może wyglądać w ten sposób:

- 1001010 NOR 1110011

Po kolejności sprawdzamy bity  
względem bramki NOR

$$\begin{array}{r} 1001010 \\ \text{NOR } 1110011 \\ \hline \end{array}$$

$$\begin{array}{r} 1001010 \\ \text{NOR } 1110011 \\ \hline 0 \end{array}$$

$$\begin{array}{r} 1001010 \\ \text{NOR } 1110011 \\ \hline 0000100 \end{array}$$

## Ćwiczenie 3.1

Wykonaj operacje wykorzystując bramki logiczne:

- a) 110011 AND 101010
- b) 011001 OR 100110
- c) 101010 XOR 110101
- d) 11110000 NOT
- e) 110011 NAND 101101

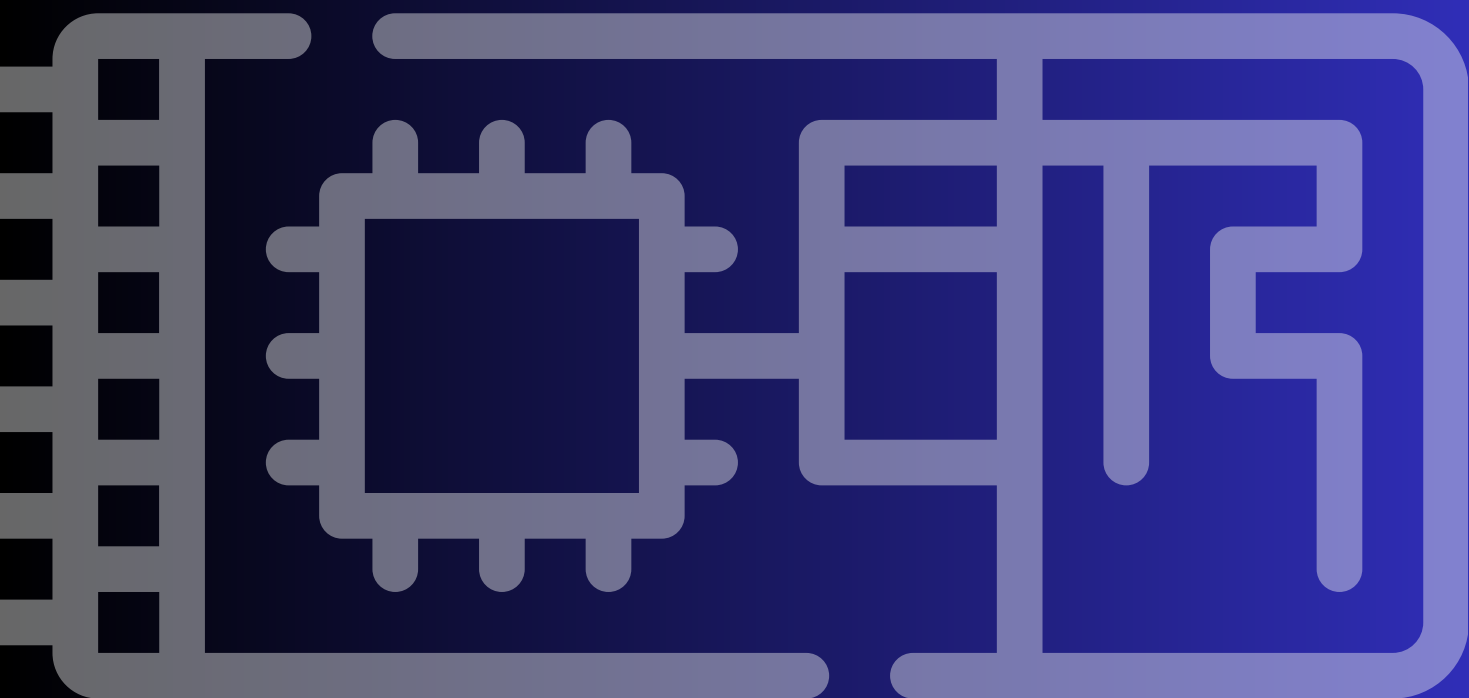
- f) 100110 NOR 011011
- g) (101001 NOT) AND 110110
- h) 100101 XNOR 111011
- i) ((111000 AND 100011) OR 100011) NOT
- j) (110101 XOR 011010) NAND (110101 NOT)

# OCENIĆ PROSZĘ TEN ROZDZIAŁ



**InfaMatura**

<https://forms.gle/aDxNePMhrjYD6Ycn7>

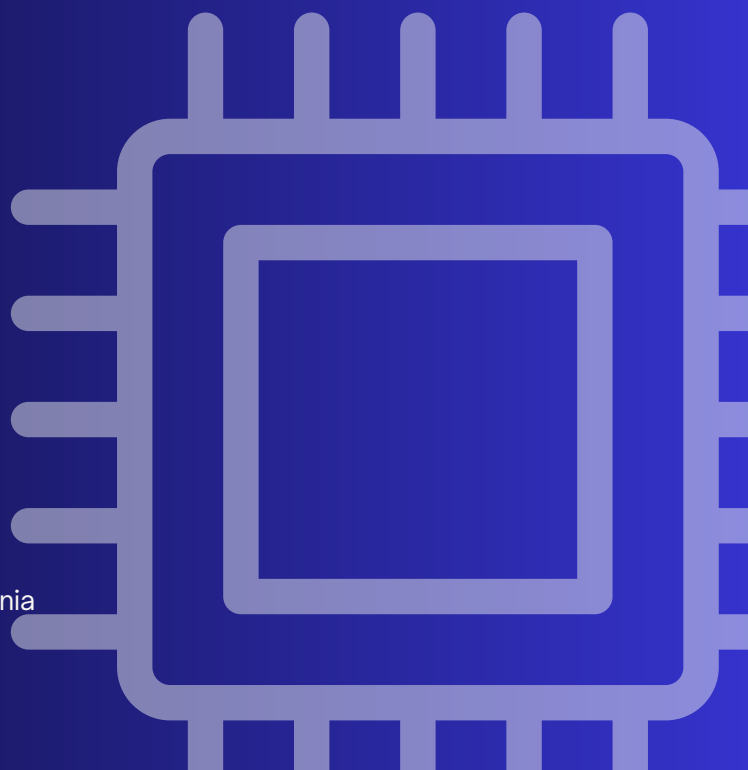


CHAPTER N.4

# Architektura Komputerowa



Podzespoły Komputera, Magistrale, Urządzenia  
Peryferyjne, Typy zadań maturalnych



# Podzespoły Komputera

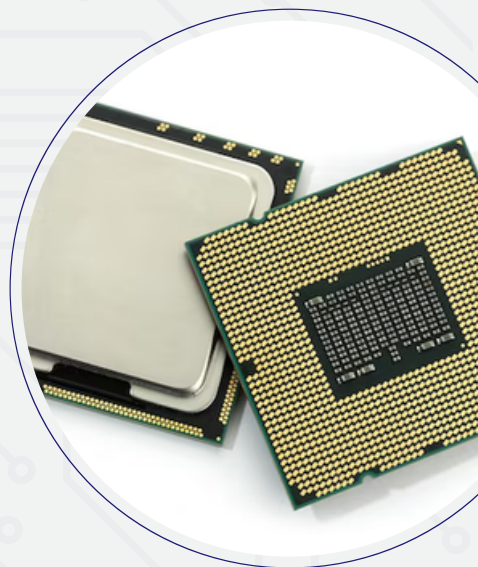
## CPU (CENTRAL PROCESSING UNIT):

### Procesor:

Kluczowy element komputera, który zarządza wszystkimi operacjami w systemie.

Składający się z:

- **Rdzeni** wykonujących instrukcje programów.
  - Każdy rdzeń obsługuje więcej niż jeden **wątek** dzięki czemu równocześnie może wykonywać wiele zadań.
- **Taktowanie:** Częstotliwość pracy procesora w gigahercach (GHz), która wpływa na szybkość przetwarzania instrukcji i ogólną wydajność
- **Pamięci podręcznej (cache)**, która przyspiesza dostęp do najczęściej używanych danych.



## GPU (GRAPHICS PROCESSING UNIT):

### Karta Graficzna:

Układ scalony zaprojektowanym do szybkiego, efektywnego i równoległego przetwarzania dużych bloków danych związanych z grafiką i wideo.

Składą się z:

- setek lub tysięcy mniejszych rdzeni, które równocześnie wykonują obliczenia graficzne.
- posiada pamięć **VRAM** do szybkiego przechowywania i dostępu do danych graficznych.



# RAM (RANDOM ACCESS MEMORY):

## Pamięć operacyjna:

Moduły pamięci RAM, które tymczasowo przechowują dane i programy w trakcie ich wykonywania przez procesor, są kluczowe dla szybkości i płynności systemu.

Składa się z:

- wielu komórek pamięci, z których każda może być szybko odczytana lub zapisana niezależnie od pozycji innych komórek
- **tranzystorów i kondensatorów** w układach **DRAM**,  
lub
- **samych tranzystorów** w układach **SRAM**, które są szybsze, ale droższe i mniej pojemne niż DRAM.



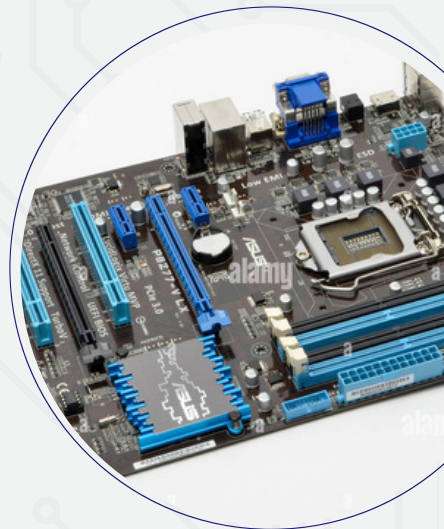
# MOTHERBOARD

## Płyta Główna:

Płyta drukowana, która łączy wszystkie komponenty komputera za pomocą szyn danych i zasilania.

Zawiera:

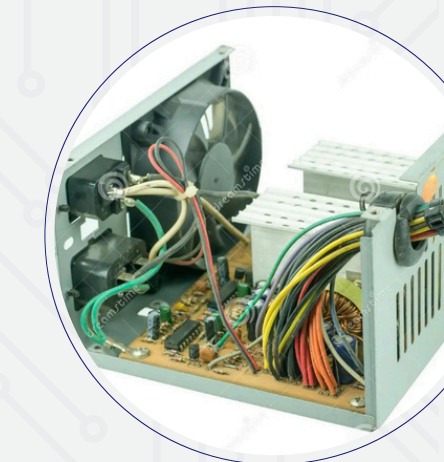
- gniazda dla CPU,
- sloty dla RAM,
- porty rozszerzeń dla GPU,
- Porty SATA / NVMe,
- System audio,
- Zintegrowana karta sieciowa,
- Złącza In/Out



# PC POWER SUPPLY

## Zasilacz:

- Konwertuje prąd zmienny (AC) na prąd stały (DC), dostarczając stabilne napięcie do każdego komponentu,
- Chroni system przed niestabilnościami prądu.





## PAMIĘĆ HDD (HARD DISK DRIVE):

### Dysk twardy:

Zawiera **ruchome talerze** pokryte materiałem magnetycznym, na których zapisywane są dane.

Główce odczytu/zapisu przemieszczają się nad talerzami, odczytując lub zapisując dane.

Są używane do zapisów i przechowywania bardzo dużych ilości danych.

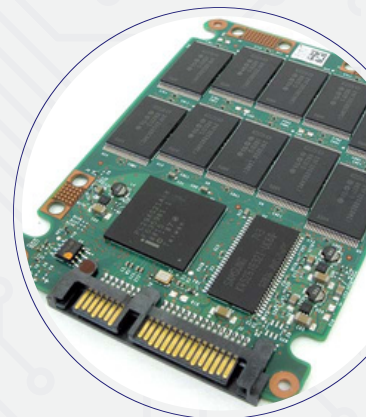


## PAMIĘĆ SSD (SOLID STATE DRIVE):

### Dysk SSD:

Szybki nośnik danych, który korzysta z pamięci **flash (NAND)** do zapisywania informacji, oferujący szybsze czasy dostępu

Nie posiada ruchomych części, co zmniejsza ryzyko uszkodzeń mechanicznych.



## NVMe (Non-Volatile Memory Express):

### Dysk SSD NVMe:

Specyfikacja interfejsu dla SSD, która wykorzystuje magistralę PCIe w komputerze do przyspieszenia procesów odczytu/zapisu, znacznie zwiększając wydajność w porównaniu do starszych interfejsów takich jak SATA.

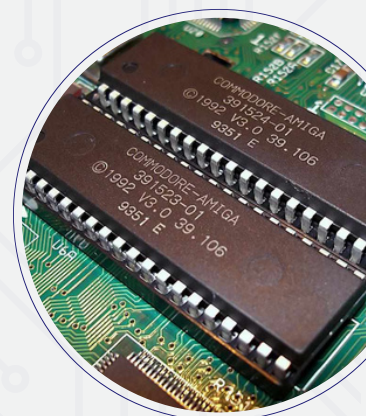


## ROM (READ-ONLY MEMORY)

### Pamięć tylko do odczytu:

Pamięć używana do przechowywania oprogramowania, które jest niezbędne do uruchamiania komputera, jak firmware czy BIOS.

ROM nie może być modyfikowany podczas normalnej operacji, co zapewnia stałą ochronę zawartości.





# INTERFEJST

Zestaw standardów i protokołów umożliwiających komunikację między różnymi komponentami komputerowymi, określający zarówno fizyczne, jak i elektroniczne aspekty połączeń.

## MAGISTRALA

System przesyłania danych wewnątrz komputera, umożliwiający wymianę informacji między procesorem, pamięcią i innymi komponentami.

## PCI (Peripheral Component Interconnect):

**Magistrala** używana do podłączania kart rozszerzeń, takich jak karty graficzne czy sieciowe, do płyty głównej.



## USB (Universal Serial Bus):

Uniwersalny **interfejs** służący do podłączania różnych urządzeń zewnętrznych, takich jak pamięci flash, drukarki, kamery.

## SATA (Serial ATA):

**Interfejs** używany do podłączania dysków twardych i napędów optycznych, zapewniający szybki transfer danych.



## Thunderbolt:

Wysokoprędkościowy **interfejs** umożliwiający połączenie wielu urządzeń zewnętrznych i monitorów przez pojedyncze złącze, oferuje również dostarczanie energii.

## HDMI (High-Definition Multimedia Interface):

**Interfejs** służący do przesyłania wideo i audio w wysokiej rozdzielczości między urządzeniami, np. od komputera do monitora czy telewizora.



## DP (DisplayPort):

Zaawansowany **interfejs** do przesyłania sygnałów wideo i audio, często używany do podłączania monitorów komputerowych i profesjonalnych systemów wizyjnych.

## Jack (audio jack):

Jest to **interfejs** audio używany do przesyłania sygnałów analogowych, służący do podłączania słuchawek, mikrofonów, oraz innych urządzeń audio.



# Urządzenia Peryferyjne

## Urządzenie peryferyjne:

Zewnętrzny komponent komputerowy podłączany do komputera, który rozszerza jego funkcjonalność lub umożliwia komunikację z użytkownikiem. Urządzenia te mogą być podłączane do komputera na różne sposoby, w tym przez porty USB, Bluetooth, Wi-Fi, czy inne interfejsy.

## PRZYKŁADY URZĄDZEŃ PERYFERYJNYCH:

### Urządzenia wejścia:

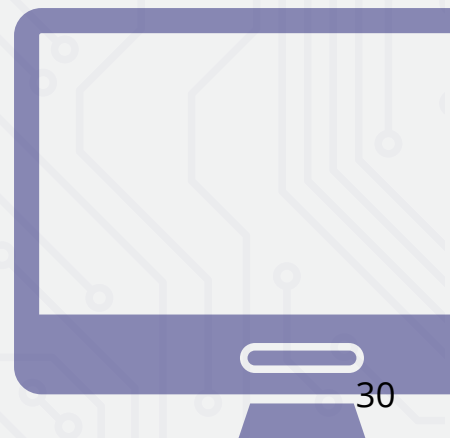
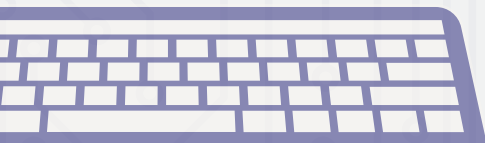
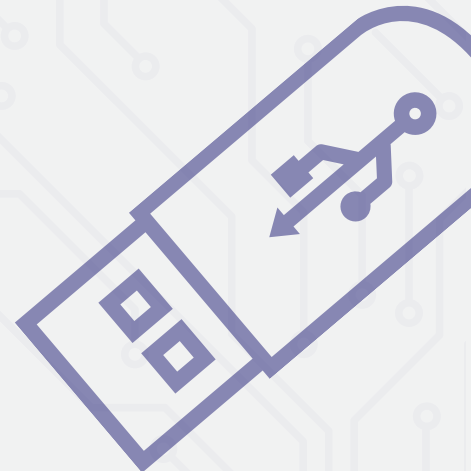
- Klawiatura
- Mysz
- Skaner
- Mikrofon
- Kamera internetowa
- Gamepad/Joystick
- Czytnik kart pamięci
- Tablet graficzny

### Urządzenia wyjścia:

- Monitor
- Drukarka
- Głośniki
- Słuchawki
- Projektor

### Urządzenia wejścia i wyjścia:

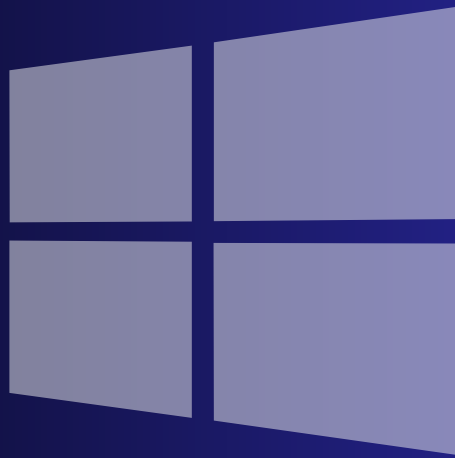
- Monitor dotykowy
- Dyski zewnętrzne (np. PenDrive, SSD)
- Słuchawki z mikrofonem
- Karty sieciowe
- Projektor Napędy optyczne (CD/DVD/Blu-ray)
- Multifunkcyjne urządzenia biurowe (drukarka, skaner, kopiarka w jednym)



OCENIĆ PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/eSLiYXC1Rhdd4gCXA>

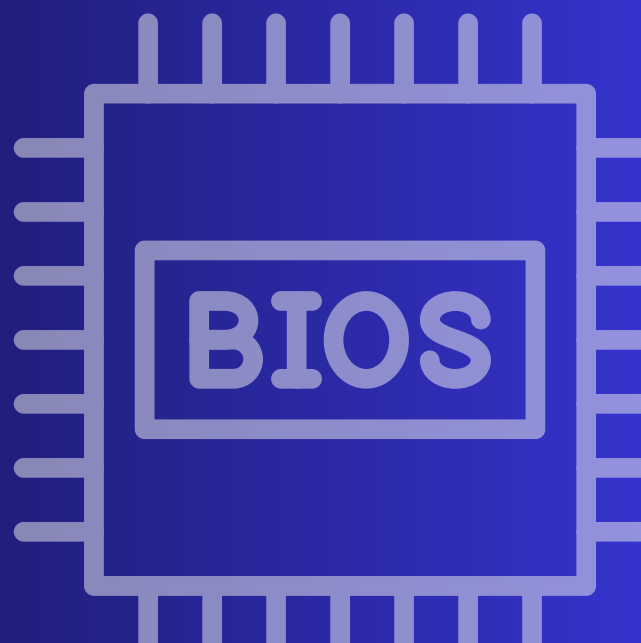


CHAPTER N.5

# Architektura Oprogramowania



Systemy wbudowane i operacyjne, Pliki,  
Typy zadań naturalnych



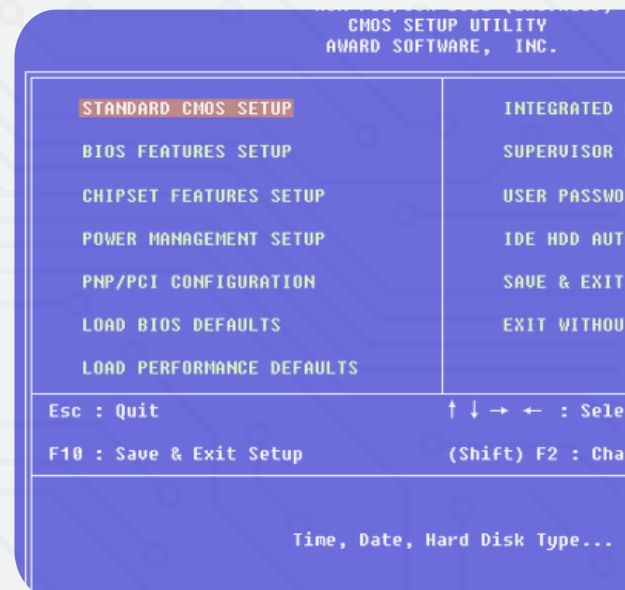
# Składniki Architektury Oprogramowania:

## FIRMWARE:

Niskopoziomowe oprogramowanie zapisane na stałe w sprzęcie, które inicjuje sprzęt i zapewnia podstawowe funkcje systemowe przed uruchomieniem systemu operacyjnego.

### BIOS:

Niskopoziomowe oprogramowanie zapisane w pamięci ROM lub flash ROM na płycie głównej, które inicjuje i testuje sprzęt komputerowy podczas uruchamiania. Pełni funkcję pośrednika między sprzętem a systemem operacyjnym w fazie bootowania komputera.



## SYSTEMY WBUDOWANE:

Specjalizowane oprogramowanie zaprojektowane do wykonywania ograniczonej liczby funkcji w ramach większego systemu, często o wysokim stopniu integracji z aparatem sprzętowym.

## SYSTEMY OPERACYJNE:

Oprogramowanie zarządzające zasobami komputerowymi, koordynujące działanie sprzętu i oprogramowania użytkowego, i zapewniające interfejs użytkownika do interakcji z komputerem. System operacyjny jest przechowywany na dysku twardym komputera.

Najbardziej znanymi systemami są:

- Windows
- Linux
- macOS



# Pliki

## Rozszerzenia Plików:

### GRAFIKA I OBRAZY

#### Grafika Rastrowa:

- TIFF (.tiff)
- BMP (.bmp)
- JPG (.jpg lub .jpeg)
- PNG (.png)
- GIF (.gif)
- RAW (.raw)

#### Grafika Wektorowa:

- SVG (.svg)

### AUDIO

- OGG (.ogg)
- WMA (.wma)
- WAV (.wav)
- MIDI (.midi)
- MP3 (.mp3)
- FLAC (.flac)
- AAC (.aac)

### VIDEO

- AVI (.avi)
- MOV (.mov)
- MPEG (.mpeg)
- MP4 (.mp4)
- MKV (.mkv)
- WMV (.wmv)
- FLV (.flv)

### DOKUMENTY

- OCR (.ocr)
- PDF (.pdf)
- DOC/DOCX (.doc/.docx)
- TXT (.txt)
- RTF (.rtf)

### KOMPRESJA

- RAR (.rar)
- ZIP (.zip)
- 7Z (.7z)



# Rozmiary Plików

| Nazwa     | Skrót | Ilość w poprzedniej jednostce | Ilość w Bajtach                                |
|-----------|-------|-------------------------------|------------------------------------------------|
| Bit       | b     | N/A                           | 1/8 (0.125 Bajta)                              |
| Bajt      | B     | 8 bitów                       | 1                                              |
| Kilobajt  | KB    | 1,024 Bajty                   | 1,024 ( $2^{10}$ ).                            |
| Megabajt  | MB    | 1,024 Kilobajty               | 1,048,576 ( $2^{20}$ ).                        |
| Gigabajt  | GB    | 1,024 Megabajty               | 1,073,741,824 ( $2^{30}$ ).                    |
| Terabajt  | TB    | 1,024 Gigabajty               | 1,099,511,627,776 ( $2^{40}$ ).                |
| Petabajt  | PB    | 1,024 Terabajty               | 1,125,899,906,842,624 ( $2^{50}$ )             |
| Eksabajt  | EB    | 1,024 Petabajty               | 1,152,921,504,606,846,976 ( $2^{60}$ )         |
| Zettabajt | ZB    | 1,024 Eksabajty               | 1,180,591,620,717,411,303,424 ( $2^{70}$ )     |
| Jottabajt | YB    | 1,024 Zettabajty              | 1,208,925,819,614,629,174,706,176 ( $2^{80}$ ) |

## Systemy Plików

### FAT32

- **Opis:** Starszy system plików używany głównie w wymiennych nośnikach danych.
- **Zalety:** Wysoka kompatybilność między różnymi systemami operacyjnymi.
- **Ograniczenia:** Maksymalny rozmiar pliku to 4 GB, a woluminu 8 TB.
- **Użycie:** Pendrive'y, karty pamięci.

### NTFS

- **Opis:** Zaawansowany system plików stosowany w systemach Windows.
- **Zalety:** Wsparcie dla dużych plików, zaawansowane funkcje bezpieczeństwa i optymalizacji.
- **Ograniczenia:** Ograniczona kompatybilność z innymi systemami bez specjalnego oprogramowania.
- **Użycie:** Wewnętrzne dyski twarde w komputerach z Windows

OCENIĆ PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/hASSxoDiGcrUD7RJ8>





CHAPTER N.6

# Stos, Sterta i Kolejka



Wytłumaczenie Zagadnień, ONP, Podstawowe operacje, Implementacje i Zastosowania, Zadania Maturalne

# STOS

## Definicja:

Stos to **struktura danych** opierająca się na zasadzie "**ostatni na wejściu, pierwszy na wyjściu**" (**LIFO** - Last In, First Out).

To oznacza, że elementy są dodawane i usuwane z tego **samego końca struktury**, zwanego "szczytem" stosu

## (LIFO - LAST IN, FIRST OUT)

W stosie ostatni element, który został **dodany** (pushed), jest **pierwszym**, który zostanie **usunięty** (popped). To sprawia, że stos jest idealny do zadań, które wymagają odwrócenia kolejności elementów lub zapamiętywania stanów.



## PODSTAWOWE OPERACJE NA STOSIE:

### push (dodawanie elementu)

Operacja **push** dodaje nowy element na szczyt stosu, dzięki czemu staje się on automatycznie pierwszym.

### pop (usuwanie elementu)

Operacja **pop** usuwa element ze szczytu stosu i może zwracać jego wartość.

### peek/top (podglądanie elementu na szczycie)

Funkcja **peek (lub top)** pozwala spojrzeć na element znajdujący się na szczycie stosu bez jego usuwania.

### isEmpty (sprawdzanie, czy stos jest pusty)

Funkcja **isEmpty** sprawdza, czy stos jest pusty. Jeśli tak, zwraca prawdę; jeśli stos zawiera jakiegokolwiek elementy, zwraca fałsz.

## IMPLEMENTACJA STOSU:

### Implementacja stosu za pomocą tablicy

Stos można zaimplementować przy użyciu tablicy, gdzie **indeks końcowy** tablicy jest **szczytem**, który jest używany do dodawania (push) i usuwania (pop) elementów.

Operacje te są zwykle wykonywane w czasie statym ( $O(1)$ ), co czyni je bardzo efektywnymi.

## ZASTOSOWANIA STOSU:

- Sprawdzanie poprawności nawiasów w wyrażeniu
- Rekurencja i stos wywołań funkcji
- Implementacja funkcji cofnij (undo) w edytorach tekstu i grafiki.
- Konwersja wyrażeń infiksowych na postfiksowe (ONP) i prefiksowe

## ONP (Odwrotna Notacja Polska)

Definicja:

**Odwrotna Notacja Polska (ONP)**, znana również jako **notacja postfiksowa**, jest sposobem zapisu wyrażeń arytmetycznych, w którym operatory są umieszczane po swoich operandach. Dzięki temu unika się potrzeby używania nawiasów do określania kolejności operacji.

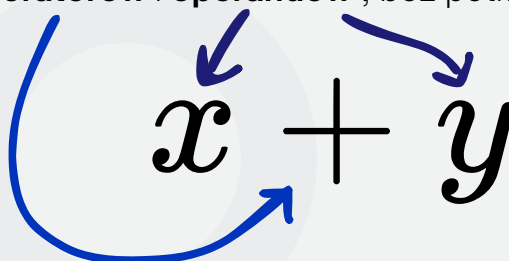
## ZASADY ONP:

### Operandy poprzedzają operatory:

Zamiast zapisu **infiksowego** (np.  $a+b$ ), używamy **postfiksowego** (np.  $ab+$ ).

### Kolejność operacji wynika z kolejności operandów i operatorów:

Kolejność wykonywania operacji jest determinowana przez kolejność pojawiania się **operatorów** i **operandów**, bez potrzeby stosowania nawiasów.



## PRZYKŁADY:

### Proste wyrażenie:

- Infiks: **a-b**
- Postfiks (ONP): **ab-**

### Wyrażenie z kilkoma operatorami:

- Infiks: **a\*b+c**
- Postfiks (ONP): **cab\*+**
- Infiks: **a+b\*c**
- Postfiks (ONP): **abc\*+**

### Wyrażenie z nawiasami:

- Infiks: **(a+b)\*c**
- Postfiks (ONP): **ab+c\***

## WYKONYWANIE OBLICZEŃ Z UŻYCIEM ONP:

Do obliczania wartości wyrażeń zapisanych w ONP używa się stosu:

1. Inicjalizacja:
  - Utwórz pusty stos.
2. Przechodzenie przez wyrażenie postfiksowe:
  - Jeśli napotkasz **operand**, **push** go na stos.
  - Jeśli napotkasz **operator**, **pop dwie wartości** ze stosu, wykonaj operację i push wynik z powrotem na stos.
3. Po przetworzeniu wyrażenia:
  - Wynik końcowy będzie na wierzchu stosu.

## Wyrażenie: $34+2 \times 7/$

1. Inicjalizacja

$34+2 \times 7/$       **STOS**

2.1. Jeśli napotkasz operand, push go na stos.

3 $4+2 \times 7/$        $\begin{matrix} 3 \\ \text{STOS} \end{matrix}$

2.1. Jeśli napotkasz operand, push go na stos.

4 $+2 \times 7/$        $\begin{matrix} 4 \\ 3 \\ \text{STOS} \end{matrix}$

2.2. Jeśli napotkasz operator, pop dwie wartości ze stosu, wykonaj operację i push wynik z powrotem na stos.

+ $2 \times 7/$        $\begin{matrix} 4 \\ 3 \\ \text{STOS} \end{matrix} \rightarrow 3 + 4 = 7 \rightarrow \begin{matrix} 7 \\ \text{STOS} \end{matrix}$

2.1. Jeśli napotkasz operand, push go na stos.

2 $\times 7/$        $\begin{matrix} 2 \\ 7 \\ \text{STOS} \end{matrix}$

2.2. Jeśli napotkasz operator, pop dwie wartości ze stosu, wykonaj operację i push wynik z powrotem na stos.

\* $7/$        $\begin{matrix} 2 \\ 7 \\ \text{STOS} \end{matrix} \rightarrow 7 \times 2 = 14 \rightarrow \begin{matrix} 14 \\ \text{STOS} \end{matrix}$

2.1. Jeśli napotkasz operand, push go na stos.

7 $/$        $\begin{matrix} 7 \\ 14 \\ \text{STOS} \end{matrix}$

2.2. Jeśli napotkasz operator, pop dwie wartości ze stosu, wykonaj operację i push wynik z powrotem na stos.

/       $\begin{matrix} 7 \\ 14 \\ \text{STOS} \end{matrix} \rightarrow 14 \div 7 = 2 \rightarrow \begin{matrix} 2 \\ \text{WYNIK} \end{matrix}$

## Ćwiczenie 6.1

Oblicz przy pomocy ONP:

a)  $3 \cdot 4 +$

b)  $5 \cdot 2 + 8 \cdot$

c)  $6 \cdot 2 + 1 \cdot 1 + /$

d)  $10 \cdot 4 - 2 \cdot 3 + \times$

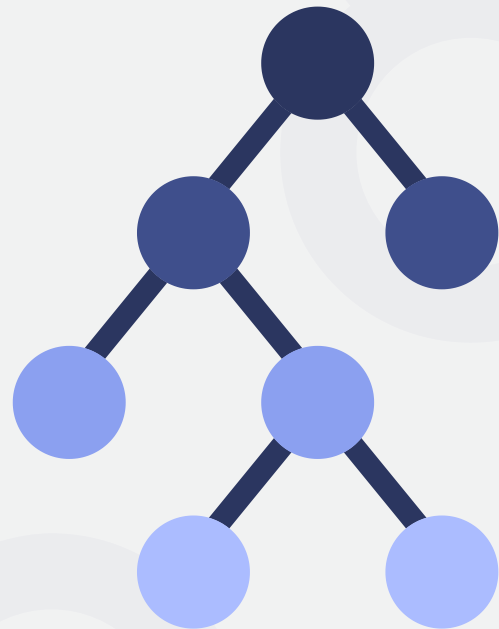
e)  $100 \cdot 10 \cdot 5 - 2 \cdot \times /$

f)  $3 \cdot 5 + 10 \cdot 2 - \times 4 /$

# STERTA

### Definicja:

Sterna to struktura danych, gdzie elementy są organizowane w sposób **hierarchiczny** - największy lub najmniejszy element znajduje się na górze. Dodatkowo każdy element można w dowolnym momencie modyfikować.



## RODZAJE STERT

- **Min-heap:** Najmniejszy element na wierzchu.
- **Max-heap:** Największy element na wierzchu.

## PODSTAWOWE OPERACJE NA STERCIE:

- **insert:** Dodaje nowy element.
- **delete:** Usuwa element z wierzchu.
- **extractMin/extractMax:** Zwraca i usuwa element z wierzchu.
- **heapify:** Przywraca właściwą strukturę sterty po zmianach

## ZASTOSOWANIA STERTY:

- **Zarządzanie pamięcią:** Używana do alokacji pamięci w programach.

Informacje o sterce są dodatkowe, nie powinna się ona pojawić na maturze. Jednakże wspominam o niej, bo trudno ją pominąć jeśli już odnosimy się do stosu i kolejki

# KOLEJKA

## Definicja:

Kolejka to struktura danych, w której elementy są przechowywane i usuwane zgodnie z zasadą pierwszeństwa: **pierwszy na wejściu, pierwszy na wyjściu** (FIFO - First In, First Out).

## (FIFO - FIRST IN, FIRST OUT)

Elementy są **dodawane** na **końcu** kolejki i **usuwane** z **przodu**.

Pierwszy element dodany do kolejki jest pierwszym, który zostanie z niej usunięty.



## PODSTAWOWE OPERACJE NA KOLEJCE:

### enqueue (dodawanie elementu)

Dodaje element na końcu kolejki.

### dequeue (usuwanie elementu)

Usuwa element z przodu kolejki i może zwracać jego wartość.

### front/peek (podglądanie pierwszego elementu)

Zwraca pierwszy element kolejki bez jego usuwania.

### isEmpty (sprawdzanie, czy kolejka jest pusta)

Sprawdza, czy kolejka nie zawiera żadnych elementów.

## IMPLEMENTACJA KOLEJKI:

### Implementacja kolejki za pomocą tablicy

Kolejka jest reprezentowana jako tablica z wskaźnikami na początek i koniec.

### Implementacja kolejki za pomocą dwóch stosów

Wykorzystuje dwa stosy do symulacji działania kolejki, jeden stos służy do dodawania elementów, a drugi do ich usuwania.

## RODZAJE KOLEJEK:

### Kolejka priorytetowa

Kolejka, w której każdy element ma przypisany priorytet, a elementy są usuwane z kolejki zgodnie z ich priorytetami.

### Kolejka kołowa (circular queue)

Kolejka, w której końce są logicznie połączone tak, że pierwszy element jest następcą ostatniego, co efektywnie wykorzystuje przestrzeń.

### Kolejka dwustronna (deque)

Struktura danych pozwalająca na dodawanie i usuwanie elementów zarówno z przodu, jak i z tyłu.

## ZASTOSOWANIA KOLEJEK:

- Symulacja kolejek w środowiskach produkcyjnych i logistycznych
- Obsługa żądań w serwerach sieciowych
- Zarządzanie zadaniami w systemach operacyjnych
- Buforowanie danych w strumieniowaniu multimedialnym
- Zarządzanie zdarzeniami w programowaniu asynchronicznym



# OCENIĆ PROSZĘ TEN ROZDZIAŁ



**InfaMatura**

<https://forms.gle/KmzLCoLfusHsb5oD6>



CHAPTER N.7

# Złożoność Obliczeniowa



Rodzaje Złożoności, Notacje, Klasy i Analiza  
Złożoności Obliczeniowej.



# Rodzaje Złożoności

## ZŁOŻONOŚĆ CZASOWA (TIME COMPLEXITY):

- Oznacza **ilość czasu**, jaką algorytm potrzebuje na wykonanie się w zależności od **rozmiaru wejścia**.
- Mierzy się liczbą operacji podstawowych wykonanych przez algorytm.

## ZŁOŻONOŚĆ PAMIĘCIOWA (SPACE COMPLEXITY):

- Oznacza **ilość pamięci**, jaką algorytm zużywa w zależności od **rozmiaru wejścia**.
- Uwzględnia pamięć potrzebną na przechowywanie danych wejściowych oraz dodatkowej pamięci używanej przez algorytm.

# Notacje Oznaczające Złożoność Obliczeniową

## $O(n)$ - Notacja dużego O:

- Opisuje **najgorszy** możliwy scenariusz (**czasu** lub **pamięci**) działania algorytmu w zależności od rozmiaru danych wejściowych **n**.

## $\Omega(n)$ - Notacja dużego Omega:

- Opisuje **najlepszy** możliwy scenariusz (**czasu** lub **pamięci**) działania algorytmu w zależności od rozmiaru danych wejściowych **n**.

## $\Theta(n)$ - Notacja dużego Theta:

- Opisuje **dokładny (średni)** scenariusz (**czasu** lub **pamięci**) działania algorytmu w zależności od rozmiaru danych wejściowych **n**.

# Typowe Klasy Złożoności Czasowej

## Stała (Constant Time) - $O(1)$

- Czas wykonania **nie zależy** od rozmiaru wejścia.
- **Przykład:** Dostęp do elementu tablicy.

## Logarytmiczna (Logarithmic Time) - $O(\log n)$

- Czas wykonania rośnie **logarytmicznie** wraz z rozmiarem wejścia.
- **Przykład:** Przeszukiwanie binarne.

## Liniowa (Linear Time) - $O(n)$

- Czas wykonania rośnie **liniowo** wraz z rozmiarem wejścia.
- **Przykład:** Przeszukiwanie liniowe.

## Liniowo-logarytmiczna (Linearithmic Time) - $O(n \times \log n)$

- Czas wykonania rośnie **w tempie  $n \log n$** .
- **Przykład:**
  - Sortowanie szybkie (**Quick Sort**),
  - Sortowanie przez scalanie (**Merge Sort**).

## Kwadratowa (Quadratic Time) - $O(n^2)$

- Czas wykonania rośnie **proporcjonalnie do kwadratu** rozmiaru wejścia.
- **Przykład:**
  - Sortowanie bąbelkowe (**Bubble Sort**),
  - Sortowanie przez wstawianie (**Insertion Sort**).

## Sześcienne (Cubic Time) - $O(n^3)$

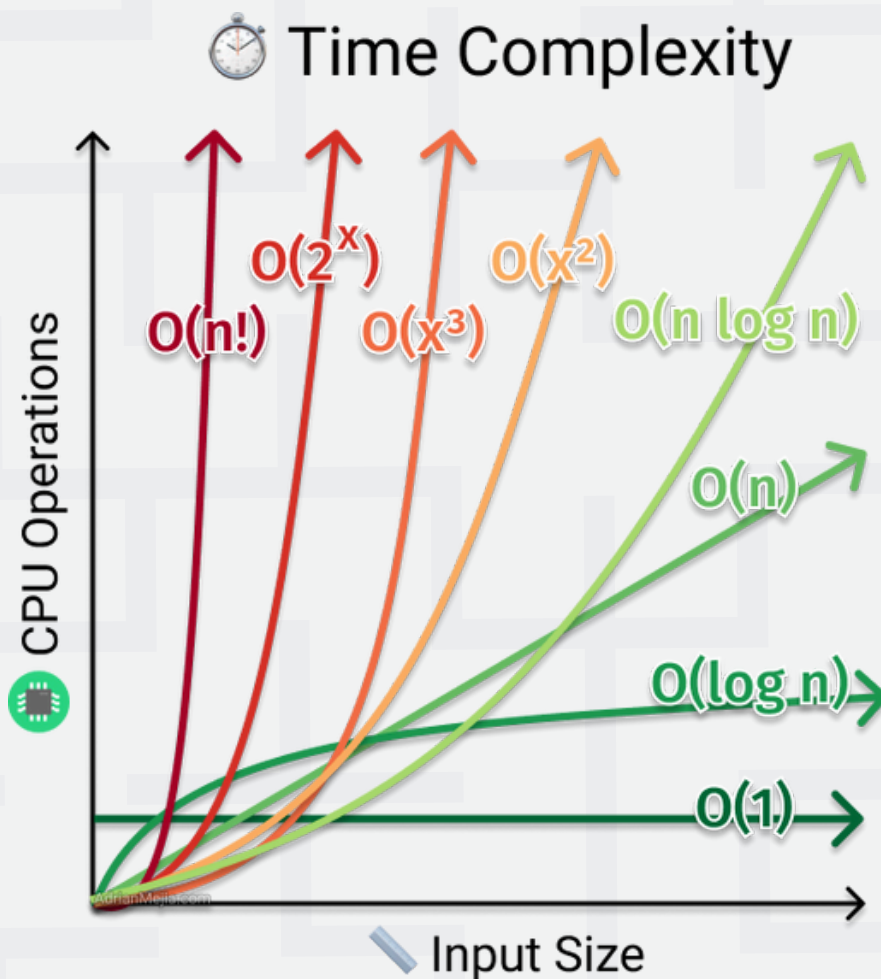
- Czas wykonania rośnie **proporcjonalnie do sześcianu** rozmiaru wejścia.
- **Przykład:** Algorytm Floyd-Warshall do znajdowania najkrótszych ścieżek w grafie.

## Ekspansyjna (Exponential Time) - $O(2^n)$

- Czas wykonania rośnie **wykładniczo** wraz z rozmiarem wejścia.
- **Przykład:** Algorytm brute-force dla problemu komiwojażera

## Złożoność Silni (Factorial Time Complexity) - $O(n!)$

- Czas wykonania algorytmu rośnie **wykładniczo w tempie silni** liczby wejściowej  $n$
- Jest to **najgorsza** możliwych złożoność obliczeniowa
- **Przykład:** Algorytmy, które generują wszystkie permutacje elementów zbioru, ponieważ istnieje  $n!$  możliwych permutacji dla zbioru  $n$  elementów.



### UWAGA:

Przy sytuacjach, gdzie mamy na przykład  $O(3n^2 + 2n + n + 1)$ , zawsze bierzemy pod uwagę **najwyższy czynnik  $n$**  w złożoności obliczeniowej i pomijamy pozostałe. W podanym przykładzie pozostanie nam  $O(n^2)$

# Analiza Złożoności Algorytmów

## NAJLEPSZY PRZYPADEK (BEST CASE):

- Analiza algorytmu w sytuacji, gdy działa on **najszybciej**.
- **Przykład:** Sortowanie bąbelkowe na już posortowanej tablicy.

## ŚREDNI PRZYPADEK (AVERAGE CASE):

- Analiza algorytmu w **przeciętnych warunkach**.
- **Przykład:** Przeszukiwanie liniowe w średniej sytuacji.

## NAJGORSZY PRZYPADEK (WORST CASE):

- Analiza algorytmu w sytuacji, gdy działa on **najwolniej**.
- **Przykład:** Przeszukiwanie binarne na tablicy, gdzie poszukiwanego elementu nie ma.

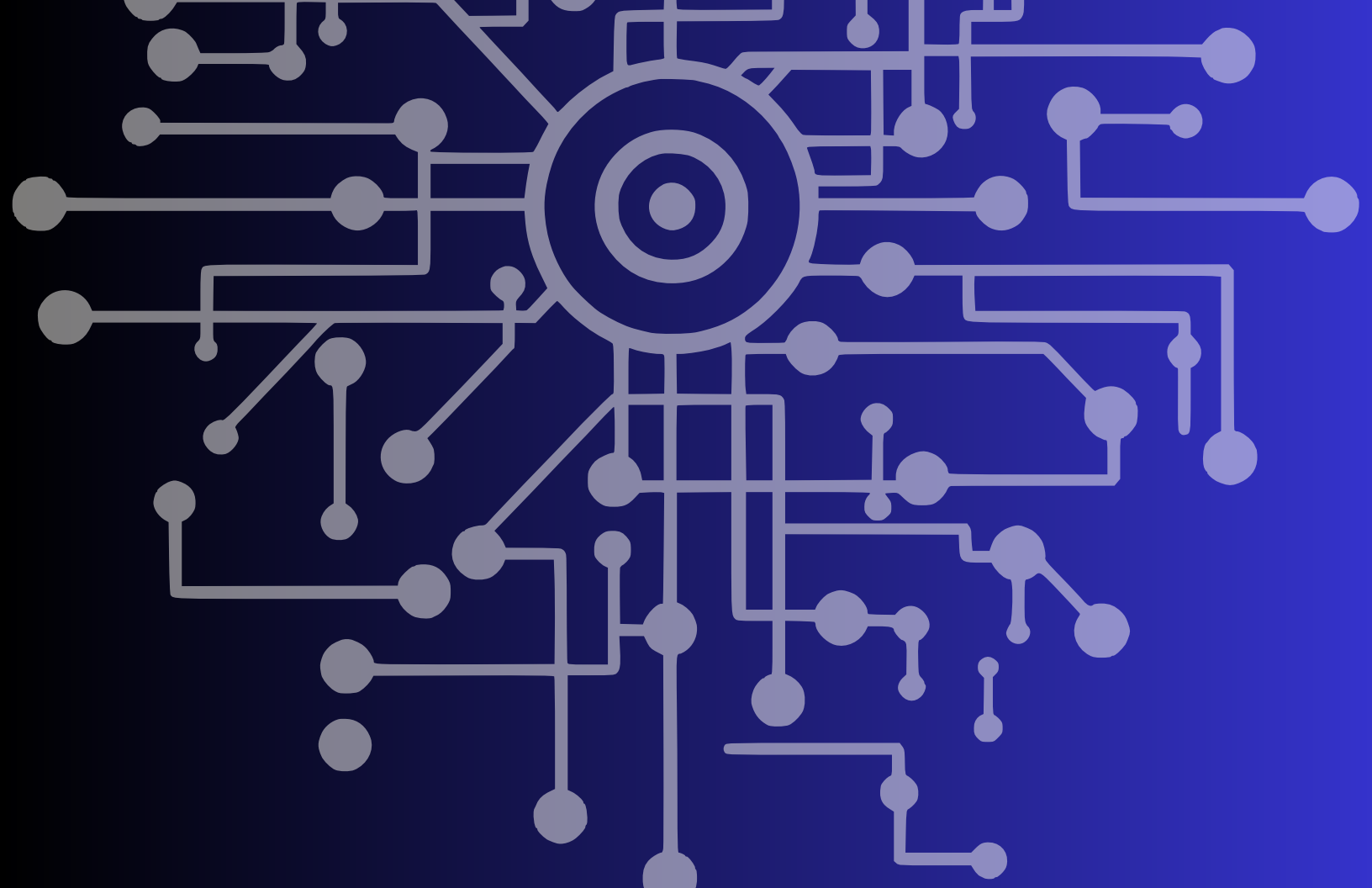
# Znaczenie Złożoności Obliczeniowej

- **Efektywność:** Pozwala ocenić, jak efektywny jest algorytm w zależności od rozmiaru danych wejściowych.
- **Skalowalność:** Pomaga zrozumieć, jak algorytm będzie się zachowywał przy zwiększających się danych wejściowych.
- **Wybór algorytmu:** Pomaga w wyborze najbardziej odpowiedniego algorytmu do danego problemu.

# OCEŃ PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/sxu5kdeUfHXwDes57>



CHAPTER N.8

# Pseudokod



Podstawowe zasady: Przypisanie, Tablice,  
Odczyt, Wczytywanie, Wypisywanie, Działania,  
Warunki, Funkcje





# Podstawowe zasady

## ZMIENNE:

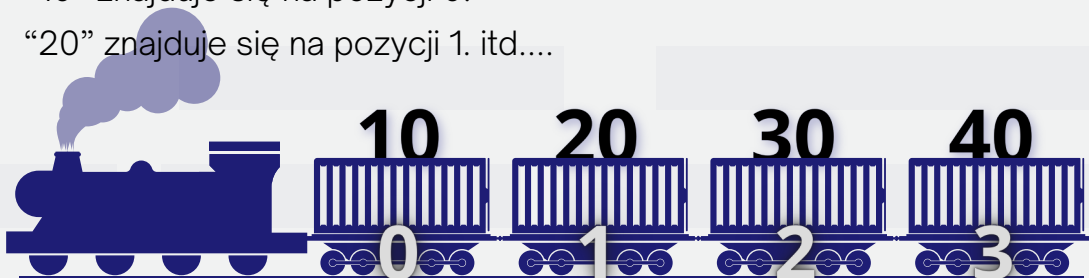
- Zmienną w pseudokodzie można przyrównać do **etykietowanego pudełka**, w którym przechowujemy **wartość**, mogą to być **liczby, tekst, wartości logiczne (prawda/fałsz)** czy **tablice**. Nazwa zmiennej pozwala łatwo odnaleźć i użyć to, co jest **w środku**. Wartości te **można zmieniać** w trakcie działania programu.

$wynik \leftarrow 1$  • wynik równa się 1  
 $n \leftarrow 0$  • n równa się 0  
 $n \leftarrow wynik$  • n równa się wynik, czyli 1



## TABLICE:

- Tablice** to struktury danych, które przechowują kolekcję elementów tego samego typu w uporządkowany sposób.
  - Można sobie je wyobrazić jako pociąg, w którym każdy wagon to miejsce na przechowywanie jednego elementu, na przykład liczbę, literę lub inny typ danych.
- Indeks** to pozycja, pod którą znajduje się element w tablicy. Indeksowanie w tablicach zazwyczaj zaczyna się od **zera**. Oznacza to, że pierwszy element tablicy znajduje się pod indeksem 0, drugi pod indeksem 1, i tak dalej.
  - Można to sobie wyobrazić tak, że każdy wagon w pociągu ma przypisany **unikalny numer**
- Przykład:**
  - Załóżmy, że mamy tablicę liczb: [10, 20, 30, 40].
    - “10” znajduje się na pozycji 0.
    - “20” znajduje się na pozycji 1. itd....



$A[ ]$  • Definiowanie pustej tablicy  
 $A[0 \dots n - 1]$  • Definiowanie pustej tablicy o **n** ilości elementów w niej  
 $T[4] \leftarrow 3$  • Przypisanie do tablicy T na indeksie 4 wartości 3

## ODCZYT, WCZYTYWANIE I WYPISYWANIE:

- **Odczyt** odnosi się do uzyskania wartości przechowywanej w zmiennej, aby można było jej użyć w obliczeniach lub operacjach logicznych. Odczyt to zwykle po prostu wykorzystanie wartości zmiennej w kodzie.

```
 $x \leftarrow 1$   
 $y \leftarrow x + 5$  • Odczytujemy wartość z “x” i dodajemy 5
```

W tym przypadku **x** jest odczytywane, a jego wartość jest używana do obliczenia **y**.

- **Wczytywanie** (czyli przyjmowanie danych wejściowych) polega na pobraniu danych od użytkownika i zapisaniu ich w zmiennej. W pseudokodzie wczytywanie często jest przedstawiane jako funkcja, która pyta użytkownika o dane.

```
wczytaj  $x$  • Pobierz wartość od użytkownika i przypisz do x
```

Użytkownik wprowadza wartość, która zostaje przypisana do zmiennej **x**.

- **Wypisywanie** (czyli prezentacja danych wyjściowych) polega na wyświetleniu wartości zmiennej lub wyniku obliczeń na ekranie lub w innym medium wyjściowym.

```
 $x \leftarrow 10$   
wypisz  $x$  • Wyświetl wartość przechowywaną w x
```

Wartość **10** zostanie wyświetlona na ekranie, ponieważ jest to aktualna wartość zmiennej **x**.

## DOZWOLONE DZIAŁANIA:

- Dodawanie +

$suma \leftarrow x + y$

- Odejmowanie -

$roznica \leftarrow x - y$

- Mnożenia \*

$iloczyn \leftarrow x * y$

- Dzielenie /

$iloraz \leftarrow x / y$

- Dzielenie z resztą MOD

$reszta \leftarrow x \text{ MOD } y$

- Dzielenie całkowite DIV

$calkowity \leftarrow x \text{ DIV } y$

- Nawiasowanie ( )

$wynik \leftarrow (x + y) / y$

- Sprawdzanie Równości, Nierówności i Różności =, <, >, ≤, ≥, ≠

$x < 5$

## WARUNKI:

- **if (jeżeli)** to instrukcja, która wykonuje kod tylko wtedy, gdy określony warunek jest prawdziwy. Jeśli warunek jest spełniony, program wykonuje kod znajdujący się w **bloku if**

jeżeli  $i < 5$  to:  
wypisz  $i$

- **else if (w przeciwnym wypadku jeżeli)** to dodatkowy warunek, który jest sprawdzany tylko jeśli poprzedni warunek **if** (lub inny **else if** przed nim) **nie** został spełniony. Pozwala to na sprawdzenie wielu różnych warunków, jeden po drugim.

w przeciwnym wypadku jeżeli  $i > 5$  to:  
wypisz  $i + 3$

- **else (w przeciwnym wypadku)** to instrukcja, która zostanie wykonana, jeśli żaden z poprzednich warunków **if** lub **else if** nie zostanie spełniony.

w przeciwnym wypadku:  
wypisz 5

## ŁĄCZENIE WARUNKÓW:

Można połączyć kilka warunków, używając operatorów:

- **ORAZ** (AND): Wykonuje blok kodu tylko wtedy, gdy **oba** warunki są prawdziwe

jeżeli ( $i < 5$  oraz  $j > 10$ ) to:  
wypisz  $i$

- **LUB** (OR): Wykonuje blok kodu, jeśli przynajmniej jeden z warunków jest prawdziwy.

jeżeli ( $i < 5$  lub  $j > 10$ ) to:  
wypisz  $i$

## ZAPRZECZENIA WARUNKÓW:

Można używać operatora **NIE** (zapisany jako NOT), aby **zaprzeczyć** warunkowi, czyli sprawdzić, czy warunek jest **falszywy**.

jeżeli (nie ( $i < 5$ )) to:  
wypisz  $i$

“ $i$ ” jest **większe**  
lub **równe** 5

## W PEŁNYM KONTEKŚCIE:

jeżeli ( $i < 5$  oraz ( $j > 10$  lub nie ( $k = 3$ ))) to:  
wypisz  $i$

W tym przykładzie “ $i$ ” zostanie wypisane, gdy:

- “ $i$ ” jest mniejsze od 5
- “ $j$ ” jest większe od 10 **lub** “ $k$ ” jest różne od 3

# FUNKCJE

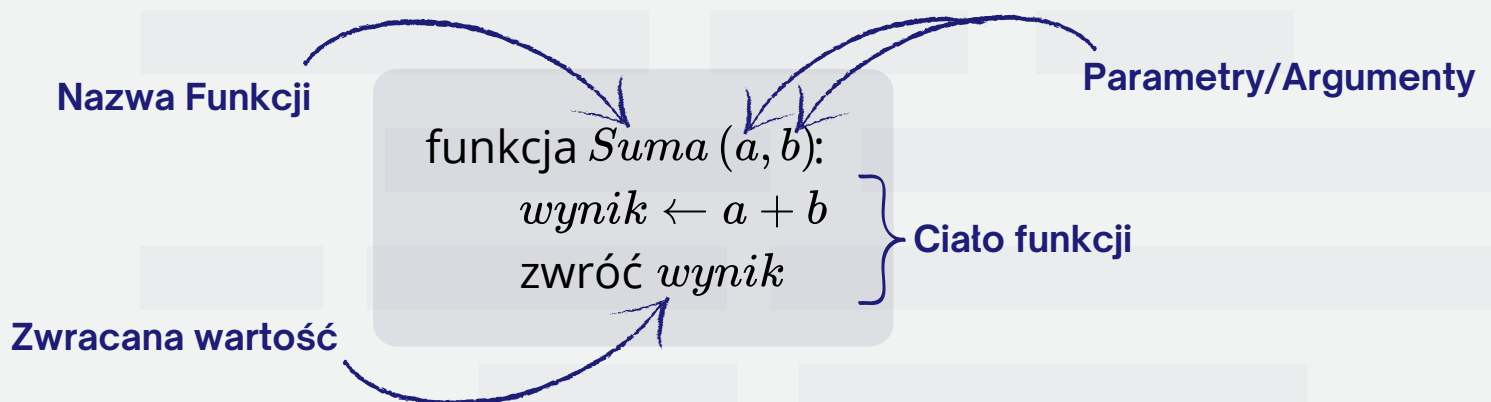
- **Funkcja** jest używana do reprezentacji bloku kodu, który wykonuje określoną operację lub obliczenie i jest zdefiniowany raz, ale może być wywoływany wielokrotnie z różnych miejsc w programie.

## STRUKTURA FUNKCJI W PSEUDOKODZIE:

Funkcja w pseudokodzie zazwyczaj zawiera następujące elementy:

- **Nazwa funkcji:**
  - Jednoznacznie identyfikuje funkcję i jest używana do jej wywołania.
- **Parametry/Argumenty (opcjonalne):**
  - Zmienne przekazane do funkcji, które funkcja używa do wykonania operacji. Parametry są opcjonalne; funkcja może ich nie mieć.
- **Ciało funkcji:**
  - Zawiera instrukcje pseudokodu, które definiują, co funkcja ma robić.
- **Zwracana wartość (opcjonalnie):**
  - Wynik operacji wykonanej przez funkcję, który jest zwracany do miejsca, z którego funkcja została wywołana. Nie wszystkie funkcje muszą zwracać wartość.

## PRZYKŁAD FUNKCJI W PSEUDOKODZIE:



## WYWOŁANIE FUNKCJI:

- Funkcję można **wywołać**, podając jej **nazwę** oraz **argumenty (jeśli wymaga)**, które odpowiadają parametrom funkcji. Oto jak można wywołać powyższą funkcję:

```
 $x \leftarrow 5$   
 $y \leftarrow 10$   
 $sumaXY \leftarrow Suma(x, y)$   
wypisz  $sumaXY$ 
```

## UŻYCIE FUNKCJI:

Funkcje w pseudokodzie są używane do:

- Organizacji kodu:** Dzielenie kodu na mniejsze, zarządzalne funkcje, które wykonują określone zadania.
- Unikania powtórzeń:** Jeśli ten sam blok kodu ma być używany wielokrotnie, lepiej jest umieścić go w funkcji i wywołać tę funkcję, zamiast kopiować i wklejać ten sam kod.
- Ukrywania szczegółów implementacji:** Użytkownik funkcji nie musi wiedzieć, jak funkcja działa wewnętrznie, aby z niej korzystać.

# JAK CZYTAĆ PSEUDOKOD

Na maturze z informatyki jednym z typów zadań jest **analiza pseudokodu** i wykonanie związanych z nim poleceń. Kluczowe jest **zrozumienie** działania kodu, jego ograniczeń oraz tego, jak reaguje na różne dane wejściowe.

Wyróżniłbym **dwa** sposoby analizy:

- **blokowy**
- **linia po linii**

## ANALIZA BLOKOWA

- **Zasada działania:** Polega na patrzeniu na cały fragment kodu jako na większe **bloki funkcjonalne**. Skupiamy się na ogólnym działaniu każdego bloku (np. **pętli, warunków, funkcji**), aby szybko zrozumieć główne założenia i cel programu bez wchodzenia w szczegóły każdej linii.
- **Zastosowanie:** Analizę blokową stosujemy głównie do szybkiego zapoznania się z podanym kodem i udzielenia odpowiedzi na kilka krótkich przykładów, które trzeba wykonać szybko.

## ANALIZA LINIA PO LINII

- **Zasada działania:** Polega na dokładnym prześledzeniu każdej linii pseudokodu, uwzględniając jej działanie oraz wpływ na **zmienne** i **logikę** programu. Zalecam **zapisywanie wyniku każdej wykonanej linii**, aby uniknąć pomyłek. Dzięki temu możemy szczegółowo ocenić, jak kod zachowa się w różnych sytuacjach, w tym dla specyficznych danych wejściowych.
- **Zastosowanie:** Analizę linia po linii stosujemy głównie podczas **tworzenia własnego pseudokodu** i chęci jego sprawdzenia. Jest to również odpowiednia metoda przy zadaniach z **jednym złożonym przykładem** do pseudokodu.

OCENI PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/gH1rkaQkuDfRxRPu6>





CHAPTER N.9

# Iteracja



Struktury Sterujące: For, While, Do-While,  
Zastosowania

# Iteracja

## Definicja:

Iteracja to technika programowania, w której blok kodu jest **wielokrotnie wykonywany**, dopóki spełniony jest **określony warunek**. Proces ten jest kontrolowany za pomocą struktur sterujących, takich jak **pętle**.

## STRUKTURY STERUJĄCE:

- **Pętla for:** Używana, gdy liczba iteracji jest znana z góry.

- Przykład w pseudokodzie:

```
dla  $i \leftarrow 1..5$  wykonuj  
wypisz  $i$ 
```

Kropki w pseudokodzie oznaczają **zakres wartości**. W tym przykładzie “1..5”, oznacza to, że zmienna “i” przyjmie po kolei wartości **1, 2, 3, 4, 5**

- **Pętla while:** Używana, gdy liczba iteracji nie jest z góry znana, a zależy od warunku logicznego.

- Przykład w pseudokodzie:

```
 $i \leftarrow 0$   
dopóki  $i < 5$  wykonuj  
wypisz  $i$   
 $i \leftarrow i + 1$ 
```

- **Pętla do-while:** Używana w niektórych językach programowania (np. C, C++), gdzie blok kodu jest wykonany **co najmniej raz**, a następnie dopóki warunek jest spełniony.

- Przykład w pseudokodzie:

```
 $i \leftarrow 0$   
powtarzaj  
wypisz  $i$   
 $i \leftarrow i + 1$   
aż  $i < 5$ 
```

## ZASTOSOWANIA:

- Przetwarzanie **list i tablic**:
  - Iteracja jest często używana do przetwarzania każdego elementu w strukturze danych.
- Algorytmy wyszukiwania:
  - Wyszukiwanie **liniowe** lub **binarne** w strukturach danych.

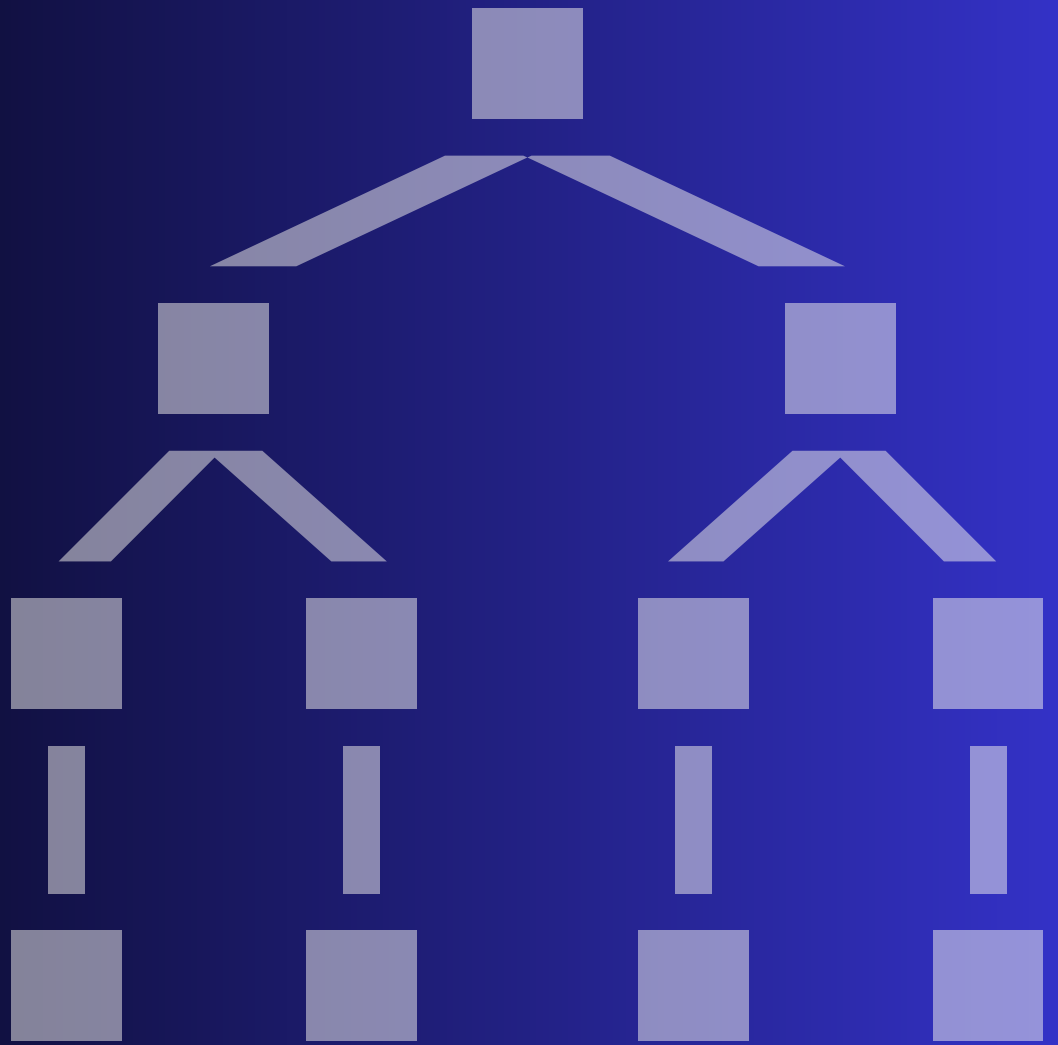
## ZALETY ITERACJI:

- **Efektywność**:
  - Iteracja zazwyczaj zużywa mniej pamięci niż rekurencja, ponieważ nie ma potrzeby przechowywania wielu wywołań funkcji na stosie wywołań.
- **Czytelność**:
  - Dla niektórych problemów iteracja może być bardziej intuicyjna i łatwiejsza do zrozumienia niż rekurencja.
- **Kontrola**:
  - Pętle umożliwiają łatwą kontrolę nad iteracjami, co jest przydatne w wielu algorytmach.

OCENIĆ PROSZĘ TEN ROZDZIAŁ

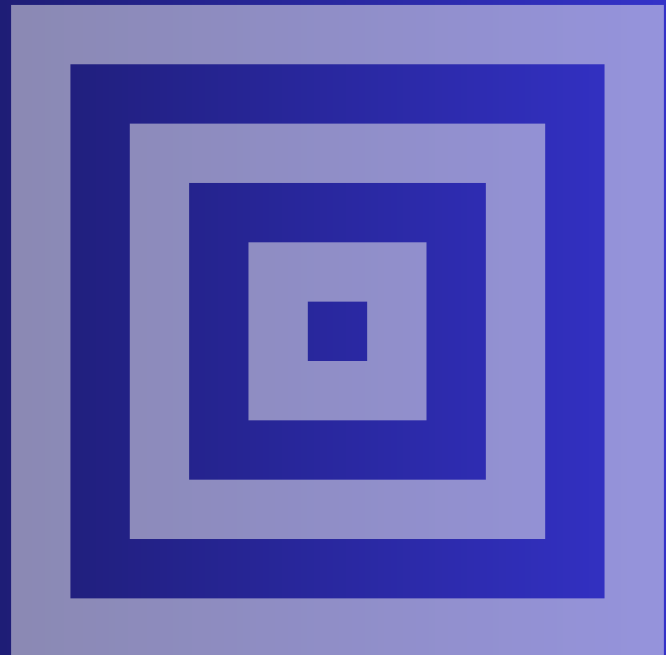


<https://forms.gle/BiBMhruT9zpD6QcL8>



CHAPTER N.10

# Rekurencja



Podstawowe Elementy, Zastosowania,  
Graficzne przedstawienie Rekurencji Drzewem

# Rekurencja

## Definicja:

Rekurencja to technika w programowaniu, gdzie funkcja wywołuje samą siebie w swoim ciele, aby rozwiązać mniejszą wersję problemu, aż osiągnie warunek końcowy (bazowy), który zatrzymuje rekursję.

## PODSTAWOWE ELEMENTY REKURENCJI:

- **Warunek bazowy (przypadek bazowy):**
  - Punkt, w którym rekurencyjne wywołania się zatrzymują. Jest to najprostszy przypadek problemu, który może być rozwiązany bez dalszych wywołań funkcji.
- **Wywołanie rekurencyjne:**
  - Funkcja wywołuje samą siebie z argumentami zmniejszającymi problem, przybliżając go do warunku bazowego.

funkcja *silnia*( $n$ ):  
jeżeli  $n = 0$  to:  
zwróć 1

w przeciwnym wypadku:

zwróć  $n * \text{silnia}(n - 1)$

Warunek bazowy

Wywołanie rekurencyjne

## ZASTOSOWANIA REKURENCJI:

- **Algorytmy sortowania:**
  - Quick Sort,
  - Merge Sort.
- **Rozwiązywanie problemów kombinatorycznych:**
  - problem wieży Hanoi,
  - generowanie permutacji.
- **Algorytmy dynamiczne:**
  - najdłuższy wspólny podciąg.

## ZALETY REKURENCJI:

- **Łatwość rozumienia i implementacji:**
  - Problemy o naturalnym podziale na mniejsze podproblemy są często prostsze do zrozumienia i zakodowania rekurencyjnie.
- **Elegancki kod:**
  - Kod rekurencyjny często bywa krótszy i bardziej zwięzły.

## ROZRYŚOWANIE REKURENCJI DRZEWEM

Aby łatwiej zrozumieć rekurencję, pomocne może być jej graficzne przedstawienie, na przykład za pomocą drzewa. Taka wizualizacja pozwala łatwiej zauważyć zależności, obserwować wykonywane działania i obliczać funkcje.

### Zadanie 1.2. (1 pkt)

Dana jest funkcja  $f$  określona wzorem rekurencyjnym

$$\begin{cases} f(1) = 4 \\ f(n+1) = \frac{1}{1-f(n)} \quad \text{dla } n \geq 1 \end{cases}$$

Wtedy:

|    |                         |   |   |
|----|-------------------------|---|---|
| 1. | $f(8) = \frac{1}{3}$    | P | F |
| 2. | $f(9) = \frac{3}{4}$    | P | F |
| 3. | $f(10) = 4$             | P | F |
| 4. | $f(100) = -\frac{1}{3}$ | P | F |

## WYWOŁANIA REKURENCJI

Dla  $n = 5$

5

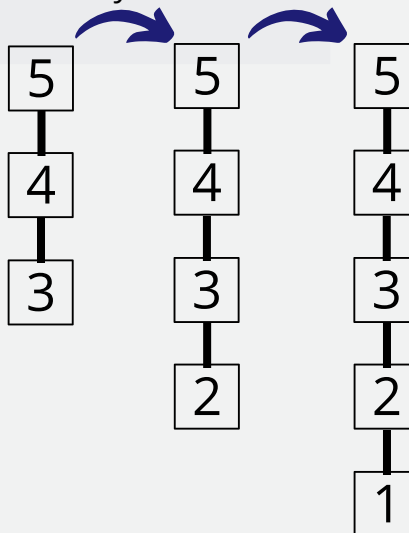
$$n = 4 \geq 1$$

$$f(5) = f(4+1) = \frac{1}{1-f(4)}$$

Jak widać do obliczenia  $f(5)$  potrzebujemy  $f(4)$

5  
4

Analogicznie sprawdzamy następne, aż natrafimy na warunek bazowy



Warunek bazowy rekurencji,  $f(1) = 4$ , pozwala zacząć obliczenia, cofając się do poprzednich etapów rekurencji.

Dobrze mamy już stworzone drzewo, możemy zacząć obliczać wyniki.

$$\begin{aligned}
 \boxed{5} &= \frac{1}{1-f(4)} = \frac{1}{1-4} = -\frac{1}{3} \\
 \boxed{4} &= \frac{1}{1-f(3)} = \frac{1}{1-\frac{3}{4}} = \frac{1}{\frac{1}{4}} = 4 \\
 \boxed{3} &= \frac{1}{1-f(2)} = \frac{1}{1-(-\frac{1}{3})} = \frac{1}{\frac{4}{3}} = \frac{3}{4} \\
 \boxed{2} &= \frac{1}{1-f(1)} = \frac{1}{1-4} = -\frac{1}{3} \\
 \boxed{1} &= f(1) = 4
 \end{aligned}$$

Widać tu pewien cykl, który pozwala nam rozwiązać zadanie.

Dzięki podzieleniu liczby wejściowej  $n$  przez  $3$  z resztą, możemy natychmiast odczytać wynik na podstawie otrzymanej reszty, co daje nam trzy różne możliwe wyniki.

$$1 \div 3 = 0 \text{ } r \text{ } 1 \rightarrow 4$$

$$2 \div 3 = 0 \text{ } r \text{ } 2 \rightarrow -\frac{1}{3}$$

$$3 \div 3 = 1 \text{ } r \text{ } 0 \rightarrow \frac{3}{4}$$

$$8 \div 3 = 2 \text{ } r \text{ } 2 \rightarrow -\frac{1}{3}$$

$$9 \div 3 = 3 \text{ } r \text{ } 0 \rightarrow \frac{3}{4}$$

$$10 \div 3 = 3 \text{ } r \text{ } 1 \rightarrow 4$$

$$100 \div 3 = 33 \text{ } r \text{ } 1 \rightarrow 4$$

Wtedy:

|    |                         |   |   |
|----|-------------------------|---|---|
| 1. | $f(8) = \frac{1}{3}$    | P | × |
| 2. | $f(9) = \frac{3}{4}$    | × | F |
| 3. | $f(10) = 4$             | × | F |
| 4. | $f(100) = -\frac{1}{3}$ | P | × |

## Ćwiczenie 10.1

$$\begin{cases} g(1) = 2 \\ g(n+1) = 1 - g(n) + n \text{ dla } n \geq 1 \end{cases}$$

Oceń prawdziwość podpunktów:

- a)  $g(6) = 2$
- b)  $g(3) = g(8)$
- c)  $g(10) = 8$
- d) funkcje  $g$  zwraca jedynie wartości dodatnie



OCENIĆ PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/wMP4wBfxSao5ykB57>

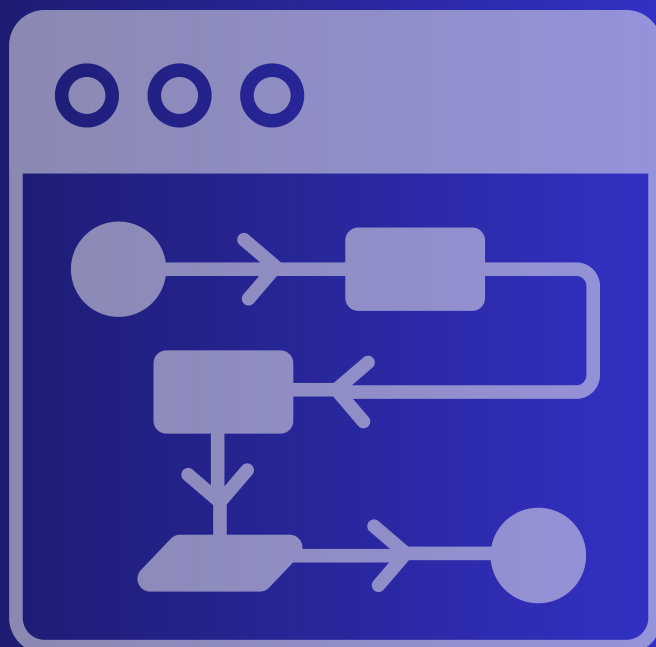


CHAPTER N.11

# Algorytmy



Algorytmy Sortowania, Przeszukiwania,  
Kryptograficzne



# Algorytmy Sortowania

**Algorytm sortowania** to procedura obliczeniowa, która porządkuje elementy w zbiorze danych według określonego kryterium, na przykład w kolejności rosnącej lub malejącej.

## SORTOWANIE BĄBELKOWE (BUBBLE SORT)

- Prosty algorytm sortowania działający poprzez wielokrotne przechodzenie przez listę, **porównywanie i zamienianie sąsiadujących elementów, jeśli są w złej kolejności**.
- Pierwszy element jest porównywany z drugim, drugi z trzecim, trzeci z czwartym itd. aż do ostatniego. Zamiana wywoływana jest przez niepoprawną kolejność.
- Cały taki proces od początku listy aż po sam koniec nazywamy **przejęciem**

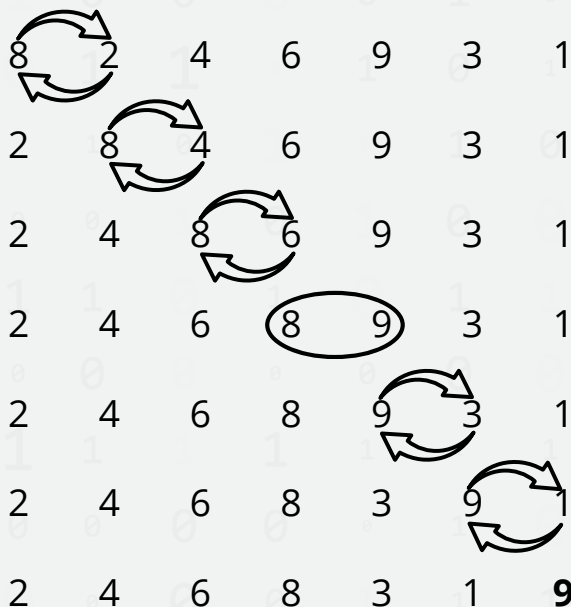
### Wskazówka

Największy element z listy po jednym przejęciu, wypływa na wierzch tak samo jak bąbelki w napojach gazowanych, stąd nazwa sortowanie bąbelkowe

- Kiedy już algorytm dotrze do **końca listy**, największa wartość została już ustawiona w prawidłowej kolejności, algorytm **wraca znów na początek listy**. Zaczyna wtedy sortować **kolejny największy element** w **pozostalej części listy**, by znalazł się na prawidłowym miejscu.
- Przejęcia są wykonywane aż **wszystkie elementy** zostaną ustawione w **poprawnej kolejności**
- Ostatnie przejęcie sprawdza wszystkie elementy, jeśli w przejęciu nie zostanie wykonana żadna zamiana, sortowanie zostało zakończone.

# WIZUALIZACJA SORTOWANIA BĄBELKOWEGO

Przejsie 1:



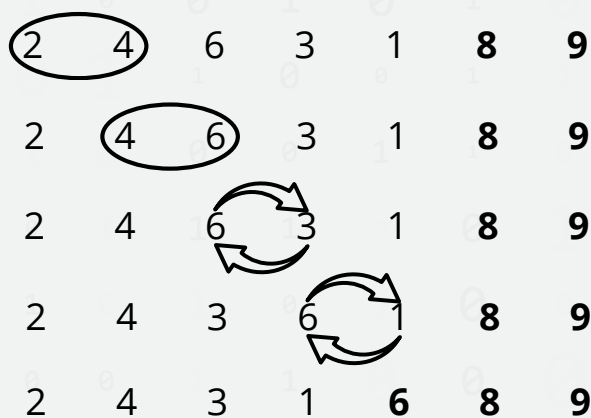
8 > 2? ✓ Zamiana  
8 > 4? ✓ Zamiana  
8 > 6? ✓ Zamiana  
8 > 9? ✗ Brak Zamiany  
9 > 3? ✓ Zamiana  
9 > 1? ✓ Zamiana  
Koniec listy,  
zakończenie 1 Przejsia

Przejsie 2:



2 > 4? ✗ Brak Zamiany  
4 > 6? ✗ Brak Zamiany  
6 > 8? ✗ Brak Zamiany  
8 > 3? ✓ Zamiana  
8 > 1? ✓ Zamiana  
Koniec czści listy,  
zakończenie 2 Przejsia

Przejsie 3:



2 > 4? ✗ Brak Zamiany  
4 > 6? ✗ Brak Zamiany  
6 > 3? ✓ Zamiana  
6 > 1? ✓ Zamiana  
Koniec czści listy,  
zakończenie 3 Przejsia

Przejście 4:

2 4 6 3 1 8 9

2 > 4?  Brak Zamiany

2 4 6 3 1 8 9

4 > 6?  Brak Zamiany

2 4 6 3 1 8 9

6 > 3?  Zamiana

2 4 3 6 1 8 9

6 > 1?  Zamiana

2 4 3 1 6 8 9

Koniec części listy,  
zakończenie 4 Przejścia

Przejście 5:

2 4 3 1 6 8 9

2 > 4?  Brak Zamiany

2 4 3 1 6 8 9

4 > 3?  Zamiana

2 3 4 1 6 8 9

4 > 1?  Zamiana

2 3 1 4 6 8 9

Koniec części listy,  
zakończenie 5 Przejścia

Przejście 6:

2 3 1 4 6 8 9

2 > 3?  Brak Zamiany

2 3 1 4 6 8 9

3 > 1?  Zamiana

2 1 3 4 6 8 9

Koniec części listy,  
zakończenie 6 Przejścia

Przejście 7:

2 3 4 6 8 9

2 > 1?  Zamiana

1 2 3 4 6 8 9

Koniec części listy,  
zakończenie 7 Przejścia

Przejście 8:

1 2 3 4 6 8 9

Brak zamiany w przejściu,  
Koniec wykonywania  
algorytmu

## SORTOWANIE BĄBELKOWE W PSEUDOKODZIE

```
① list ← [5, 9, 4, 2, 6, 7, 1, 2, 4, 3]
② last ← dł(list)
③ swap ← True
  i ← 0
④ dopóki i < (last - 1) oraz (swap = True):
    swap ← False
    ⑤ dla j ← 0...(last - i - 2):
      ⑥ jeżeli list[j] > list[j+1]:
        temp ← list[j]
        list[j] ← list[j+1]
        list[j+1] ← temp
        swap ← True
    ⑦ i ← i + 1
⑧ wypisz list
```

- 1 **Inicjalizacja listy:** Lista o nazwie **list** jest zdefiniowana z zestawem wartości. Na przykładzie: [5, 9, 4, 2, 6, 7, 1, 2, 4, 3].
- 2 **Obliczenie długości listy:** Zmienna **last** jest inicjalizowana jako **długość listy**. Jest to potrzebne, aby wiedzieć, jak daleko iterować w pętlach. Jest wykorzystywana funkcja **dł()**. Nie jest ona dostępna w podstawowej wersji pseudokodu. Jednakże często w zadaniach maturalnych jeśli jest potrzebna, będzie zadeklarowana.
- 3 **Zmienna sterująca pętlą:** Zmienna **swap** jest ustawiona na **True**, aby uruchomić pętlę **dopóki** i kontrolować, czy w danym przejściu pętli nastąpiła jakakolwiek zamiana elementów.
- 4 **Pętla główna - dopóki:** Ta pętla będzie kontynuowana, dopóki nie przejdziemy przez **całą listę** ( $i < last - 1$ ) oraz będą **występować zamiany** (**swap = True**). Przed **rozpoczęciem każdego przejścia pętli** wewnętrznej ustawiamy **swap na False**.
- 5 **Pętla wewnętrzna - dla:** Iteruje przez listę **od początku do last - i - 2**. Długość iteracji **zmniejsza się** z każdym kolejnym przejściem pętli zewnętrznej, ponieważ **ostatni element** (lub więcej) jest już na **swoim miejscu**.

```

① list ← [5, 9, 4, 2, 6, 7, 1, 2, 4, 3]
② last ← dł(list)
③ swap ← True
  i ← 0
④ dopóki i < (last - 1) oraz (swap = True):
    swap ← False
    ⑤ dla j ← 0...(last - i - 2):
      ⑥ jeżeli list[j] > list[j+1]:
        temp ← list[j]
        list[j] ← list[j+1]
        list[j+1] ← temp
        swap ← True
    ⑦ i ← i + 1
⑧ wypisz list

```

**6 Warunek - jeżeli:** W tej części kodu sprawdzamy, czy aktualny element **list[j]** jest **większy** niż element następny **list[j+1]**. Jeśli tak, to:

- **Zamieniamy je** miejscami, używając zmiennej tymczasowej **temp**.
- Ustawiamy **swap na True**, oznaczając, że została wykonana zamiana i konieczne może być kolejne przejście przez listę.

**7 Inkrementacja licznika:** Po każdym przejściu przez pętlę wewnętrzną zwiększamy **i o 1**. Zapewnia to, że kolejny przejście pętli wewnętrznej będzie **krótszy o jeden element**.

**8 Wypisanie posortowanej listy:** Na końcu, po zakończeniu wszystkich pętli, lista jest posortowana od najmniejszego do największego elementu i jest wypisywana.

## ZŁOŻONOŚĆ CZASOWA

**najlepsza**

$\Omega(n)$

**średnia**

$\Theta(n^2)$

**najgorsza**

$O(n^2)$

## ZŁOŻONOŚĆ PAMIĘCIOWA

$O(1)$

Algorytm składa się z:

- 6 instrukcji o złożoności  $O(1)$
- pętli zagnieżdżonych
  - zewnętrzna pętla while -  $O(n)$
  - wewnętrzna pętla for -  $O(n)$

**Złożoność pamięciowa** algorytmu

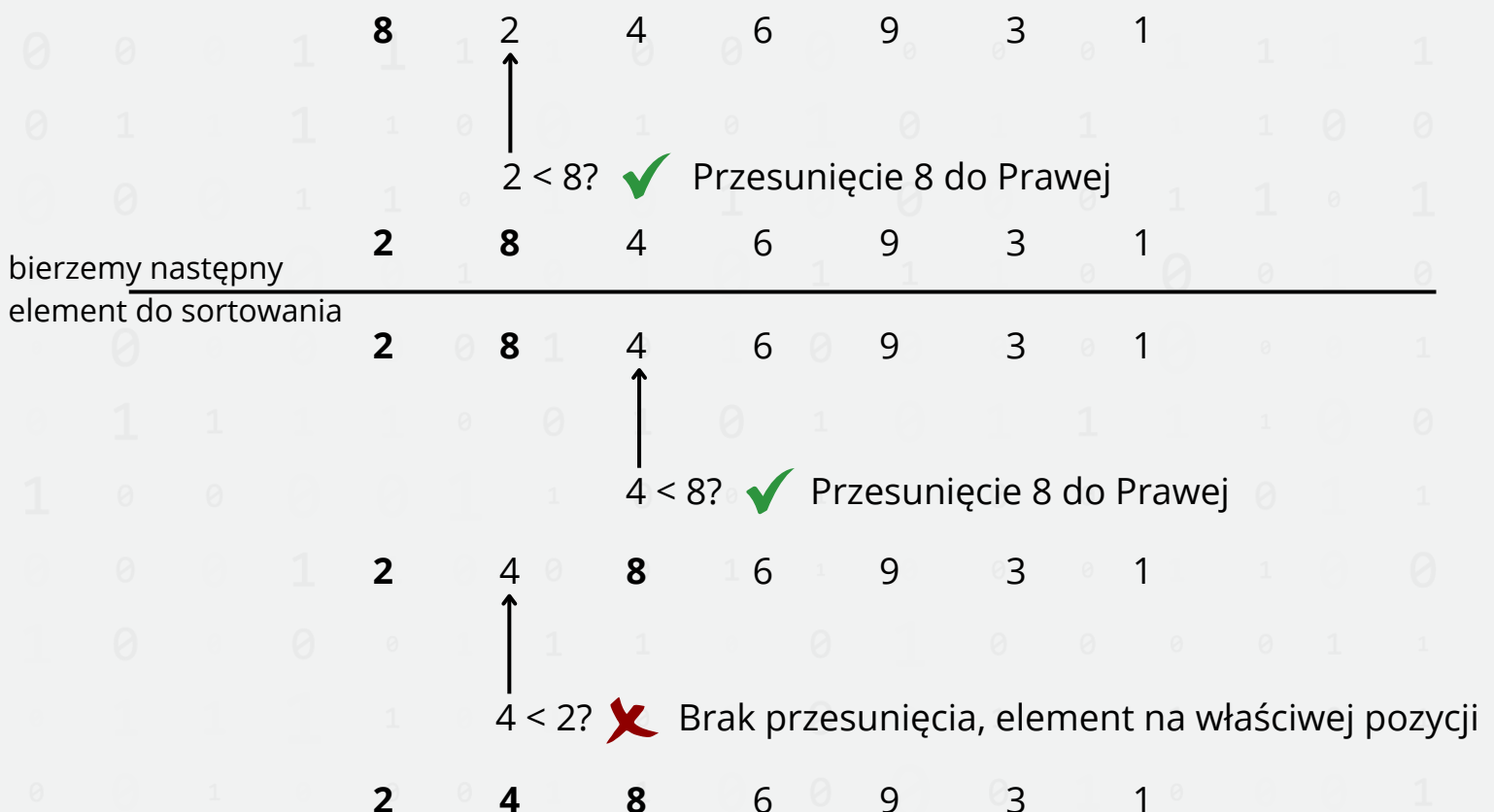
sortowania bąbelkowego to  **$O(1)$** , co oznacza, że algorytm ten **nie wymaga** dodatkowej pamięci poza przechowywaną listą.

# SORTOWANIE PRZEZ WSTAWIANIE (INSERTION SORT)

- Sortowanie przez wstawianie, działa tak, że sortuje tylko jeden element na raz poprzez ustawienie go na prawidłowej pozycji w nieposortowanej liście. Ten proces jest powtarzany aż algorytm przejdzie po wszystkich elementach listy.
- Dokładniej element który jest w danym momencie sortowany jest porównywany do każdego poprzedniego sortowanego elementu.
  - Jeśli jest mniejszy niż poprzedni element, wtedy poprzedni element jest przesuwany w prawo, a obecnie sprawdzana wartość zajmuje jego miejsce.
  - Jeśli jest większy niż poprzedni element, oznacza to że jest on na prawidłowej pozycji i algorytm bierze kolejny element do sortowania.

## WIZUALIZACJA SORTOWANIA PRZEZ WSTAWIANIE

 8 jest już uznane za posortowane, ponieważ jest pierwsze





|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 2 | 4 | 8 | 6 | 9 | 3 | 1 |
|---|---|---|---|---|---|---|

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 2 | 4 | 8 | 6 | 9 | 3 | 1 |
|---|---|---|---|---|---|---|



6 < 8?



Przesunięcie 8 do Prawej

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 8 | 9 | 3 | 1 |
|---|---|---|---|---|---|---|



6 < 4?



Brak przesunięcia, element na właściwej pozycji

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 8 | 9 | 3 | 1 |
|---|---|---|---|---|---|---|

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 8 | 9 | 3 | 1 |
|---|---|---|---|---|---|---|



9 < 8?



Brak przesunięcia, element na właściwej pozycji

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 8 | 9 | 3 | 1 |
|---|---|---|---|---|---|---|

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 8 | 9 | 3 | 1 |
|---|---|---|---|---|---|---|



3 < 9?



Przesunięcie 9 do Prawej

3 < 8?



Przesunięcie 8 do Prawej

3 < 6?



Przesunięcie 6 do Prawej

3 < 4?



Przesunięcie 4 do Prawej

3 < 2?



Brak przesunięcia, element na właściwej pozycji

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 6 | 8 | 9 | 1 |
|---|---|---|---|---|---|---|

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 6 | 8 | 9 | 1 |
|---|---|---|---|---|---|---|



1 < 9?



Przesunięcie 9 do Prawej

1 < 8?



Przesunięcie 8 do Prawej

1 < 6?



Przesunięcie 6 do Prawej

1 < 4?



Przesunięcie 4 do Prawej

1 < 3?



Przesunięcie 3 do Prawej

1 < 2?



Przesunięcie 2 do Prawej



Brak przesunięcia, element na pierwszej pozycji

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 6 | 8 | 9 |
|---|---|---|---|---|---|---|

# SORTOWANIE PRZEZ WSTAWIANIE W PSEUDOKODZIE

```
① list ← [5, 9, 4, 2, 7, 1, 2, 4, 3]
   n ← dł(list)
② dla i ← 1...(n - 1) wykonuj:
   ③ item ← list[i]
     position ← i
   ④ dopóki position > 0 oraz list[position - 1] > item wykonuj:
     ⑤ list[position] ← list[position - 1]
       position ← position - 1
   ⑥ list[position] ← item
⑦ wypisz list
```

- 1 **Inicjalizacja listy i zmiennej n:** Pseudokod zaczyna od zdefiniowania listy wartości, np. [5, 9, 4, 2, 7, 1, 2, 4, 3], a następnie przypisuje do zmiennej **n** długość tej listy.
- 2 **Pętla główna:** Pętla **dla** rozpoczyna iterację od **indeksu 1 aż do n-1**. Wartości indeksu **i** reprezentują pozycję elementu w liście, który ma być wstawiony na odpowiednie miejsce.

## Przypomnienie

W informatyce **indeksowanie** zaczynamy od **0**. Więc tutaj zaczynając pętlę od **i** równym **1**, zaczynamy od elementu na drugiej pozycji, nie na pierwszej.

- 3 **Pobieranie elementu i określenie pozycji startowej:** W każdej iteracji pętli głównej pobierany jest **element** z listy na pozycji **i** (przechowywany w zmiennej **item**) oraz ustawiana jest zmienna **position** na wartość **i**, co oznacza aktualną pozycję elementu w procesie wstawiania.
- 4 **Pętla wewnętrzna:** Wewnętrzna pętla **dopóki** kontynuuje działanie, dopóki **position** jest **większe od 0** (czyli element nie jest pierwszym elementem w liście) i element poprzedzający (na pozycji **position - 1**) jest **większy** od wstawianego elementu (**item**).

```

① list ← [5, 9, 4, 2, 7, 1, 2, 4, 3]
  n ← dł(list)
② dla i ← 1...(n - 1) wykonuj:
  ③ item ← list[i]
    position ← i
  ④ dopóki position > 0 oraz list[position - 1] > item wykonuj:
    ⑤ list[position] ← list[position - 1]
      position ← position - 1
  ⑥ list[position] ← item
⑦ wypisz list

```

- ⑤ **Przesuwanie elementów:** Jeśli warunki są spełnione,
- element na pozycji **position - 1** (większy element) jest przesuwany na pozycję **position**.
  - Następnie **position** jest **dekrementowane o 1**, aby kontynuować porównania z kolejnymi elementami w lewo.
- ⑥ **Wstawienie elementu:** Po wyjściu z pętli **dopóki**, wstawiany element (**item**) jest umieszczany na swojej nowej pozycji w posortowanej części listy.
- ⑦ **Wypisanie posortowanej listy:** Po zakończeniu wszystkich iteracji pętli **dla**, lista jest wypisywana, co daje listę **posortowaną rosnąco**.

## ZŁOŻONOŚĆ CZASOWA

**najlepsza**  
 $\Omega(n)$

**średnia**  
 $\Theta(n^2)$

**najgorsza**  
 $O(n^2)$

Algorytm składa się z:

- 5 instrukcji o złożoności  $O(1)$
- pętli zagnieżdżonych
  - zewnętrzna pętla for -  $O(n)$
  - wewnętrzna pętla while -  $O(n)$

## ZŁOŻONOŚĆ PAMIĘCIOWA

$O(1)$

**Złożoność pamięciowa** algorytmu

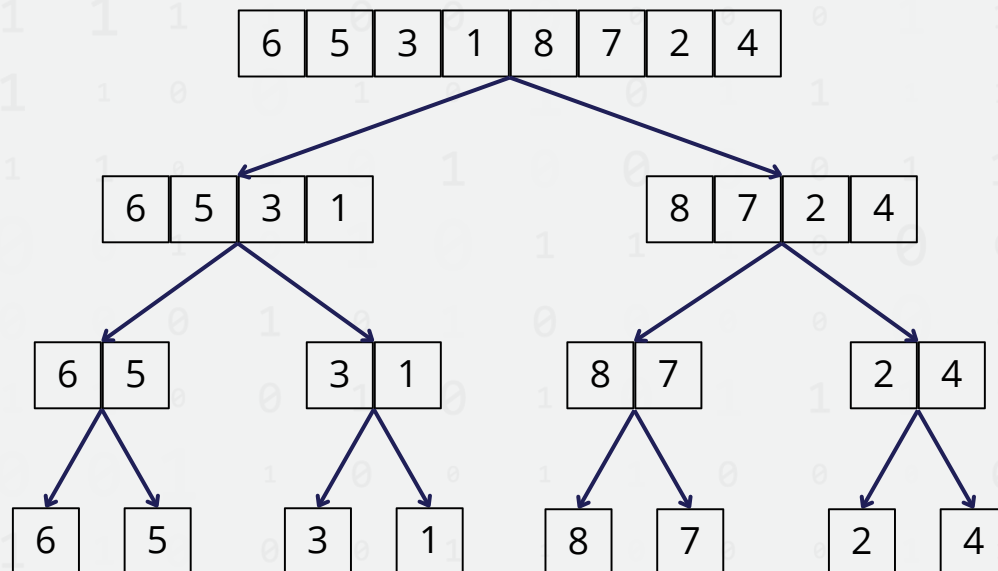
sortowania przez wstawianie to  **$O(1)$** , co oznacza, że algorytm ten **nie wymaga** dodatkowej pamięci poza przechowywaną listą.

# SORTOWANIE PRZEZ SCALANIE (MERGE SORT)

- Merge sort dzieli listę na coraz mniejsze części, wykorzystując zasadę “**dziel i zwyciężaj**”, aż każda ma jeden element. Następnie łączy te części, porównując elementy i układając je w odpowiedniej kolejności, aż powstanie posortowana lista.
- Dokładne kroki algorytmu:
  - **Podział:** Podziel zbiór danych na mniejsze podzbiory, dzieląc go na pół (Dziel), aż każdy zbiór ma 1 element.
  - **Scalanie:** Połącz pary podzbiorów, porównując pierwszą wartość z każdą z inną wartością w kolejnych zbiorów (Zwyciężaj).
  - **Powtórzenie:** Powtarzaj krok 2, aż wszystkie podzbiory zostaną scalone w jeden, posortowany zbiór.

## WIZUALIZACJA SORTOWANIA PRZEZ SCALANIE

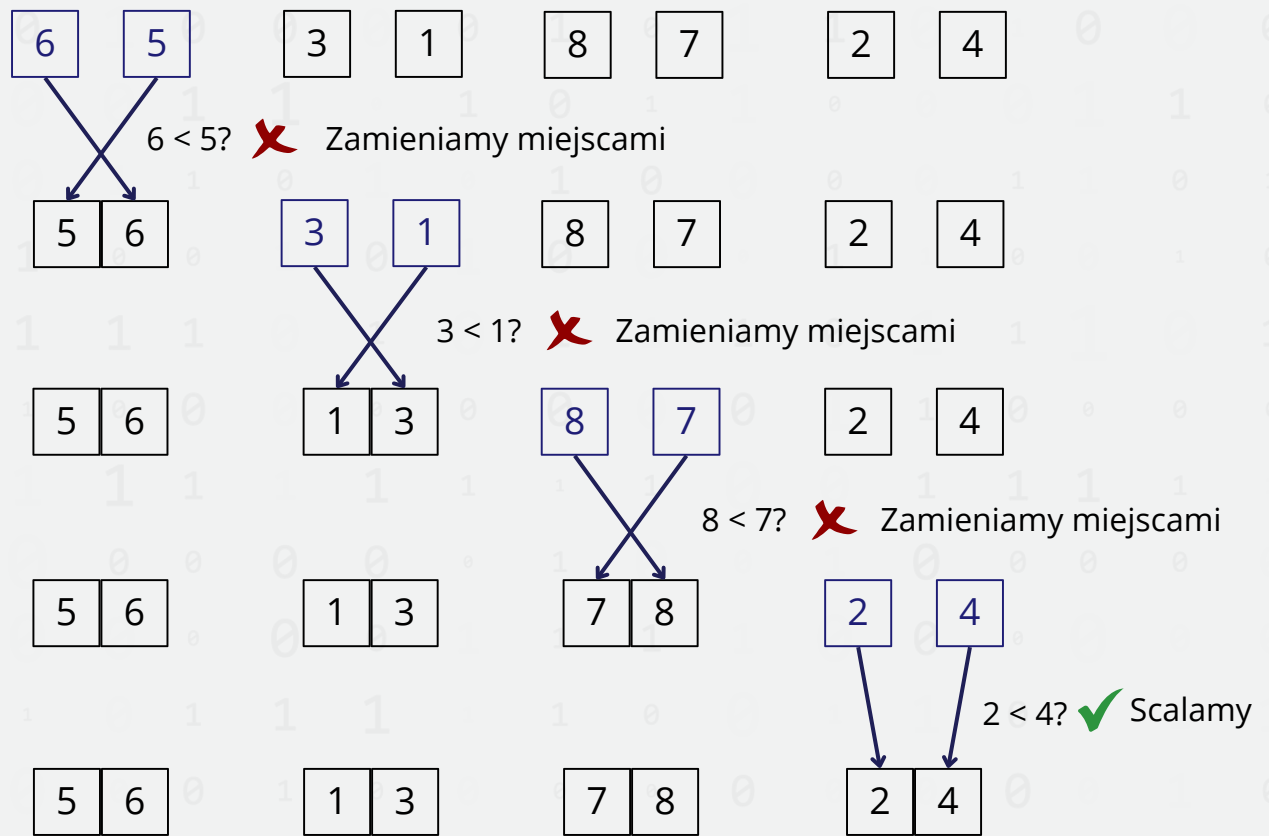
### Podział:



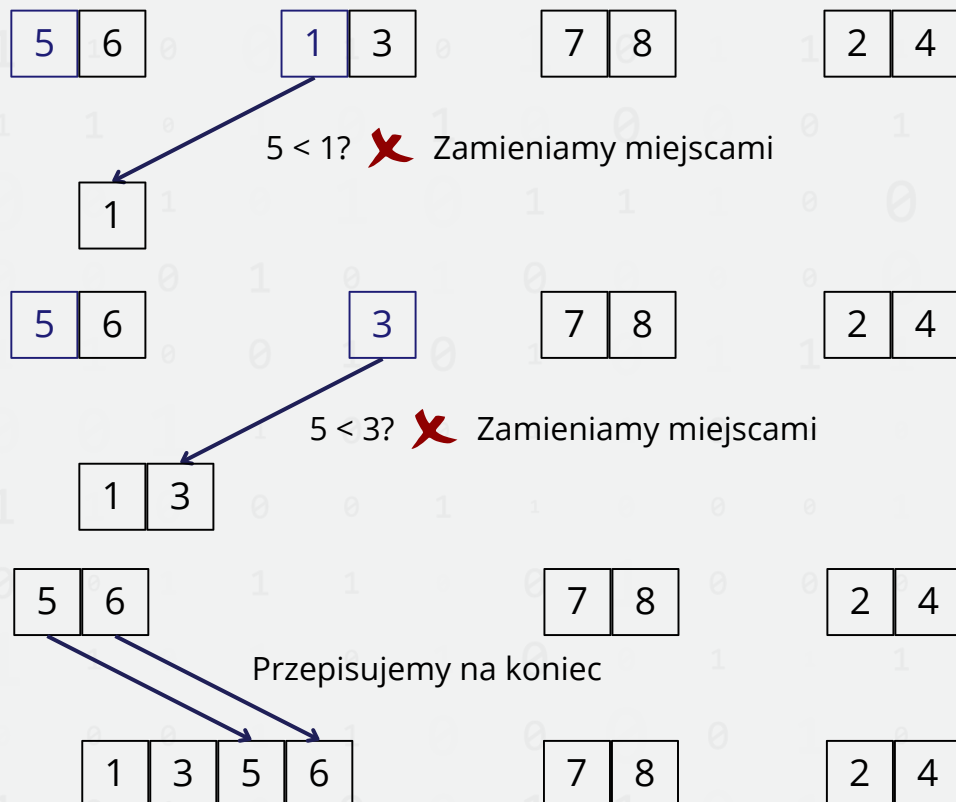
- Podział zakończony, ponieważ otrzymaliśmy jedno elementowe zbiory.

## Scalanie

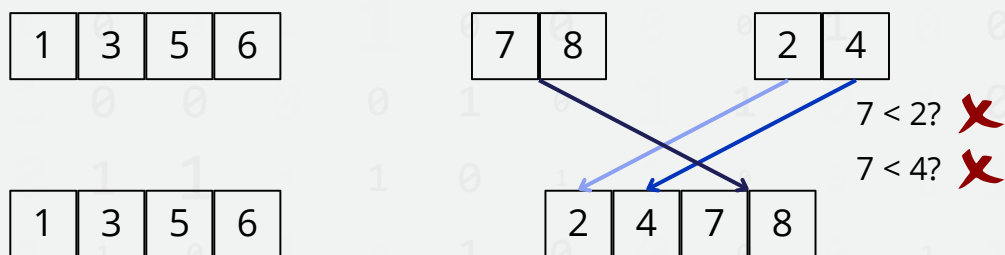
- Porównujemy elementy w każdej parze



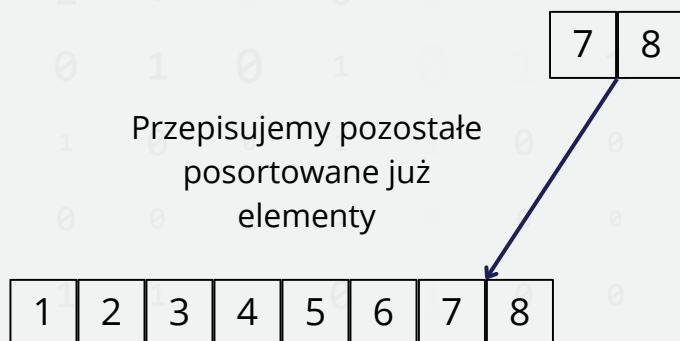
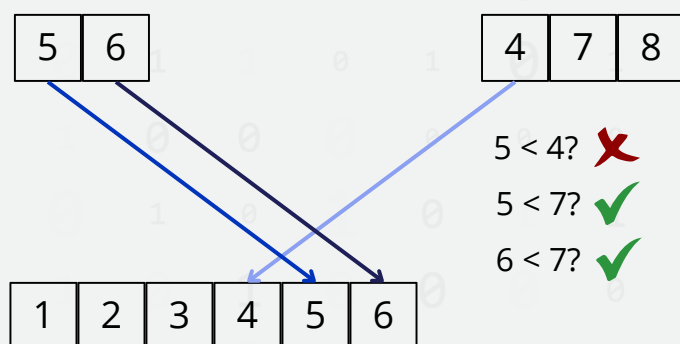
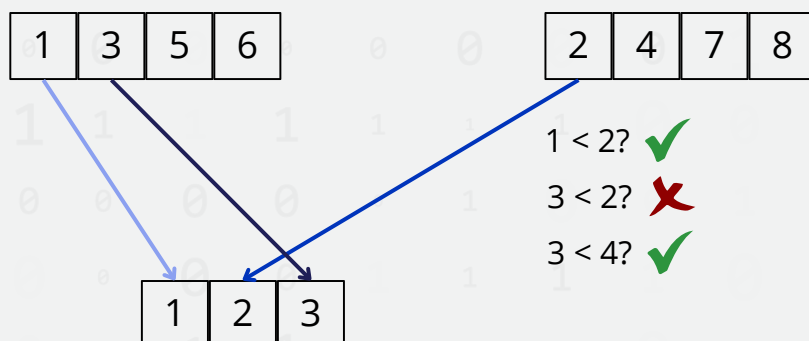
- Porównujemy elementy w 2 zbiorach a następnie scalamy.



## Scalanie

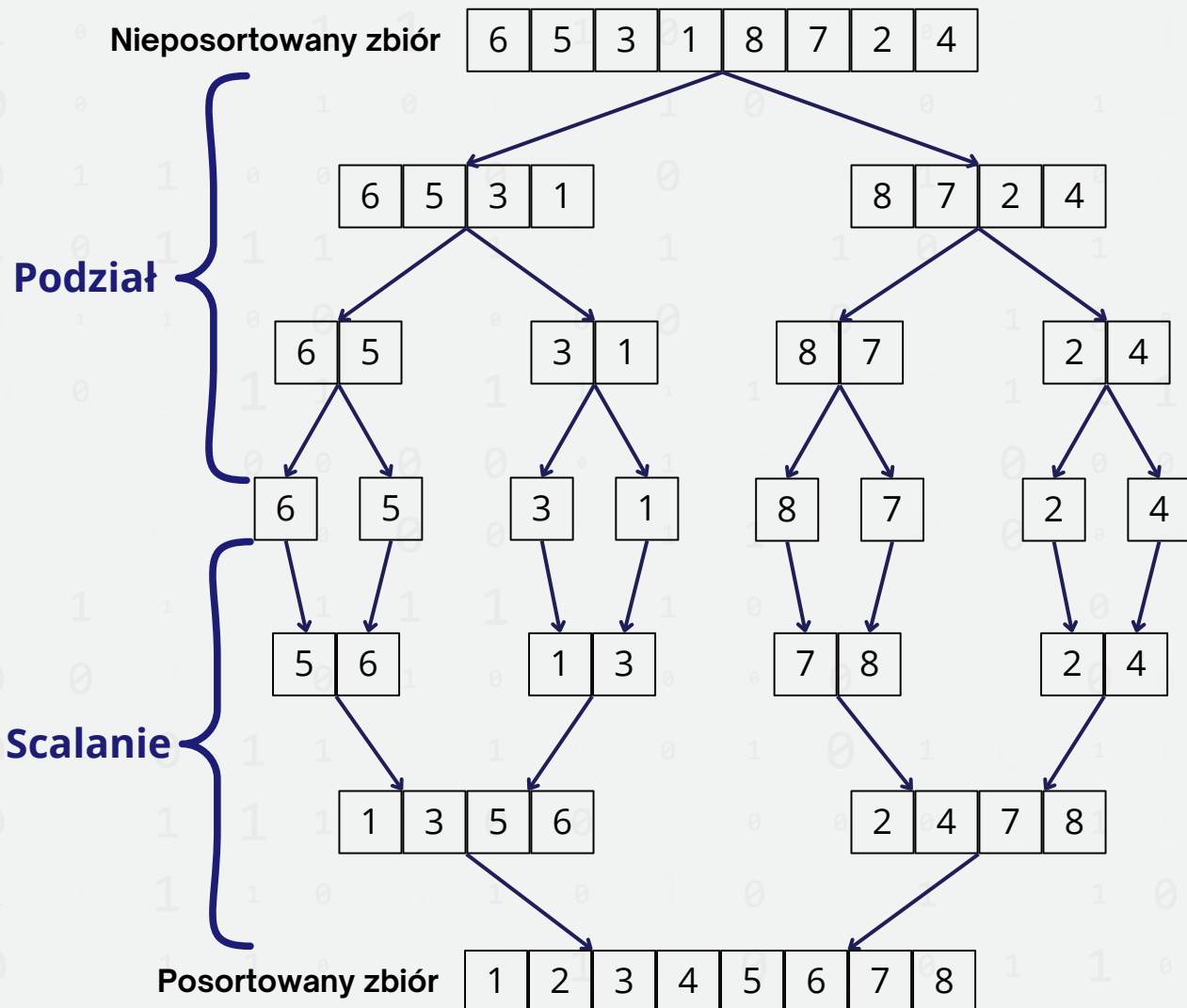


- Porównujemy elementy w 2 zbiorach a następnie scalamy.
- Ten który jest mniejszy wpisujemy w nowy zbiór



Otrzymujemy posortowany zbiór

# UPROSZCZONA WIZUALIZACJA SORTOWANIA PRZEZ SCALANIE



## ZŁOŻONOŚĆ CZASOWA

## ZŁOŻONOŚĆ PAMIĘCIOWA

najlepsza  $\Omega(n \times \log n)$ 
średnia  $\Theta(n \times \log n)$ 
najgorsza  $O(n \times \log n)$

$O(n)$

Algorytm **merge sort** ma złożoność obliczeniową  **$O(n \log n)$**  we wszystkich przypadkach, ponieważ zawsze **dzieli** listę na **połowy** i **scala** je w **porządku rosnącym**.

**Merge sort** wymaga **dodatkowej pamięci** do przechowywania skopiowanych podtablic podczas scalania. Używa przestrzeni  **$O(n)$** , ponieważ potrzebuje miejsca na **tymczasowe przechowywanie elementów**, które są łączone.



# SORTOWANIE SZYBKIE (QUICK SORT)

- Algorytm sortowania wykorzystujący zasadę „**dziel i zwyciężaj**”. Dzieli listę na mniejsze **podlisty** wokół wybranego elementu zwanego **pivotem**.
- Wybór elementu osiowego (pivot):** Algorytm wybiera jeden element z listy, który będzie służył jako punkt podziału. Wybór tego elementu może być losowy lub może to być pierwszy, ostatni, środkowy element z listy.
  - Podział (partitioning):** Lista jest dzielona na dwie podlisty: elementy mniejsze niż pivot są przesuwane na lewą stronę, a elementy większe na prawą stronę pivotu.
  - Rekurencyjne sortowanie:** Obie podlisty są sortowane rekurencyjnie stosując te same kroki 1, 2 i 3.
  - Łączenie:** Ponieważ sortowanie działa w miejscu (in-place), nie jest potrzebne dodatkowe łączenie.
- Algorytm kończy działanie, kiedy wszystkie podlisty składają się z jednego lub żadnego elementu, co oznacza, że są już posortowane. Quick Sort jest szybki i efektywny dla dużych zbiorów danych.

## ZŁOŻONOŚĆ CZASOWA

**najlepsza**      **średnia**      **najgorsza**  
 $\Omega(n \times \log n)$      $\Theta(n \times \log n)$      $O(n^2)$

Typowa złożoność obliczeniowa to  $O(n \log n)$  dla przypadku średniego, ale w najgorszym przypadku może osiągnąć  $O(n^2)$ , zwłaszcza gdy **pivoty** są źle wybrane (np. zawsze najmniejszy lub największy element).

## ZŁOŻONOŚĆ PAMIĘCIOWA

$O(n)$

**Quick Sort** wykorzystuje pamięć stosu do przechowywania danych o rekurencji. Każde rekurencyjne wywołanie zużywa przestrzeń na stosie dla zwracanych adresów i zmiennych lokalnych. W najgorszym przypadku, gdy rekurencja dochodzi do pojedynczego elementu na każdym poziomie (np. gdy pivot jest zawsze największy lub najmniejszy), głębokość stosu może osiągnąć  **$O(n)$** .

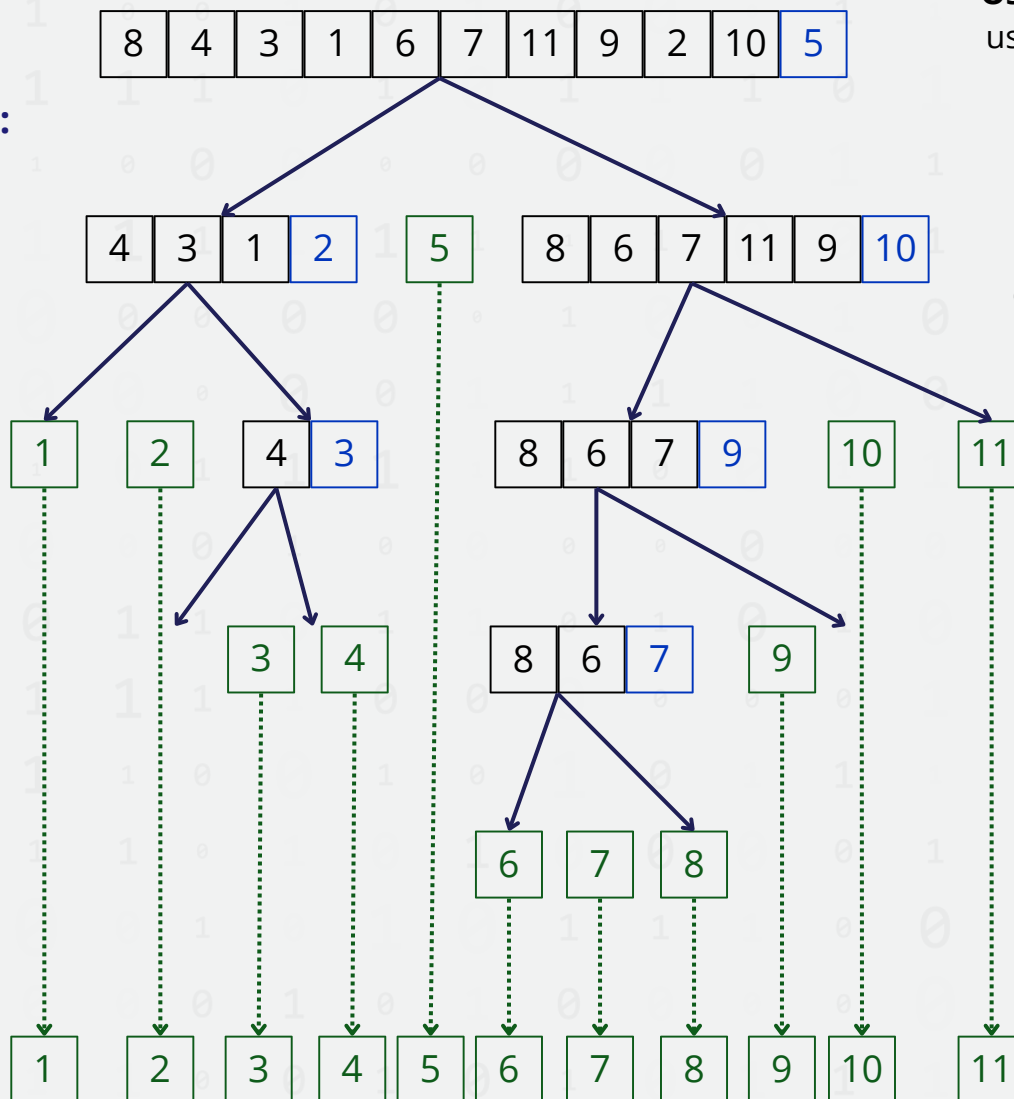


# UPROSZCZONA WIZUALIZACJA SZYBKIEGO SORTOWANIA (QUICK SORT)

**Podział:**

**Ostatni** element  
ustalwiamy jako  
**pivot**

Elementy  
**posortowane**  
oznaczamy na  
**zielono**



**Posortowany zbiór**

|   |   |   |   |   |   |   |   |   |    |    |
|---|---|---|---|---|---|---|---|---|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|----|----|

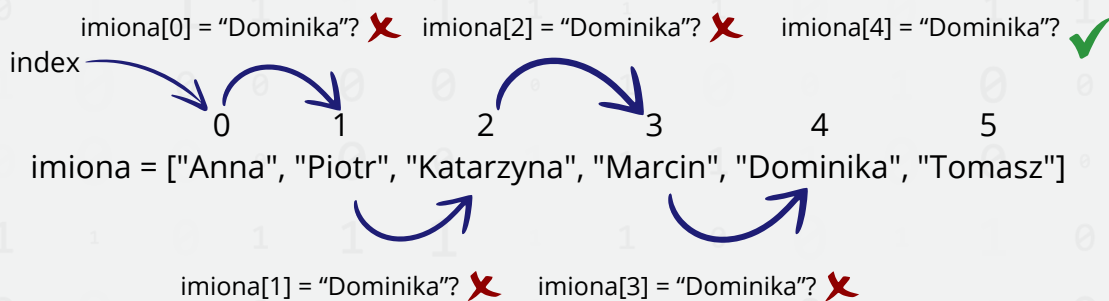
# Algorytmy Przeszukiwania

**Algorytmy przeszukiwania** to metody służące do znalezienia **konkretnego** elementu w zbiorze danych. Mogą one operować na różnorodnych strukturach danych, takich jak listy, tablice, drzewa czy grafy. My skupiamy się jedynie na 2 pierwszych.

## PRZESZUKIWANIE LINIOWE (LINEAR SEARCH)

- Przeszukiwanie listy przez sprawdzanie każdego elementu od początku do końca.

Element do znalezienia: "Dominika"



```
list ← imiona
item ← "Dominika"
index ← -1
i ← 0
found ← False
dopóki i < dł(list) oraz found = False
    jeżeli list[i] = item to:
        index ← i
        found ← True
    i ← i + 1
wypisz index
```

Algorytm rozpoczyna od ustawienia **indeksu na -1**, co oznacza, że element nie został jeszcze znaleziony. Następnie **iteruje** przez każdy element listy od początku do końca. Jeśli znajdzie szukany element, aktualizuje zmienną **index na bieżącą pozycję** i przerywa pętlę. Na zakończenie, wypisuje wartość zmiennej `index`, która pokaże pozycję skutanego elementu w liście

## ZŁOŻONOŚĆ CZASOWA

**najlepsza**

$\Omega(1)$

**średnia**

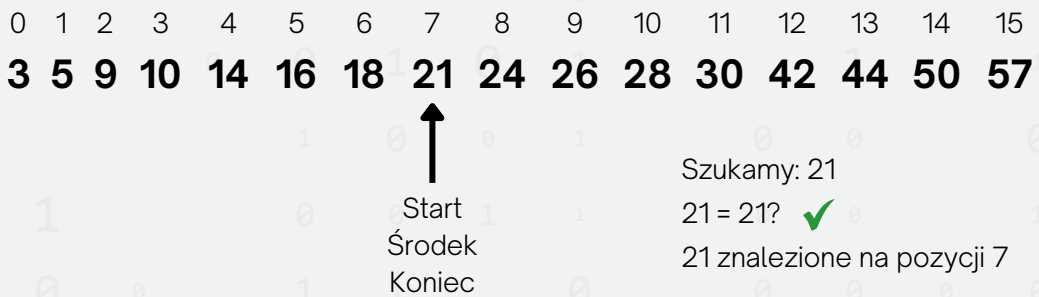
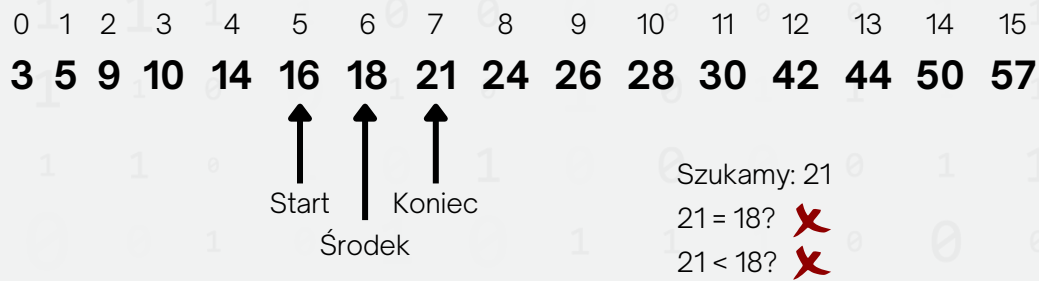
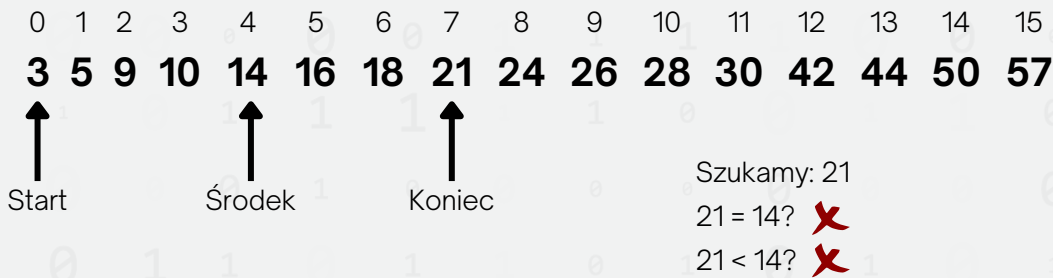
$\Theta(n)$

**najgorsza**

$O(n)$

## PRZESZUKIWANIE BINARNE (BINARY SEARCH)

- Szybki algorytm przeszukiwania **posortowanej** listy poprzez wielokrotne **dzielenie** zakresu przeszukiwania **na pół**.



## PRZESZUKIWANIE BINARNE W PSEUDOKODZIE

- Szybki algorytm przeszukiwania **posortowanej** listy poprzez wielokrotne **dzielenie** zakresu przeszukiwania **na pół**.

```
function binarySearch(list, item)
  ① found ← False
    index ← -1
    start ← 0
    end ← len(list) - 1

  ② dopóki start <= end oraz found = False
    mid ← zaokrągluj((start + end) / 2)
    jeśli list[mid] = item to
      found ← True
      index ← mid
    w przeciwnym wypadku
      ③ jeśli list[mid] < item to
        start ← mid + 1
      w przeciwnym wypadku
        end ← mid - 1

  ④ zwróć index
```

### Inicjalizacja zmiennych:

- **found** to zmienna typu boolowskiego, która informuje, czy szukany element został znaleziony.
- **index** to zmienna przechowująca indeks znalezionego elementu, inicjalnie ustawiona na **-1** (oznacza to, że element nie został jeszcze znaleziony).
- **start** i **end** to indeksy określające obecnie przeszukiwany zakres listy, początkowo ustawione na początek i koniec listy.

```

function binarySearch(list, item)
  ① found ← False
    index ← -1
    start ← 0
    end ← len(list) - 1

  ② dopóki start <= end oraz found = False
    mid ← zaokrąglaj((start + end) / 2)
    jeśli list[mid] = item to
      found ← True
      index ← mid
    w przeciwnym wypadku
      ③ jeśli list[mid] < item to
        start ← mid + 1
      w przeciwnym wypadku
        end ← mid - 1

  ④ zwróć index

```

#### Wykonanie pętli:

- Pętla wykonuje się, dopóki zmienna **start** jest **mniejsza lub równa end** i element nie został jeszcze znaleziony (**found = False**).
- W każdym obiegu pętli, obliczany jest środek (**mid**) obecnego zakresu przez zaokrąglenie średniej z **start** i **end**.
- Następnie sprawdzane jest, czy element w środku listy (**list[mid]**) jest równy szukanemu elementowi (**item**). Jeśli tak, ustawiane są **found** na **True** i **index** na **mid**.

#### Aktualizacja zakresu:

- Jeżeli element środkowy jest **mniejszy** od szukanego, przeszukiwanie kontynuowane jest w **prawej** połowie listy, aktualizując **start** na **mid + 1**.
- Jeżeli element środkowy jest **większy**, przeszukiwanie skupia się na **lewej** połowie, aktualizując **end** na **mid - 1**.

#### Zakończenie:

- Po zakończeniu pętli, funkcja zwraca **index**. Jeśli element został znaleziony, **index** wskazuje jego pozycję w liście. Jeżeli nie, **index** pozostaje równy **-1**, co oznacza, że element **nie** znajduje się w liście.

## ZŁOŻONOŚĆ CZASOWA

**najlepsza**

$\Omega(1)$

**średnia**

$\Theta(\log n)$

**najgorsza**

$O(\log n)$

# Algorytmy Kryptograficzne

**Algorytmy kryptograficzne** to metody matematyczne służące do zabezpieczania informacji poprzez szyfrowanie i deszyfrowanie danych. Umożliwiają one bezpieczną komunikację, autoryzację oraz ochronę prywatności i integralności danych.

## KRYPTOGRAFIA SYMETRYCZNA

**Kryptografia symetryczna** to typ kryptografii, w którym do szyfrowania i deszyfrowania danych używany jest **ten sam klucz**.

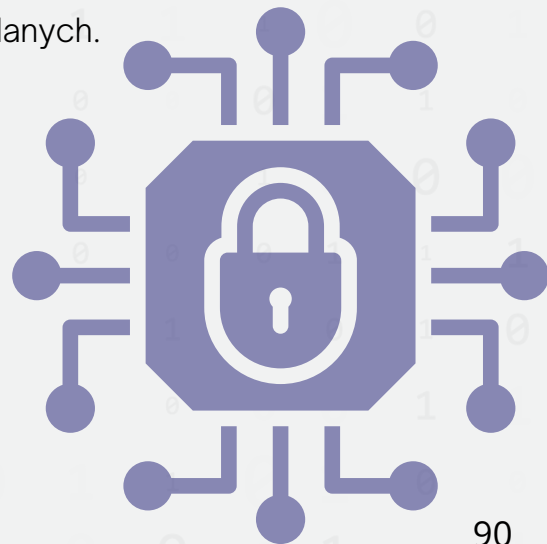
Oto dwa popularne algorytmy symetryczne

### DES (Data Encryption Standard)

- Szyfruje dane w blokach **64-bitowych**.
- Używa klucza o długości **56** bitów.
- Obecnie uważany za nieskuteczny
- ze względu na krótką długość klucza, co czyni go podatnym na ataki typu **brute-force**.

### AES (Advanced Encryption Standard)

- Szyfruje dane w blokach **128-bitowych**.
- Umożliwia użycie kluczy o różnych długościach: **128**, **192**, lub **256** bitów.
- Jest obecnie standardem w bezpiecznym szyfrowaniu danych.



# KRYPTOGRAFIA ASYMETRYCZNA

**Kryptografia asymetryczna**, znana również jako kryptografia klucza publicznego, wykorzystuje parę kluczy – jeden **publiczny** do szyfrowania danych lub weryfikacji, a drugi **prywatny** do ich deszyfrowania lub podpisywania.

## Algorytm RSA

- Jeden z najbardziej rozpowszechnionych i uznanych algorytmów kryptograficznych.
- Wykorzystuje właściwości matematyczne dużych liczb pierwszych oraz ich zastosowanie w funkcjach jednokierunkowych.
- Służy do szyfrowania danych, weryfikacji tożsamości oraz cyfrowego podpisu.

## ElGamal

- Algorytm oparty na trudności obliczeniowej rozwiązania problemu logarytmu dyskretnego w grupie cyklicznej.
- Często stosowany do szyfrowania i systemów cyfrowych podpisów.
- Charakteryzuje się tym, że generuje większe rozmiary szyfrogramów w porównaniu do innych metod, takich jak RSA.

## ECDSA (Elliptic Curve Digital Signature Algorithm)

- Jest to algorytm podpisu cyfrowego, który bazuje na matematyce krzywych eliptycznych.
- Zapewnia wysokie bezpieczeństwo przy stosunkowo krótkich kluczach, co czyni go efektywniejszym w porównaniu z tradycyjnymi algorytmami takimi jak RSA.
- Jest powszechnie używany w nowoczesnych technologiach, takich jak blockchain i kryptowaluty, gdzie ważne są zarówno bezpieczeństwo, jak i wydajność.



# FUNKCJE HASZUJĄCE:

**Funkcje haszujące** to algorytmy matematyczne przekształcające dowolne dane wejściowe na stałodługościowy ciąg bitów, znanym jako skrót (**hash**).

## SHA-256 (Secure Hash Algorithm 256-bit):

- Jest to część rodziny funkcji haszujących **SHA-2**, używana do generowania unikalnych skrótów danych o długości **256 bitów**.
- SHA-256 jest powszechnie stosowany w różnych aplikacjach bezpieczeństwa, w certyfikatach SSL, systemach blockchain oraz jako część procesów weryfikacji integralności danych.

## MD5 (Message Digest Algorithm 5):

- MD5 był szeroko stosowany do tworzenia skrótów danych, szczególnie do weryfikacji integralności plików.
- Obecnie uznawany jest za nieskuteczny ze względu na szybkość, z jaką mogą być generowane kolizje (dwa różne wejścia produkujące ten sam skrót), co czyni go podatnym na ataki i niebezpiecznym w zastosowaniach wymagających wysokiego poziomu bezpieczeństwa.



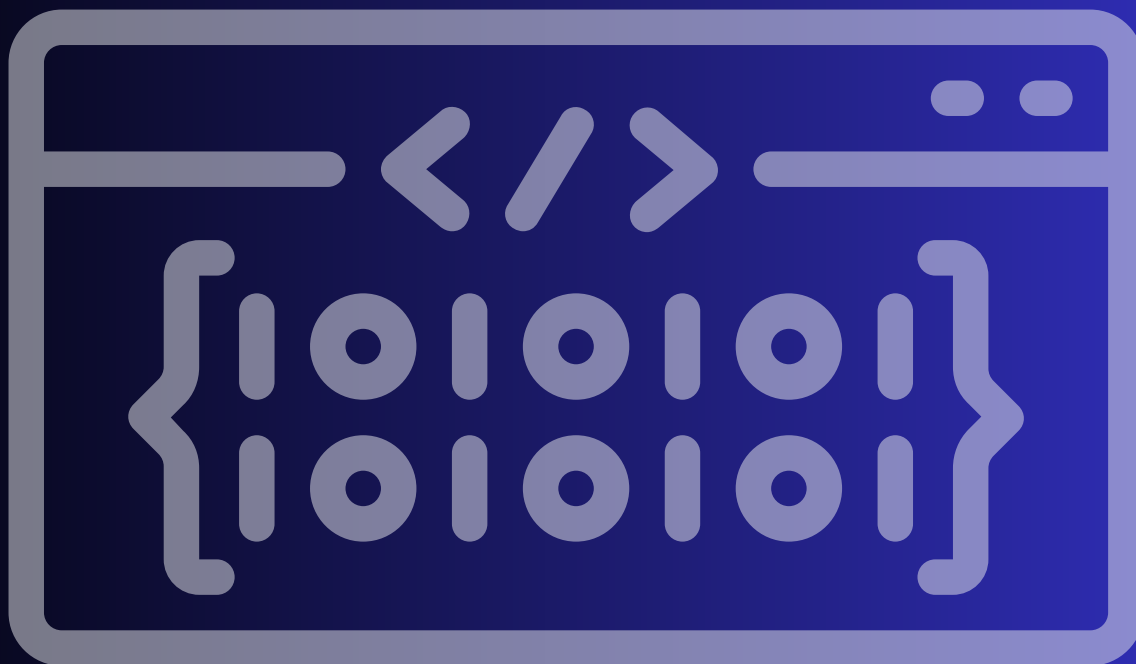
101001011



# OCEŃ PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/FsR2nNLo3Lzfowus9>

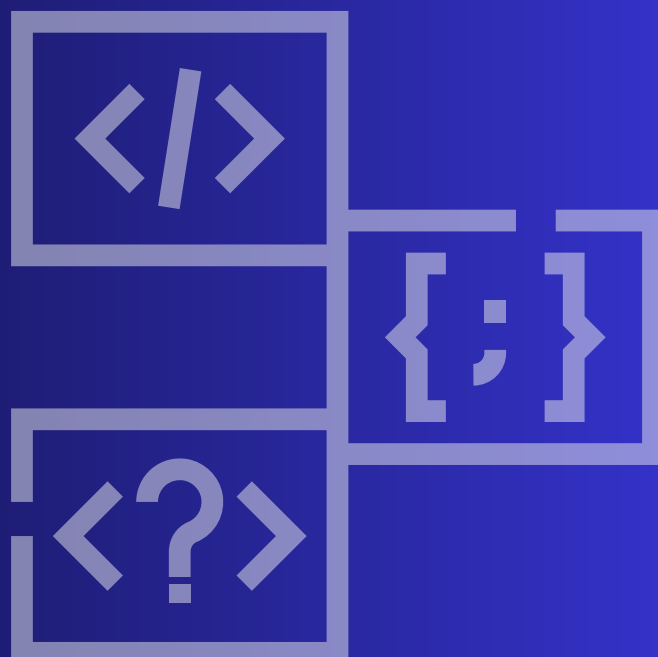


CHAPTER N.12

# Języki Programowania



Definicja, Podział na poziomy,  
Sposób wykonywania programów



# Języki Programowania

**Język programowania** to formalny język zawierający instrukcje, które umożliwiają tworzenie oprogramowania. Służy do komunikacji z komputerem, pozwalając programistom pisać kod wykonujący zadania, od prostych obliczeń po złożone algorytmy.

## PODZIAŁ JĘZYKÓW NA POZIOM

Podział języków programowania na **wysokopoziomowe**, **średniopoziomowe** i **niskopoziomowe** odnosi się do stopnia abstrakcji, jaki oferują one w odniesieniu do sprzętu komputerowego. Oto, co te pojęcia oznaczają:

### Wysokopoziomowe języki programowania

- **Charakterystyka:** Oferują wysoką abstrakcję od sprzętu, co oznacza, że są bliższe językowi naturalnemu niż maszynowemu.
- **Zalety:** Łatwiejsze do nauki i zrozumienia, co pozwala na szybsze tworzenie oprogramowania. Zazwyczaj zawierają bogate biblioteki i narzędzia, które upraszczają rozwój aplikacji.
- **Zastosowania:** Używane do szerokiego zakresu zastosowań, od aplikacji internetowych po oprogramowanie mobilne i gry komputerowe.



#### Python

- **Zastosowania:** Data Science, AI, automatyzacja, web development



#### JavaScript

- **Zastosowania:** Web development (front-end i back-end), aplikacje mobilne, serwery (Node.js).



#### Java

- **Zastosowania:** Aplikacje webowe, aplikacje mobilne (Android), systemy wbudowane, aplikacje serwerowe



#### C#

- **Zastosowania:** Aplikacje desktopowe, webowe, gry (Unity), aplikacje mobilne (Xamarin).

# Wysokopoziomowe języki programowania



## PHP

- **Zastosowania:** Web development, systemy CMS, aplikacje serwerowe



## Ruby

- **Zastosowania:** Web development (Ruby on Rails)



## Swift

- **Zastosowania:** Aplikacje na iOS i macOS



## Kotlin

- **Zastosowania:** Aplikacje na Androida, aplikacje webowe, aplikacje desktopowe.



## R

- **Zastosowania:** Data Science, statystyka



## Go (Golang)

- **Zastosowania:** Aplikacje serwerowe, mikroserwisy, aplikacje webowe i API



## TypeScript

- **Zastosowania:** Web development (typowana wersja JavaScript)



## SQL

- **Zastosowania:** Zarządzanie bazami danych



## Dart

- **Zastosowania:** Aplikacje mobilne (Flutter), aplikacje webowe, aplikacje desktopowe



## Pascal

- **Zastosowania:** Tworzenia oprogramowania desktopowego

I wiele innych których już nie będę wymieniać ....



## Średniopoziomowe języki programowania

- **Charakterystyka:** łączą cechy języków wysokiego i niskiego poziomu, umożliwiając zarówno programowanie bliskie maszynie, jak i abstrakcyjne.
- **Zalety:** Dają większą kontrolę nad sprzętem niż języki wysokiego poziomu, ale są łatwiejsze w użyciu niż języki niskopoziomowe.
- **Zastosowania:** Często wykorzystywane w programowaniu systemowym, tworzeniu sterowników oraz w miejscach, gdzie wymagana jest wydajność i kontrola nad sprzętem.



### Rust

- **Zastosowania:** Systemy operacyjne, oprogramowanie wbudowane, aplikacje o wysokiej wydajności



### C++

- **Zastosowania:** Systemy operacyjne, systemy wbudowane, aplikacje desktopowe, gry, silniki gier, aplikacje o wysokiej wydajności



### C

- **Zastosowania:** Systemy operacyjne, oprogramowanie wbudowane, tworzenia kompilatorów i interpreterów
- **Poziom:** Średni (blisko niskiego)

I inne których już nie będę wymieniać ....

## Niskopoziomowe języki programowania

- **Charakterystyka:** Bardzo bliskie kodowi maszynowemu, co oznacza minimalną abstrakcję od sprzętu komputerowego.
- **Zalety:** Dają programistom maksymalną kontrolę nad działaniem sprzętu, co jest kluczowe w programowaniu systemowym i operacjach o wysokiej wydajności.
- **Zastosowania:** Używane do programowania systemów operacyjnych, oprogramowania wbudowanego oraz w sytuacjach, gdy wymagana jest optymalizacja pod kątem wydajności sprzętowej.



### Assembly (Asembler)

- **Zastosowania:** Programowanie na poziomie sprzętu, systemy wbudowane

# Niskopoziomowe języki programowania



## VHDL / Verilog

- **Zastosowania:** Projektowanie układów cyfrowych, FPGA, ASIC



## Forth

- **Zastosowania:** Systemy wbudowane, systemy sterowania

I inne których już nie będę wymieniać ....

# Jak Komputer Wykonuje Program w Języku Programowania

## 1. Napisanie Kodów Źródłowych

- Programista pisze kod źródłowy w wybranym języku programowania (np. Python, Java, C++) za pomocą edytora tekstu lub środowiska IDE.

## 2. Kompilacja lub Interpretacja

- **Kompilacja** (np.: C, C++, Java):
  - **Kompilator** tłumaczy kod źródłowy na kod maszynowy, tworząc plik wykonywalny (.exe).
- **Interpretacja** (np.: Python, JavaScript):
  - **Interpreter** tłumaczy i wykonuje kod w czasie rzeczywistym, bez generowania pliku wykonywalnego.

## 3. Linkowanie (dla Języków Kompilowanych)

- Linker łączy **moduły** i **biblioteki**, tworząc finalny plik wykonywalny.

## 4. Załadowanie do Pamięci

- System operacyjny ładuje plik wykonywalny do pamięci **RAM**, przygotowując do uruchomienia.

## 5. Wykonanie Programu

- Procesor zaczyna wykonywać instrukcje od punktu startowego programu, realizując cykl pobierania-dekodowania-wykonania.

## 6. Zarządzanie Zasobami

- System operacyjny zarządza czasem procesora, pamięcią i operacjami wejścia/wyjścia, wspierając synchronizację i komunikację między procesami.

## 7. Zakończenie Programu

- Program kończy działanie, zwracając kod zakończenia i zwalniając zasoby alokowane przez system operacyjny.

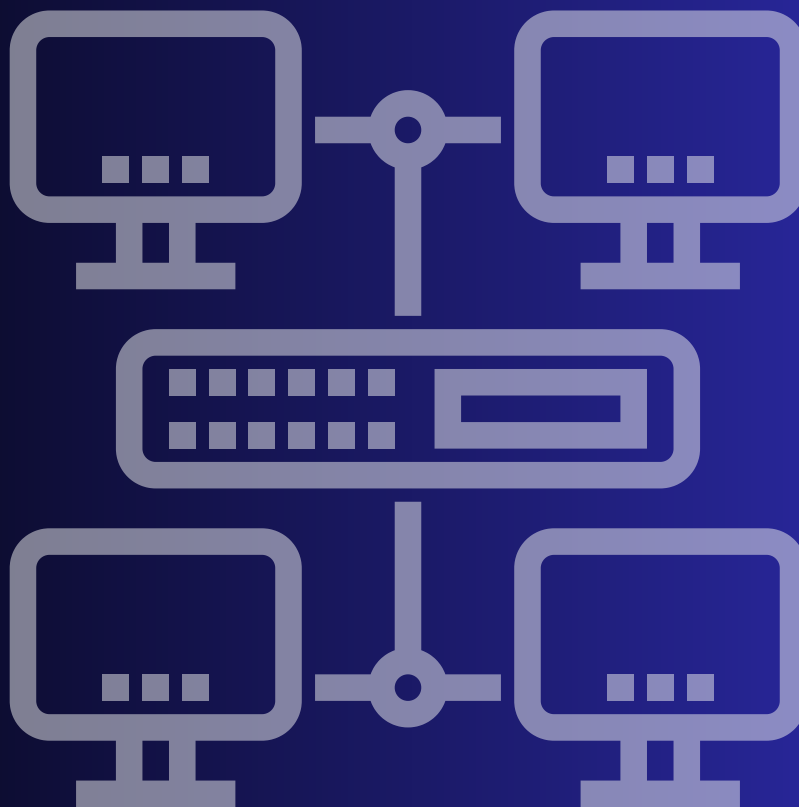


# OCEŃ PROSZĘ TEN ROZDZIAŁ



**InfaMatura**

<https://forms.gle/qBPMhXeM4Pof6J6h9>



CHAPTER N.13

# Sieci Komputerowe



Rodzaje Sieci, Infrastruktura, Urządzenia,  
Technologie połączeń, Identyfikacja,  
Klasy i obliczenie masek IPv4





# Sieci Komputerowe

## Rodzaje sieci:

### LAN (LOCAL AREA NETWORK)

LAN (Local Area Network) to sieć komputerowa łącząca urządzenia na ograniczonym obszarze, jak dom czy biuro, umożliwiającą szybkie przesyłanie danych i udostępnianie zasobów.

### MAN (METROPOLITAN AREA NETWORK)

MAN (Metropolitan Area Network) to sieć komputerowa, która łączy lokalne sieci (LAN) na obszarze miasta lub kampusu, umożliwiając wymianę danych i udostępnianie zasobów na większą skalę.

## Infrastruktura sieciowa

### HOST

Każde urządzenie podłączone do sieci, które może wysyłać lub odbierać dane.

### KARTA SIECIOWA

Karta sieciowa (NIC - Network Interface Card) to urządzenie pozwalające komputerowi łączyć się z siecią. Może być zintegrowana z płytą główną lub podłączana jako oddzielny komponent.



# Urządzenia sieciowe:

## ROUTER

**Router** to urządzenie sieciowe, które łączy różne sieci i kieruje ruch między nimi, decydując o najlepszej trasie dla pakietów danych. Używa adresów **IP** do przesyłania danych między sieciami **lokalnymi i internetem**.

## SWITCH

**Switch** to urządzenie sieciowe łączące urządzenia w **lokalnej sieci (LAN)**, które kieruje **pakiety danych** do właściwych portów na podstawie adresów **MAC**, umożliwiając jednoczesną i wydajną komunikację między nimi.

## HUB

Urządzenie sieciowe, które rozsyła dane do wszystkich podłączonych urządzeń, bez względu na docelowy adres.

## FIREWALL

Firewall to urządzenie lub oprogramowanie zabezpieczające sieć przed nieautoryzowanym dostępem, kontrolując ruch przychodzący i wychodzący.

## MODEM

**Modem** to urządzenie, które przekształca sygnały **cyfrowe** z komputera na sygnały **analogowe** i **odwrotnie**, umożliwiając komunikację z internetem za pośrednictwem linii **telefonicznych, kablowych** lub **innych mediów transmisji**.

# Technologie połączeń:

## ETHERNET

Technologia sieci przewodowych używana do łączenia komputerów w lokalnej sieci.

## WI-FI

Technologia sieci bezprzewodowych umożliwiająca łączenie urządzeń bez kabli.

## Elementy identyfikacyjne i komunikacyjne w sieci:

### ADRES MAC

**Adres MAC (Media Access Control)** to unikalny **identyfikator** przypisany do **każdego interfejsu sieciowego**. Pozwala na jednoznaczną identyfikację urządzeń w sieci **LAN**. Składa się z 6 oktetów (48 bitów).

Przykład adresu MAC: **00:1A:2B:3C:4D:5E**

### IP

**IP (Internet Protocol)** to protokół komunikacyjny, który służy do przesyłania danych w sieci internetowej. Umożliwia **adresowanie** i **routowanie** pakietów danych między urządzeniami, zapewniając ich dostarczenie do odpowiedniego miejsca docelowego. Adres IP **identyfikuje każde urządzenie w sieci**, a istnieją dwie główne wersje adresów IP:

- **IPv4**
- **IPv6**

# IPv4

IPv4 (Internet Protocol version 4) to czwarta wersja protokołu internetowego, która określa sposób **adresowania** i **routowania pakietów danych** w sieciach komputerowych. IPv4 jest najbardziej powszechnie używaną wersją IP.

## Główne cechy IPv4

- **Rozmiar Adresu**
  - długość: **32 bity**
  - Format: zwykle **cztery liczby dziesiętne oddzielone kropkami**, gdzie każda liczba może mieć wartość od **0 do 255**
- **Przestrzeń adresowa**
  - IPv4 oferuje  $2^{32}$  (ponad 4 miliardy) **unikalnych adresów**
- **Oto kilka przykładów adresów IPv4:**
  - 192.168.1.1
  - 10.0.0.1
  - 172.16.0.1
  - 8.8.8.8 (publiczny adres IP Google DNS)

## Maskowanie (maski podsieci) w IPv4

Maskowanie pozwala na **podzielenie** sieci na **mniejsze segmenty** (podsieci) i określenie, która część adresu IP identyfikuje **sieć**, a która część identyfikuje **urządzenia (hosty)** w tej sieci.

- **Struktura Maski Podsieci:**
  - Maska podsieci (subnet mask) jest również 32-bitową liczbą, gdzie bity ustawione na "**1**" oznaczają część adresu IP należącą do **sieci**, a bity ustawione na "**0**" oznaczają część należącą do **hostów**.
  - Maska podsieci może być przedstawiona w formie:
    - dziesiętnej 255.255.255.0
    - binarnej 11111111.11111111.11111111.00000000

# PRZYKŁADY MASKI PODSIECI

## 255.255.255.0

- **Binarnie:** 11111111.11111111.11111111.00000000
- **Opis:**
  - Oznacza, że pierwsze **trzy oktety/części (24 bity)** adresu IP **identyfikują sieć**, a **ostatni** oktet (8 bitów) identyfikuje **hosty** czyli **urządzenia**.
  - na chłopski rozum **dużo** sieci, **mało** urządzeń (hostów)

## 255.255.0.0

- **Binarnie:** 11111111.11111111.00000000.00000000
- **Opis:**
  - Oznacza, że pierwsze **dwa oktety** (16 bitów) adresu IP identyfikują **sieć**, a ostatnie **dwa oktety** (16 bitów) identyfikują **hosty**.
  - na chłopski rozum **średnia ilość** sieci, **średnia ilość** urządzeń (hostów)

## 255.0.0.0

- **Binarnie:** 11111111.00000000.00000000.00000000
- **Opis:**
  - Oznacza, że **pierwszy oktet** (8 bitów) adresu IP identyfikuje **sieć**, a pozostałe **trzy oktety** (24 bity) identyfikują **hosty**.
  - na chłopski rozum **mało** sieci, **dużo** urządzeń (hostów)

# KLASY ADRESÓW IPv4

## Uzupełnienie

127.0.0.1 to pętla zwrotna, dlatego adresy 127.n.n.n nie są używane w adresacji hostów. Tak samo 0.n.n.n nie jest używany.

## Klasa A

- **Zakres adresów:** 1.0.0.0 - 126.255.255.255
- Domyślna **maska podsieci:** 255.0.0.0 (/8 w notacji CIDR)
- Liczba **sieci:** 126
- Liczba **hostów** na sieć: 16,777,214
- **Zastosowanie:** Bardzo duże sieci, np. globalni dostawcy usług internetowych.

## Klasa B

- **Zakres adresów:** 128.0.0.0 - 191.255.255.255
- Domyślna **maska podsieci:** 255.255.0.0 (/16)
- Liczba **sieci:** 16 382
- Liczba **hostów** na sieć: 65,534
- **Zastosowanie:** Średnie i duże sieci, np. duże przedsiębiorstwa.

## Klasa C

- **Zakres adresów:** 192.0.0.0 - 223.255.255.0
- Domyślna **maska podsieci:** 255.255.255.0 (/24)
- Liczba **sieci:** czyli 2 097 152
- Liczba **hostów** na sieć: 254
- **Zastosowanie:** Małe sieci, np. małe firmy i domy.

## Klasa D

- **Zakres adresów:** 224.0.0.0 - 239.255.255.255
- **Zastosowanie:** Multicasting (przesyłanie danych do wielu odbiorców jednocześnie).

## Klasa E

- **Zakres adresów:** 240.0.0.0 - 255.255.255.255
- **Zastosowanie:** Do celów eksperymentalnych i badawczych.

# PRZYKŁAD DZIAŁANIA MASKI PODSIECI

Chcemy dowiedzieć się jaki będzie Adres Sieci:

**Adres IP:** 192.168.1.10

**Maska Podsieci:** 255.255.255.0

- Adres IP binarnie: 11000000.10101000.00000001.00001010
- Maska Podsieci binarnie: 11111111.11111111.11111111.00000000

**Obliczanie Adresu Sieci:**

- Adres sieci uzyskuje się poprzez wykonanie operacji logicznej AND pomiędzy adresem IP a maską podsieci.

11000000.10101000.00000001.00001010 (Adres IP)  
AND 11111111.11111111.11111111.00000000 (Maska Podsieci)  
-----  
11000000.10101000.00000001.00000000 (Adres Sieci)

**Bramka AND**

| A | B | Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

0 AND 0 → 0

wynik na podstawie bramki AND

**Adres Sieci binarnie:** 11000000.10101000.00000001.00000000

**Adres Sieci:** 192.168.1.0

## Przykład 2

**Adres IP:** 172.16.45.67

**Maska Podsieci:** 255.255.240.0

- Adres IP binarnie: 10101100.00010000.00101101.01000011
- Maska Podsieci binarnie: 11111111.11111111.11110000.00000000

|     |                                     |                  |
|-----|-------------------------------------|------------------|
|     | 10101100.00010000.00101101.01000011 | (Adres IP)       |
| AND | 11111111.11111111.11110000.00000000 | (Maska Podsieci) |
|     | <hr/>                               |                  |
|     | 10101100.00010000.00100000.00000000 | (Adres Sieci)    |

**Adres Sieci binarnie:** 10101100.00010000.00100000.00000000

**Adres Sieci:** 172.16.32.0

## Przykład 3

**Adres IP:** 172.16.45.67

**Maska Podsieci:** 255.255.240.0

- Adres IP binarnie: 10101100.00010000.00101101.01000011
- Maska Podsieci binarnie: 11111111.11111111.11110000.00000000

|     |                                     |
|-----|-------------------------------------|
|     | 10101100.00010000.00101101.01000011 |
| AND | 11111111.11111111.11110000.00000000 |
|     | <hr/>                               |
|     | 10101100.00010000.00100000.00000000 |

**Adres Sieci binarnie:** 10101100.00010000.00100000.00000000

**Adres Sieci:** 172.16.32.0

## Ćwiczenie 13.1

**Oblicz adres sieci**

- Adres IP: 172.18.34.150, Maska Podsieci: 255.255.240.0
- Adres IP: 192.168.200.75, Maska Podsieci: 255.255.255.224
- Adres IP: 10.200.15.132, Maska Podsieci: 255.255.255.128



# IPv6

IPv6 (Internet Protocol version 6) to nowoczesna wersja protokołu internetowego zaprojektowana w celu rozwiązania problemu wyczerpywania się adresów IPv4. IPv6 wprowadza wiele ulepszeń, które czynią go bardziej wydajnym i bezpiecznym w porównaniu z IPv4.

## Główne cechy IPv6

- **Rozmiar Adresu**

- długość: 128 bitów
- Format: : Osiem grup czteroznakowych liczb szesnastkowych oddzielonych dwukropkami

- **Przestrzeń adresowa**

- IPv6 oferuje  $3.4 \times 10^{38}$  (340 undecylionów) unikalnych adresów, co jest ogromnym wzrostem w porównaniu do IPv4 (79 oktylionowo krotnie XD)

- **Oto kilka przykładów adresów IPv6:**

- Pełny adres: 2001:0db8:0000:0042:0000:8a2e:0370:7334
  - Skrócony: 2001:db8:0:42::8a2e:370:7334
- Pełny adres: fe80:0000:0000:0000:0202:b3ff:fe1e:8329
  - Skrócony: fe80::202:b3ff:fe1e:8329
- Pełny adres: 2001:0da8:0000:1234:0000:0000:0000:abcd
  - Skrócony: 2001:da8:0:1234::abcd
- Pełny adres: 2001:0db8:1234:ffff:0000:0000:5678:abcd
  - Skrócony: 2001:db8:1234:ffff::5678:abcd

- **Wyjaśnienie skrótów**

- Usuwanie wiodących zer: W każdej sekcji adresu IPv6 można usunąć wiodące zera. Na przykład 0db8 staje się db8.
- Zastępowanie serii zer: Kolejne grupy zer mogą być zastąpione przez ::, ale tylko raz w całym adresie, np. 0000:0000 staje się ::.

- **Typy Adresów IPv6**

- Adresy unicast: Identyfikują pojedyncze urządzenie w sieci.
- Adresy anycast: Identyfikują grupę urządzeń, gdzie pakiety są dostarczane do najbliższego z nich.
- Adresy multicast: Identyfikują grupę urządzeń, gdzie pakiety są dostarczane do wszystkich członków grupy.

## Prefiksy IPv6

Prefiksy IPv6 są używane do identyfikacji sieci. Podobnie jak w IPv4, prefiksy są zapisywane z użyciem notacji CIDR (Classless Inter-Domain Routing).

- **Przykład:** 2001:db8::/32 oznacza, że pierwsze 32 bity adresu identyfikują sieć, a pozostałe bity są używane do identyfikacji hostów.

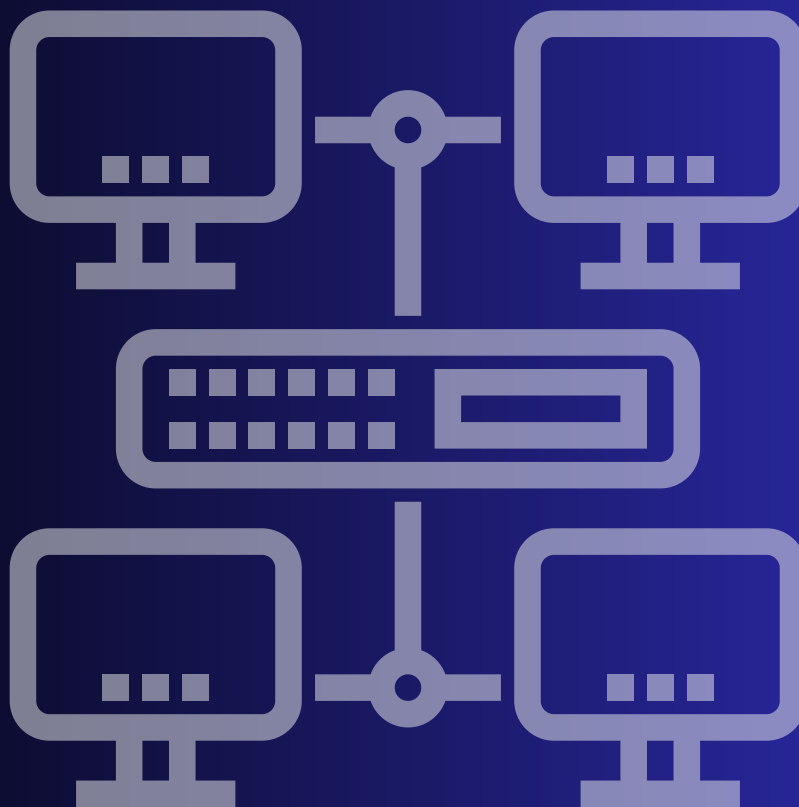
## Mechanizmy IPv6

- Stateless Address Autoconfiguration (SLAAC)
  - Umożliwia urządzeniom automatyczne uzyskiwanie adresów IP bez potrzeby użycia serwera DHCP.
- Network Address Translation (NAT)
  - IPv6 eliminuje potrzebę NAT dzięki ogromnej przestrzeni adresowej, umożliwiając bezpośrednią komunikację end-to-end.
- IPsec (Internet Protocol Security)
  - IPv6 ma wbudowaną obsługę IPsec, co zapewnia bezpieczeństwo na poziomie protokołu.

# OCENIĆ PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/jaLvi64dRNmPm6Aj7>



CHAPTER N.14

# Protokoły Sieciowe



Komunikacja w sieci, Zarządzanie siecią i adresacją,  
Poczta elektroniczna, Zabezpieczenia



# Protokoły Sieciowe

## Protokoły komunikacyjne w sieci:

### HTTP (HYPERTEXT TRANSFER PROTOCOL)

Protokół używany do przesyłania stron internetowych. HTTP umożliwia przeglądarkom pobieranie dokumentów hipertekstowych z serwerów WWW.

### HTTPS (HYPERTEXT TRANSFER PROTOCOL SECURE)

Zabezpieczona wersja HTTP, używa SSL/TLS do szyfrowania danych przesyłanych między przeglądarką a serwerem, co zapewnia poufność i integralność danych.

### FTP (FILE TRANSFER PROTOCOL)

Protokół używany do przesyłania plików między klientem a serwerem. FTP umożliwia zarówno pobieranie, jak i przesyłanie plików.

### VoIP (Voice over Internet Protocol)

VoIP umożliwia przesłanie głosu przez Internet, co pozwala na prowadzenie rozmów telefonicznych przez sieć IP.

### TELNET

Protokół umożliwiający zdalne logowanie do innego komputera w sieci. Telnet przesyła dane w formie tekstowej, co stanowi zagrożenie bezpieczeństwa.

### SSH (SECURE SHELL)

Protokół używany do bezpiecznego logowania się do zdalnych systemów komputerowych oraz do przesyłania plików. SSH szyfruje dane przesyłane między klientem a serwerem.

# Protokoły zarządzania siecią i adresacją:

## DHCP (DYNAMIC HOST CONFIGURATION PROTOCOL)

Protokół używany do automatycznego przydzielania adresów IP oraz innych parametrów sieciowych urządzeniom w sieci, co upraszcza zarządzanie adresami IP.

## DNS (DOMAIN NAME SYSTEM)

System tłumaczący nazwy domenowe (np. [www.przyklad.com](http://www.przyklad.com)) na adresy IP (np. 192.168.1.1), co umożliwia komunikację między urządzeniami w sieci.

## NAT (NETWORK ADDRESS TRANSLATION)

NAT pozwala na mapowanie wielu prywatnych adresów IP na jeden publiczny adres IP, co jest używane do oszczędzania adresów IP oraz zwiększenia bezpieczeństwa sieci.

## SNMP (SIMPLE NETWORK MANAGEMENT PROTOCOL)

Protokół używany do zarządzania i monitorowania urządzeń sieciowych, takich jak routery, przełączniki i serwery.

## NTP (NETWORK TIME PROTOCOL)

Protokół służący do synchronizacji czasu systemowego urządzeń w sieci. NTP umożliwia dokładne ustawienie czasu na komputerach i serwerach.

# Poczta elektroniczna:

## SMTP (SIMPLE MAIL TRANSFER PROTOCOL)

Protokół używany do wysyłania poczty elektronicznej między serwerami pocztowymi. SMTP jest używany przez programy pocztowe do przesyłania wiadomości na serwer.

## IMAP (INTERNET MESSAGE ACCESS PROTOCOL)

Protokół używany do pobierania poczty elektronicznej z serwera. IMAP pozwala na zarządzanie wiadomościami na serwerze i synchronizację skrzynki pocztowej na wielu urządzeniach.

## POP3 (POST OFFICE PROTOCOL 3)

Protokół używany do pobierania poczty elektronicznej z serwera. POP3 pobiera wiadomości na urządzenie klienta i zwykle usuwa je z serwera.

# Protokoły bezpieczeństwa:

## SSL (SECURE SOCKETS LAYER)

Protokół kryptograficzny zapewniający bezpieczne połączenia w sieci. SSL jest używany do szyfrowania danych przesyłanych między klientem a serwerem.

## TLS (TRANSPORT LAYER SECURITY)

Następca SSL, protokół kryptograficzny zapewniający bezpieczne połączenia w sieci. TLS jest używany do szyfrowania danych przesyłanych między klientem a serwerem.

## VPN (VIRTUAL PRIVATE NETWORK)

VPN tworzy bezpieczne połączenie (tunel) między dwoma lub więcej urządzeniami przez Internet, umożliwiając bezpieczny dostęp do sieci prywatnych.

## HTTPS (HYPERTEXT TRANSFER PROTOCOL SECURE)

Zabezpieczona wersja HTTP, używa SSL/TLS do szyfrowania danych przesyłanych między przeglądarką a serwerem, co zapewnia poufność i integralność danych.

## SSH (SECURE SHELL)

Protokół używany do bezpiecznego logowania się do zdalnych systemów komputerowych oraz do przesyłania plików. SSH szyfruje dane przesyłane między klientem a serwerem.



# Inne technologie sieciowe:

## PROXY SERVER

Proxy server działa jako pośrednik między klientem a serwerem, może być używany do filtrowania ruchu, poprawy wydajności i zwiększenia prywatności.

## TCP/IP (TRANSMISSION CONTROL PROTOCOL/INTERNET PROTOCOL)

To zestaw protokołów, który umożliwia komunikację w internecie. TCP dzieli dane na małe pakiety, które są przesyłane przez IP na docelowy adres IP. Na końcu TCP ponownie łączy pakiety, aby dane były pełne.

TCP/IP stanowi podstawę działania internetu.

OCENIĆ PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/itsSEDwqAdZdxug67>



CHAPTER N.15

# Grafika Komputerowa



Grafika Wektorowa, Grafika Rastrowa,  
Obliczenie plików graficznych, Modele Barw



# Grafika Wektorowa

## Definicja:

**Grafika wektorowa** to metoda reprezentacji obrazów przy użyciu **matematycznych równań** do opisu **punktów, linii, krzywych i kształtów**.

## CECHY CHARAKTERYSTYCZNE:

- **Skalowalność:** Obrazy wektorowe mogą być powiększane lub pomniejszane bez utraty jakości, ponieważ są definiowane przez równania matematyczne.
- **Mniejszy Rozmiar Plików:** Proste obrazy i schematy wektorowe zajmują mniej miejsca niż ich rastrowe odpowiedniki.
- **Edytowalność:** Elementy grafiki wektorowej można łatwo edytować, modyfikując ich właściwości geometryczne.

## FORMATY PLIKÓW:

### Popularne formaty plików wektorowych to:

- SVG (Scalable Vector Graphics),
- AI (Adobe Illustrator),
- EPS (Encapsulated PostScript),
- PDF (Portable Document Format).



# DO CZEGO WYKORZYSTYWANA JEST GRAFIKA WEKTOROWA

## **Schematy i Kompozycje Figur Geometrycznych:**

- Idealna do tworzenia schematów, diagramów, wykresów i innych kompozycji geometrycznych. Dzięki precyzyjnym definicjom matematycznym, obrazy te mogą być dokładnie odwzorowane i skalowane.

## **Czcionki:**

- Czcionki komputerowe są często zapisane w formacie wektorowym, co pozwala na ich skalowanie do dowolnej wielkości bez utraty jakości. Dzięki temu czcionki wyglądają ostro na ekranach o różnych rozdzielczościach oraz w druku.

## **Logo i Ikony:**

- Loga firm i ikony interfejsów użytkownika są zazwyczaj tworzone jako grafika wektorowa, ponieważ muszą być wyświetlane w różnych rozmiarach i na różnych nośnikach bez utraty jakości.

## **Ilustracje i Rysunki Techniczne:**

- Wykorzystywana w projektowaniu ilustracji, które wymagają precyzyjnych linii i kształtów, np. w rysunkach technicznych, projektach CAD (Computer-Aided Design) oraz mapach.

## **Animacje i Grafika Interaktywna:**

- W animacjach i grafice interaktywnej (np. na stronach internetowych), gdzie wymagana jest skalowalność i łatwość edycji, często używa się formatów wektorowych jak SVG.

## **Projekty Drukowane:**

- Wykorzystywana w projektowaniu materiałów drukowanych, takich jak plakaty, ulotki, wizytówki, ponieważ zapewnia wysoką jakość druku przy różnych rozmiarach.

# Grafika Rastrowa

## Definicja:

Grafika rastrowa (bitmapowa) to metoda reprezentacji obrazów przy użyciu siatki pikseli, gdzie każdy piksel ma przypisany określony kolor.

## CECHY CHARAKTERYSTYCZNE:

- **Jakość i Szczegółowość:** Grafika rastrowa jest idealna do przedstawiania skomplikowanych obrazów z dużą ilością szczegółów, takich jak fotografie.
- **Pikselizacja przy Skalowaniu:** Przy powiększaniu obrazu rastrowego, jego jakość ulega pogorszeniu (pikselizacja), ponieważ powiększane są indywidualne piksele.
- **Duże Rozmiary Plików:** Detaliczne obrazy rastrowe mogą zajmować dużo miejsca na dysku, szczególnie bez kompresji.

## FORMATY PLIKÓW:

### Popularne formaty plików rastrowych to:

- BMP (Bitmap),
- JPG/JPEG (Joint Photographic Experts Group),
- PNG (Portable Network Graphics),
- GIF (Graphics Interchange Format)
- TIFF (Tagged Image File Format).



# DO CZEGO WYKORZYSTYWANA JEST GRAFIKA RASTROWA

## **Fotografie i Zdjęcia:**

- Grafika rastrowa jest najczęściej wykorzystywana do przechowywania zdjęć cyfrowych, ponieważ może precyzyjnie oddać szczegóły i kolory.

## **Skanowane Obrazy:**

- Obrazy pochodzące bezpośrednio ze skanera są zapisane w formacie rastrowym, co pozwala na odwzorowanie wszystkich szczegółów zeskanowanego dokumentu lub zdjęcia.

## **Grafika Webowa:**

- W internecie grafika rastrowa jest szeroko stosowana na stronach internetowych do wyświetlania zdjęć, ilustracji i innych obrazów. Format JPG jest popularny do zdjęć, PNG do grafik z przezroczystością, a GIF do prostych animacji.

## **Edytowanie Zdjęć i Grafika Artystyczna:**

- Programy do edycji zdjęć, takie jak Adobe Photoshop, pracują głównie z grafiką rastrową, umożliwiając retuszowanie, manipulację kolorami i dodawanie efektów.

## **Materiały Drukowane:**

- Wydruki zdjęć i grafik w książkach, magazynach i plakatach często korzystają z obrazów rastrowych, aby zachować wysoką jakość i szczegółowość.

## **Ekrany Komputerowe i Telewizory:**

- Wszystkie ekrany cyfrowe wyświetlają obrazy w formie rastrowej, wykorzystując siatkę pikseli do renderowania obrazu.

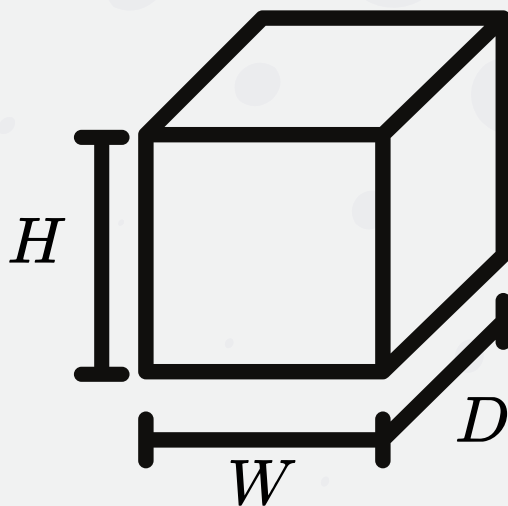
## **Mapy i Obrazy Satelitarne:**

- Mapy i obrazy satelitarne są często przechowywane w formie rastrowej, aby uchwycić wszystkie detale terenu i infrastruktury.

# OBLICZANIE LICZBY BITÓW

$$\text{Rozmiar w bitach} = W \times H \times D$$

- **Szerokość obrazu (W):** liczba pikseli szerokości obrazu.
- **Wysokość obrazu (H):** liczba pikseli wysokości obrazu.
- **Głębina kolorów (D):** liczba bitów na piksel, która określa, ile bitów jest używanych do reprezentacji koloru jednego piksela.



Mamy plik graficzny o rozmiarze 400 na 200 pikseli zapisanym z użyciem 256 kolorów

$W = 400$

$H = 200$

$D = 256$

$$400 \times 200 \times 256 = 20\,480\,000 \text{ bitów}$$

Plik graficzny ma rozmiar:

- 20 480 000 bitów
- $20\,480\,000 / 8 = 2\,560\,000$  bajtów
- $2\,560\,000 / 1024 = 2\,500$  kilobajtów



# RGB (RED, GREEN, BLUE):

## Definicja:

- RGB to model barw używany na ekranach cyfrowych.

## Kolory Podstawowe:

- Czerwony (**Red**), zielony (**Green**), niebieski (**Blue**).

## Mieszanie Kolorów:

- Dodawanie trzech podstawowych kolorów w różnych intensywnościach tworzy inne kolory.
- Wszystkie w maksymalnej intensywności dają **biały**. Brak wszystkich trzech kolorów daje **czarny**.

## Reprezentacja Kolorów:

- Hex: #RRGGBB (np. #FF0000 dla czerwonego).
- Funkcja CSS: rgb(R, G, B) (np. rgb(255, 0, 0) dla czerwonego).

## Zastosowania:

- Wyświetlacze komputerowe, telewizory, projektory, kamery.

## Charakterystyka:

- Model addytywny (dodawanie światła).
- Idealny do wyświetlania obrazów na ekranach.

## Przykłady

- **Czerwony**: #FF0000, rgb(255, 0, 0)
- **Zielony**: #00FF00, rgb(0, 255, 0)
- **Niebieski**: #0000FF, rgb(0, 0, 255)
- **Biały**: #FFFFFF, rgb(255, 255, 255)
- **Czarny**: #000000, rgb(0, 0, 0)



# CMYK (CYAN, MAGENTA, YELLOW, KEY/BLACK)

## Definicja:

- Model barw używany w druku.

## Kolory Podstawowe:

- Cyjan (Cyan), magenta (Magenta), żółty (Yellow), czarny (Key/Black).

## Mieszanie Kolorów:

- Kolory są tworzone przez odejmowanie światła, nakładając tusze.
- Nałożenie wszystkich trzech kolorów (C, M, Y) w maksymalnej intensywności teoretycznie daje czarny, ale w praktyce dodaje się czarny tusz (K) dla głębszej czerni i lepszego kontrastu.

## Zastosowania:

- Drukowanie na papierze: książki, magazyny, plakaty, ulotki.

## Charakterystyka:

- Model subtraktywny (odejmowanie światła).
- Dokładne odwzorowanie kolorów w procesie drukowania.

## Przykłady Mieszania Kolorów

- Czerwony: Nałożenie magenty (M) i żółtego (Y).
- Zielony: Nałożenie cyjanu (C) i żółtego (Y).
- Niebieski: Nałożenie cyjanu (C) i magenty (M).
- Czarny: Nałożenie cyjanu (C), magenty (M) i żółtego (Y), oraz dodatkowy tusz czarny (K) dla głębszej czerni.



# OCENIĆ PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/kj97yotUGAeCXzbd9>



CHAPTER N.16

# Prawa Autorskie i Licencje



Prawa Autorskie, Licencje, Subskrypcje,  
Prawa stron WWW

# PRAWA AUTORSKIE IT

- Ochrona praw autorskich przysługuje twórcy niezależnie od spełnienia jakichkolwiek formalności, takich jak rejestracja czy depozyt utworu. Prawo to jest automatyczne od momentu stworzenia utworu.
- Programy komputerowe są dziełami chronionymi prawami autorskimi. Ochrona ta obejmuje zarówno kod źródłowy, jak i wykonawczy.

## LICENCJE

### **GNU GPL (General Public License):**

- Licencja GNU GPL jest jedną z najbardziej znanych licencji wolnego oprogramowania. Umożliwia użytkownikom uruchamianie, studiowanie, modyfikowanie i rozpowszechnianie oprogramowania, pod warunkiem, że zmiany i dystrybucja są również udostępniane na tej samej licencji.

### **Adware:**

- Oprogramowanie, które jest darmowe, ale wyświetla reklamy. Dochód z reklam jest często używany do finansowania rozwoju oprogramowania. Przykład: Skype w wersji darmowej.

### **Shareware:**

- Oprogramowanie, które można bezpłatnie używać przez określony czas lub z ograniczoną funkcjonalnością. Użytkownicy są zachęceni do płacenia za pełną wersję. Przykład: WinRAR.

### **Freeware:**

- Oprogramowanie, które jest całkowicie darmowe do użytku, ale prawa autorskie są zachowane przez twórcę. Użytkownicy nie mogą modyfikować ani rozpowszechniać zmienionej wersji. Przykład: Adobe Acrobat Reader.

# DODATKOWE LICENCJE

## WOLNE I OTWARTE LICENCJE

### **MIT License:**

- Opis: Bardzo liberalna licencja pozwalająca na dowolne użycie, kopiowanie, modyfikowanie i dystrybucję, pod warunkiem zachowania informacji o prawach autorskich.

### **LGPL (Lesser General Public License):**

- Opis: Podobna do GPL, ale pozwala na linkowanie do bibliotek objętych LGPL bez konieczności udostępniania całego kodu aplikacji na tej samej licencji.
- Przykład: GNU C Library, FFmpeg.

## KOMERCYJNE I SUBSKRYPCYJNE LICENCJE

### **Proprietary License:**

- Opis: Licencje zastrzegające prawa do oprogramowania, zabraniające kopiowania, modyfikowania lub dystrybucji bez zgody właściciela.
- Przykład: Microsoft Windows, Adobe Photoshop.

### **Subscription License:**

- Opis: Umożliwia korzystanie z oprogramowania na zasadzie subskrypcji, gdzie użytkownik płaci regularne opłaty za dostęp do najnowszych wersji i aktualizacji.
- Przykład: Microsoft Office 365, Adobe Creative Cloud.

# SPECJALNE LICENCJE

## Creative Commons (CC):

- Opis: Zestaw licencji umożliwiających twórcom udostępnianie swoich prac z określonymi prawami, np. tylko do użytku niekomercyjnego lub z zachowaniem atrybucji.
- Przykład: Wikipedia (CC BY-SA).

# PRAWA AUTORSKIE STRON WWW

## Ochrona Treści:

- Treści publikowane na stronach WWW, takie jak teksty, obrazy, wideo i kod źródłowy, są chronione prawami autorskimi. Każde nieautoryzowane kopiowanie, modyfikowanie czy rozpowszechnianie jest naruszeniem praw autorskich.

## Prawo Cytatu:

- Możliwe jest używanie fragmentów treści chronionej prawami autorskimi na zasadach prawa cytatu, pod warunkiem podania źródła i autora oraz wykorzystania tylko fragmentów w celach takich jak krytyka, analiza czy edukacja.

# CIASTEczKA (COOKIES)

## Ciasteczka (cookies):

- Opis: Małe **pliki tekstowe**, które strony zapisują na Twoim urządzeniu, aby **zapamiętać informacje o Tobie**, takie jak stan **logowania** czy **preferencje**.
- Cookies przesyłane są między **przeglądarką** a **serwerem** w nagłówkach **HTTP** i pomagają witrynom działać sprawniej oraz dostosowywać treści do użytkownika.

# OCENIĆ PROSZĘ TEN ROZDZIAŁ



**InfaMatura**

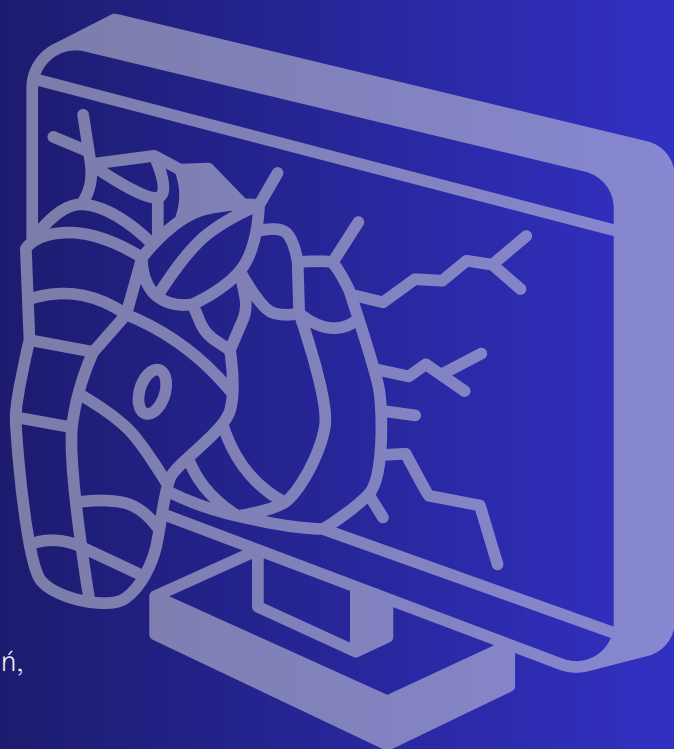
<https://forms.gle/kuqLauTzjpW2m87VA>





CHAPTER N.17

# Wirusy Komputerowe



Czym są i do czego służą, Typy oprogramowań,  
Formy rozprzestrzeniania

# Wirusy Komputerowe

**Wirusy komputerowe** oraz złośliwe oprogramowania (ang. **malware**) to programy stworzone przynajmniej w jednym z tych celów:

- **uszkodzenie systemów komputerowych,**
- **kradzież danych,**
- **uzyskiwanie nieautoryzowanego dostępu do systemów.**

Są one różnorodne pod względem **działania**, sposobu **rozprzestrzeniania** oraz **szkód**, jakie mogą wyrządzić.

## PODSTAWOWE TYPY ZŁOŚLIWEGO OPROGRAMOWANIA

### Wirusy komputerowe

- Są to programy, które dołączają się do plików lub programów w systemie, **aktywują się po ich uruchomieniu**, a następnie mogą **replikować się**, rozprzestrzeniać i powodować szkody. Wirusy często **wymagają interakcji użytkownika** (np. otwarcia pliku) **do aktywacji**.

### Robaki (ang. worms)

- Podobnie jak wirusy, mogą się replikować, ale **nie potrzebują interakcji użytkownika ani dołączania się do innych plików**. Robaki rozprzestrzeniają się automatycznie przez sieci, często wykorzystując luki w zabezpieczeniach oprogramowania. Ich celem jest najczęściej **zakłócanie pracy sieci lub serwerów**.

### Trojan (ang. Trojan Horse)

- **Udaje** przydatne oprogramowanie, ale w rzeczywistości wykonuje złośliwe działania, takie jak **kradzież danych** lub **uzyskiwanie zdalnej kontroli nad komputerem**. Trojany **nie replikują się samodzielnie**, ale często stanowią wektor dla innych rodzajów malware.

### Ransomware

- **Blokuje dostęp do systemu lub plików**, a następnie wymaga okupu za ich odblokowanie. Ransomware stało się szczególnie popularne w ostatnich latach, często wymuszając na ofiarach płacenie dużych sum w kryptowalutach.

### Spyware

- Służy do **szpiegowania użytkownika**, często bez jego wiedzy, zbierając informacje o aktywności, takie jak **dane logowania**, **aktywność internetowa**, czy nawet **dane osobiste**. Tego typu oprogramowanie często jest instalowane razem z innymi programami.

### Adware

- Powoduje **wyświetlanie niechcianych reklam** na komputerze użytkownika. Adware nie zawsze jest szkodliwy, ale może znacząco spowolnić działanie systemu oraz prowadzić do instalacji innych złośliwych programów.

### Rootkit

- Ukrywa swoją obecność w systemie i pozwala atakującemu na **uzyskanie pełnej kontroli nad systemem** bez wiedzy użytkownika. Rootkity są trudne do wykrycia i usunięcia.

### Keylogger

- **Monitoruje i zapisuje wszystkie klawisze naciskane przez użytkownika**. Zwykle jest stosowany do **kradzieży** danych osobowych, takich jak **hasła** i **informacje o kartach kredytowych**.

## NAJCZĘSTSZE METODY ROZPRZESTRZENIANIA MALWARE:

- Załączniki w e-mailach.
- Złośliwe linki w wiadomościach e-mail lub na stronach internetowych.
- Korzystanie z niezabezpieczonych sieci Wi-Fi.
- Podłączanie zainfekowanych nośników (np. pendrive).
- Pobieranie zcrackowanych programów, gier

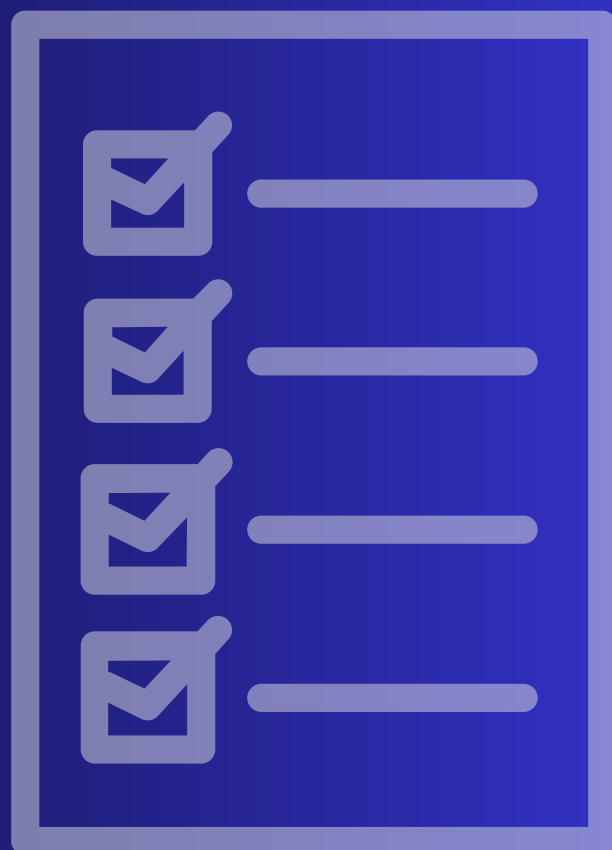
# OCEŃ PROSZĘ TEN ROZDZIAŁ



<https://forms.gle/3n3MRfmZjEQoVM3c8>



# Odpowiedzi



# 1.1 - Zamien z systemu dziesiętnego

## a) 4380 na system piątkowy

$$\begin{array}{l} 4380 \div 5 = 876 \text{ r } 0 \\ 876 \div 5 = 175 \text{ r } 1 \\ 175 \div 5 = 35 \text{ r } 0 \\ 35 \div 5 = 7 \text{ r } 0 \\ 7 \div 5 = 1 \text{ r } 2 \\ 1 \div 5 = 0 \text{ r } 1 \end{array} \quad \begin{array}{l} \uparrow \\ \text{Wynik:} \\ 120010_5 \end{array}$$

## b) 345 na system binarny

$$\begin{array}{l} 345 \div 2 = 172 \text{ r } 1 \\ 172 \div 2 = 86 \text{ r } 0 \\ 86 \div 2 = 43 \text{ r } 0 \\ 43 \div 2 = 21 \text{ r } 1 \\ 21 \div 2 = 10 \text{ r } 1 \\ 10 \div 2 = 5 \text{ r } 0 \\ 5 \div 2 = 2 \text{ r } 1 \\ 2 \div 2 = 1 \text{ r } 0 \\ 1 \div 2 = 0 \text{ r } 1 \end{array} \quad \begin{array}{l} \uparrow \\ \text{Wynik:} \\ 101011001_2 \end{array}$$

## c) 2048 na system szesnastkowy

$$\begin{array}{l} 2048 \div 16 = 128 \text{ r } 0 \\ 128 \div 16 = 8 \text{ r } 0 \\ 8 \div 16 = 0 \text{ r } 8 \end{array} \quad \begin{array}{l} \uparrow \\ \text{Wynik:} \\ 800_{16} \end{array}$$

## d) 5678 na system ósemkowy

$$\begin{array}{l} 5678 \div 8 = 709 \text{ r } 6 \\ 709 \div 8 = 88 \text{ r } 5 \\ 88 \div 8 = 11 \text{ r } 0 \\ 11 \div 8 = 1 \text{ r } 3 \\ 1 \div 8 = 0 \text{ r } 1 \end{array} \quad \begin{array}{l} \uparrow \\ \text{Wynik:} \\ 13056_8 \end{array}$$

## e) 789 na system binarny

$$\begin{array}{l} 789 \div 2 = 394 \text{ r } 1 \\ 394 \div 2 = 197 \text{ r } 0 \\ 197 \div 2 = 98 \text{ r } 1 \\ 98 \div 2 = 49 \text{ r } 0 \\ 49 \div 2 = 24 \text{ r } 1 \\ 24 \div 2 = 12 \text{ r } 0 \\ 12 \div 2 = 6 \text{ r } 0 \\ 6 \div 2 = 3 \text{ r } 0 \\ 3 \div 2 = 1 \text{ r } 1 \\ 1 \div 2 = 0 \text{ r } 1 \end{array} \quad \begin{array}{l} \uparrow \\ \text{Wynik:} \\ 1100010101_2 \end{array}$$

## f) 9999 na system dziewiątkowy

$$\begin{array}{l} 9999 \div 9 = 1111 \text{ r } 0 \\ 1111 \div 9 = 123 \text{ r } 4 \\ 123 \div 9 = 13 \text{ r } 6 \\ 13 \div 9 = 1 \text{ r } 4 \\ 1 \div 9 = 0 \text{ r } 1 \end{array} \quad \begin{array}{l} \uparrow \\ \text{Wynik:} \\ 14640_9 \end{array}$$

## g) 1234 na system trójkowy

$$\begin{array}{l} 1234 \div 3 = 411 \text{ r } 1 \\ 411 \div 3 = 137 \text{ r } 0 \\ 137 \div 3 = 45 \text{ r } 2 \\ 45 \div 3 = 15 \text{ r } 0 \\ 15 \div 3 = 5 \text{ r } 0 \\ 5 \div 3 = 1 \text{ r } 2 \\ 1 \div 3 = 0 \text{ r } 1 \end{array} \quad \begin{array}{l} \uparrow \\ \text{Wynik:} \\ 1200201_3 \end{array}$$

## f) 4096 na System szóstkowy

$$\begin{array}{l} 4096 \div 6 = 682 \text{ r } 4 \\ 682 \div 6 = 113 \text{ r } 4 \\ 113 \div 6 = 18 \text{ r } 5 \\ 18 \div 6 = 3 \text{ r } 0 \\ 3 \div 6 = 0 \text{ r } 3 \end{array} \quad \begin{array}{l} \uparrow \\ \text{Wynik:} \\ 30544_6 \end{array}$$

## 1.2 - Zamień na system dziesiętny

a)  $110101_2$

$$1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = \\ = 32 + 16 + 0 + 4 + 0 + 1 = 53$$

b)  $FFF_{16}$

$$F \cdot 16^2 + F \cdot 16^1 + F \cdot 16^0 = \\ = 15 \cdot 16^2 + 15 \cdot 16^1 + 15 \cdot 16^0 = \\ = 3840 + 240 + 15 = 4095$$

c)  $1267_8$

$$1 \cdot 8^3 + 2 \cdot 8^2 + 6 \cdot 8^1 + 7 \cdot 8^0 = \\ = 512 + 128 + 48 + 7 = 695$$

d)  $432_5$

$$4 \cdot 5^2 + 3 \cdot 5^1 + 2 \cdot 5^0 = \\ = 100 + 15 + 2 = 117$$

e)  $B7_{12}$

$$B \cdot 12^1 + 7 \cdot 12^0 = \\ = 11 \cdot 12^1 + 7 \cdot 12^0 = \\ = 132 + 7 = 139$$

f)  $632_7$

$$6 \cdot 7^2 + 3 \cdot 7^1 + 2 \cdot 7^0 = \\ = 294 + 21 + 2 = 317$$

g)  $12G_{20}$

$$1 \cdot 20^2 + 2 \cdot 20^1 + G \cdot 20^0 = \\ = 1 \cdot 400 + 2 \cdot 20 + 16 \cdot 1 = \\ = 400 + 40 + 16 = 456$$

h)  $1110100101_2$

$$1 \cdot 2^9 + 1 \cdot 2^8 + 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + \\ + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = \\ = 512 + 256 + 128 + 0 + 32 + 0 + 0 + 4 + 0 + 1 = 933$$

## 1.3 - Konwersja szybkim sposobem

a)  $101011_2$  na system 16, 8 i 4

16)  $101011_2 = 00101011_2 \rightarrow 0010 \ 1011$

$$0010_2 \rightarrow 2_{16}$$

$$1011_2 \rightarrow B_{16}$$

**Wynik:**  $2B_{16}$

4)  $101011_2 \rightarrow 10 \ 10 \ 11$

$$10_2 \rightarrow 2_4$$

$$10_2 \rightarrow 2_4$$

$$11_2 \rightarrow 3_4$$

**Wynik:**  $223_4$

8)  $101011_2 \rightarrow 101 \ 011$

$$101_2 \rightarrow 5_8$$

$$011_2 \rightarrow 3_8$$

**Wynik:**  $53_8$

b)  $2011_3$  na system 9

$$2011_3 \rightarrow 20 \ 11$$

$$20_3 \rightarrow 6_9$$

$$11_3 \rightarrow 4_9$$

**Wynik:**  $64_9$

c)  $ADA_{16}$  na system 2 i 4

$$2) \ ADA_{16} \rightarrow A \ D \ A$$

$$A_{16} \rightarrow 1010_2$$

$$D_{16} \rightarrow 1101_2$$

$$A_{16} \rightarrow 1010_2$$

**Wynik:**  $101011011010_2$

$$4) \ ADA_{16} \rightarrow A \ D \ A$$

$$A_{16} \rightarrow 22_4$$

$$D_{16} \rightarrow 31_4$$

$$A_{16} \rightarrow 22_4$$

**Wynik:**  $223122_4$

d)  $11010111_2$  na system 16, 8 i 4

$$16) \ 11010111_2 \rightarrow 1101 \ 0111$$

$$1101_2 \rightarrow D_{16}$$

$$0111_2 \rightarrow 7_{16}$$

**Wynik:**  $D7_{16}$

$$8) \ 11010111_2 \rightarrow 011 \ 010 \ 111$$

$$011_2 \rightarrow 3_8$$

$$010_2 \rightarrow 2_8$$

$$111_2 \rightarrow 7_8$$

**Wynik:**  $327_8$

$$4) \ 11010111_2 \rightarrow 11 \ 01 \ 01 \ 11$$

$$11_2 \rightarrow 3_4$$

$$01_2 \rightarrow 1_4$$

$$01_2 \rightarrow 1_4$$

$$11_2 \rightarrow 3_4$$

**Wynik:**  $3113_4$

e)  $3E7_{16}$  na system 2 i 4

$$2) \ 3E7_{16} \rightarrow 3 \ E \ 7$$

$$3_{16} \rightarrow 0011_2$$

$$E_{16} \rightarrow 1110_2$$

$$7_{16} \rightarrow 0111_2$$

**Wynik:**  $001111100111_2 = 1111100111_2$

$$4) \ 3E7_{16} \rightarrow 3 \ E \ 7$$

$$3_{16} \rightarrow 03_4$$

$$E_{16} \rightarrow 32_4$$

$$7_{16} \rightarrow 13_4$$

**Wynik:**  $033213_4 = 33213_4$

f)  $512_8$  na system 2

$$512_8 \rightarrow 5 \ 1 \ 2$$

$$5_8 \rightarrow 101_2$$

$$1_8 \rightarrow 001_2$$

$$2_8 \rightarrow 010_2$$

**Wynik:**  $101001010_2$

g)  $1042_9$  na system 3

$$1042_9 \rightarrow 1 \ 0 \ 4 \ 2$$

$$1_9 \rightarrow 01_3$$

$$0_9 \rightarrow 00_3$$

$$4_9 \rightarrow 11_3$$

$$2_9 \rightarrow 02_3$$

**Wynik:**  $01001102_3 = 1001102_3$

h)  $321_4$  na system 16

$$321_4 = 0321_4 \rightarrow 03 \ 21$$

$$03_4 \rightarrow 3_{16}$$

$$21_4 \rightarrow 9_{16}$$

**Wynik:**  $39_{16}$



# 1.4 - Operacje na systemie binarnym

a)  $100101_2 + 1001011_2 = 1110000_2$

$$\begin{array}{r} \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \\ 100101 \\ + 1001011 \\ \hline 1110000 \end{array}$$

d)  $100110_2 - 100001_2 = 101_2$

$$\begin{array}{r} \phantom{0}^0 \phantom{0}^2 \phantom{0}^0 \phantom{0}^2 \\ 100110 \\ - 10001 \\ \hline 000101 \end{array}$$

b)  $1111_2 + 1111_2 = 11110_2$

$$\begin{array}{r} \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \\ 1111 \\ + 1111 \\ \hline 11110 \end{array}$$

e)  $1111_2 - 111_2 = 1000_2$

$$\begin{array}{r} 1111 \\ - 111 \\ \hline 1000 \end{array}$$

c)  $100110_2 + 110101_2 = 1011011_2$

$$\begin{array}{r} \phantom{0}^1 \phantom{0}^1 \\ 100110 \\ + 110101 \\ \hline 1011011 \end{array}$$

f)  $101010_2 - 101_2 = 100101_2$

$$\begin{array}{r} \phantom{0}^0 \phantom{0}^2 \phantom{0}^0 \phantom{0}^2 \\ 101010 \\ - 101 \\ \hline 100101 \end{array}$$

# 1.5 - Operacje na różnych systemach

a)  $101_3 + 202_3 = 1010_3$

$$\begin{array}{r} \overset{1}{1} \overset{1}{0}1 \\ +202 \\ \hline 1010 \end{array}$$

e)  $726_8 + 652_8 = 1600_8$

$$\begin{array}{r} \overset{1}{7} \overset{1}{2} \overset{1}{6} \\ +652 \\ \hline 1600 \end{array}$$

b)  $123_4 + 321_4 = 1110_4$

$$\begin{array}{r} \overset{1}{1} \overset{1}{2} \overset{1}{3} \\ +321 \\ \hline 1110 \end{array}$$

f)  $312_9 + 231_9 = 543_9$

$$\begin{array}{r} 312 \\ +231 \\ \hline 543 \end{array}$$

c)  $234_5 + 321_5 = 1110_5$

$$\begin{array}{r} \overset{1}{2} \overset{1}{3} \overset{1}{4} \\ +321 \\ \hline 1110 \end{array}$$

g)  $3E4_{16} + 1C5_{16} = 5A9_{16}$

$$\begin{array}{r} \overset{1}{3}E4 \\ +1C5 \\ \hline 5A9 \end{array}$$

d)  $543_6 + 345_6 = 1332_6$

$$\begin{array}{r} \overset{1}{5} \overset{1}{4} \overset{1}{3} \\ +345 \\ \hline 1332 \end{array}$$

$$h) 210202_3 - 102_3 = 210100_3$$

$$\begin{array}{r} 210202 \\ - 102 \\ \hline 210100 \end{array}$$

$$i) 1321_4 - 123_4 = 1132_4$$

$$\begin{array}{r} \phantom{1}^5 \\ 2 \phantom{1}^5 \\ 1321 \\ - 123 \\ \hline 1132 \end{array}$$

$$j) 432_5 - 234_5 = 143_5$$

$$\begin{array}{r} \phantom{4}^7 \\ 3 \phantom{2}^7 \\ 432 \\ - 234 \\ \hline 143 \end{array}$$

$$k) 654_6 - 453_6 = 201_6$$

$$\begin{array}{r} 654 \\ - 453 \\ \hline 201 \end{array}$$

$$l) 1565_8 - 767_8 = 576_8$$

$$\begin{array}{r} \phantom{1}^{12} \phantom{1}^{13} \\ 0 \phantom{4}^4 \phantom{5}^5 \phantom{13}^{13} \\ 1565 \\ - 767 \\ \hline 576 \end{array}$$

$$m) 43112_9 - 234_9 = 42767_9$$

$$\begin{array}{r} \phantom{4}^9 \phantom{3}^9 \\ 2 \phantom{0}^0 \phantom{0}^0 \phantom{11}^{11} \\ 43112 \\ - 234 \\ \hline 42767 \end{array}$$

$$n) 5A9_{16} - 1BF_{16} = 3EA_{16}$$

$$\begin{array}{r} \phantom{5}^{25} \\ 4 \phantom{0}^0 \phantom{25}^{25} \\ 5A9 \\ - 1BF \\ \hline 3EA \end{array}$$

## 2.1 - Zamiana na system U2

a)  $-46_{10}$  na U2

$$\begin{aligned}46_{10} &= 00101110_2 \\00101110_2 &\rightarrow 11010001_2 \\11010001_2 + 1_2 &= 11010010_{U2}\end{aligned}$$

e)  $-56_{10}$  na U2

$$\begin{aligned}56_{10} &= 00111000_2 \\00111000_2 &\rightarrow 11000111_2 \\11000111_2 + 1_2 &= 11001000_{U2}\end{aligned}$$

b)  $-123_{10}$  na U2

$$\begin{aligned}123_{10} &= 01111011_2 \\01111011_2 &\rightarrow 10000100_2 \\10000100_2 + 1_2 &= 10000101_{U2}\end{aligned}$$

f)  $-128_{10}$  na U2

$$\begin{aligned}128_{10} &= 10000000_2 \\10000000_2 &\rightarrow 01111111_2 \\01111111_2 + 1_2 &= 10000000_{U2}\end{aligned}$$

c)  $-14_{10}$  na U2

$$\begin{aligned}14_{10} &= 00001110_2 \\00001110_2 &\rightarrow 11110001_2 \\11110001_2 + 1_2 &= 11110010_{U2}\end{aligned}$$

d)  $-201_{10}$  na U2

$$\begin{aligned}201_{10} &= 0000000011001001_2 \\0000000011001001_2 &\rightarrow 1111111100110110_2 \\1111111100110110_2 + 1_2 &= 1111111100110111_{U2}\end{aligned}$$

Musimy użyć zakresu 16-bitowego,  
ponieważ liczba -201 nie należy do zbioru  
8-bitowego [127,-128]



# 3.1 - Zadania na Bramkach Logicznych

a) 110011 AND 101010

$$\begin{array}{r} 110011 \\ \text{AND } 101010 \\ \hline 100010 \end{array}$$

b) 011001 OR 100110

$$\begin{array}{r} 011001 \\ \text{OR } 100110 \\ \hline 111111 \end{array}$$

c) 101010 XOR 110101

$$\begin{array}{r} 101010 \\ \text{XOR } 110101 \\ \hline 011111 \end{array}$$

d) 11110000 NOT

$$\begin{array}{r} \text{NOT } 11110000 \\ \hline 00001111 \end{array}$$

e) 110011 NAND 101101

$$\begin{array}{r} 110011 \\ \text{NAND } 101101 \\ \hline 011110 \end{array}$$

f) 100110 NOR 011011

$$\begin{array}{r} 100110 \\ \text{NOR } 011011 \\ \hline 000000 \end{array}$$

g) 101001 NOT AND 110110

$$\begin{array}{r} \text{NOT } 101001 \\ \hline 010110 \\ \text{AND } 110110 \\ \hline 010110 \end{array}$$

h) 100101 XNOR 111011

$$\begin{array}{r} 100101 \\ \text{XNOR } 111011 \\ \hline 100001 \end{array}$$

i) 111000 AND 100011 OR 100011 NOT

$$\begin{array}{r} 111000 \\ \text{AND } 100011 \\ \hline 100000 \\ \text{OR } 100011 \\ \hline 100011 \end{array}$$
$$\begin{array}{r} \text{NOT } 100011 \\ \hline 011100 \end{array}$$

i)  $(110101 \text{ XOR } 011010) \text{ NAND } (110101 \text{ NOT})$

|     |               |      |               |
|-----|---------------|------|---------------|
|     | 110101        |      | 101111        |
| XOR | <u>011010</u> | NOT  | <u>110101</u> |
|     | 101111        |      | 001010        |
|     |               | NAND | <u>001010</u> |
|     |               |      | 110101        |

## 6.1 - Obliczanie zadań ONP

a) 3 4 +

| Krok   | Wyrażenie    | Stos       | Oblicznia | Stos |
|--------|--------------|------------|-----------|------|
| 1.     | <u>3</u> 4 + | [3]        |           |      |
| 2.     | <u>4</u> +   | [3, 4]     |           |      |
| 3.     | <u>+</u>     | [3, 4]     | 3+4=7     | [7]  |
| Koniec |              | <u>[7]</u> |           |      |

b) 5 2 8 × +

| Krok   | Wyrażenie        | Stos        | Oblicznia | Stos    |
|--------|------------------|-------------|-----------|---------|
| 1.     | <u>5</u> 2 8 × + | [5]         |           |         |
| 2.     | <u>2</u> 8 × +   | [5, 2]      |           |         |
| 3.     | <u>8</u> × +     | [5, 2, 8]   |           |         |
| 4.     | <u>×</u> +       | [5, 2, 8]   | 2×8=16    | [5, 16] |
| 5.     | <u>+</u>         | [5, 16]     | 5+16=21   | [21]    |
| Koniec |                  | <u>[21]</u> |           |         |

c) 6 2 + 11 +

| Krok   | Wyrażenie         | Stos        | Oblicznia | Stos |
|--------|-------------------|-------------|-----------|------|
| 1.     | <u>6</u> 2 + 11 + | [6]         |           |      |
| 2.     | <u>2</u> + 11 +   | [6, 2]      |           |      |
| 3.     | <u>+</u> 11 +     | [6, 2]      | 6+2=8     | [8]  |
| 4.     | <u>11</u> +       | [8, 11]     |           |      |
| 5.     | <u>+</u>          | [8, 11]     | 8+11=19   | [19] |
| Koniec |                   | <u>[19]</u> |           |      |

d)  $10\ 4 - 2\ 3 + \times$

| Krok   | Wyrażenie                     | Stos        | Oblicznia         | Stos   |
|--------|-------------------------------|-------------|-------------------|--------|
| 1.     | <u>10</u> $4 - 2\ 3 + \times$ | [10]        |                   |        |
| 2.     | <u>4</u> $- 2\ 3 + \times$    | [10, 4]     |                   |        |
| 3.     | <u>-</u> $2\ 3 + \times$      | [10, 4]     | $10-4=6$          | [6]    |
| 4.     | <u>2</u> $3 + \times$         | [6, 2]      |                   |        |
| 5.     | <u>3</u> $+ \times$           | [6, 2, 3]   |                   |        |
| 6.     | <u>+</u> $\times$             | [6, 2, 3]   | $2+3=5$           | [6, 5] |
| 7.     | <u>x</u>                      | [6, 5]      | $6 \times 5 = 30$ | [30]   |
| Koniec |                               | <u>[30]</u> |                   |        |

e)  $100\ 10\ 5 - 2 / \times$

| Krok   | Wyrażenie                       | Stos         | Oblicznia              | Stos       |
|--------|---------------------------------|--------------|------------------------|------------|
| 1.     | <u>100</u> $10\ 5 - 2 / \times$ | [100]        |                        |            |
| 2.     | <u>10</u> $5 - 2 / \times$      | [100, 10]    |                        |            |
| 3.     | <u>5</u> $- 2 / \times$         | [100, 10, 5] |                        |            |
| 4.     | <u>-</u> $2 / \times$           | [100, 10, 5] | $10-5=5$               | [100, 5]   |
| 5.     | <u>2</u> $/ \times$             | [100, 5, 2]  |                        |            |
| 6.     | <u>/</u> $\times$               | [100, 5, 2]  | $5/2=2.5$              | [100, 2.5] |
| 7.     | <u>x</u>                        | [100, 2.5]   | $100 \times 2.5 = 250$ | [250]      |
| Koniec |                                 | <u>[250]</u> |                        |            |

f) 35 10 2 / 4 /×

| Krok   | Wyrażenie             | Stos           | Oblicznia     | Stos       |
|--------|-----------------------|----------------|---------------|------------|
| 1.     | <u>35</u> 10 2 / 4 /× | [35]           |               |            |
| 2.     | 10 2 / 4 /×           | [35, 10]       |               |            |
| 3.     | 2 / 4 /×              | [35, 10, 2]    |               |            |
| 4.     | / 4 /×                | [35, 10, 2]    | 10/2=5        | [35, 5]    |
| 5.     | <u>4</u> /×           | [35, 5, 4]     |               |            |
| 6.     | /×                    | [35, 5, 4]     | 5/4=1.25      | [35, 1.25] |
| 7.     | <u>×</u>              | [35, 1.25]     | 35×1.25=43.75 | [43.75]    |
| Koniec |                       | <u>[43.75]</u> |               |            |



## 10.1 - Obliczanie Rekurencji

$$\begin{cases} g(1) = 3 \\ g(n+1) = 1 - g(n) + n \text{ dla } n \geq 1 \end{cases}$$

$$\boxed{10} = 1 - g(9) + 9 = 1 - 6 + 9 = 4$$

$$\boxed{9} = 1 - g(8) + 8 = 1 - 3 + 8 = 6$$

$$\boxed{8} = 1 - g(7) + 7 = 1 - 5 + 7 = 3$$

$$\boxed{7} = 1 - g(6) + 6 = 1 - 2 + 6 = 5$$

$$\boxed{6} = 1 - g(5) + 5 = 1 - 4 + 5 = 2$$

$$\boxed{5} = 1 - g(4) + 4 = 1 - 1 + 4 = 4$$

$$\boxed{4} = 1 - g(3) + 3 = 1 - 3 + 3 = 1$$

$$\boxed{3} = 1 - g(2) + 2 = 1 - 0 + 2 = 3$$

$$\boxed{2} = 1 - g(1) + 1 = 1 - 2 + 1 = 0$$

$$\boxed{1} = g(1) = 3$$

Oceń prawdziwość podpunktów:

a)  $g(6) = 2$  **P**

b)  $g(3) = g(8)$  **P**

c)  $g(10) = 8$  **F**

d) funkcje  $g$  zwraca jedynie wartości dodatnie **F**

## 13.1 - Obliczanie adresu sieci

a)

**Adres IP:** 172.18.34.150

**Maska Podsieci:** 255.255.240.0

- Adres IP binarnie: 10101100.00010010.00100010.10010110
- Maska Podsieci binarnie: 11111111.11111111.11110000.00000000

$$\begin{array}{r} 10101100.00010010.00100010.10010110 \\ \text{AND } 11111111.11111111.11110000.00000000 \\ \hline 10101100.00010010.00100000.00000000 \end{array}$$

**Adres Sieci binarnie:** 10101100.00010010.00100000.00000000

**Adres Sieci:** 172.18.32.0

b)

**Adres IP:** 192.168.200.75

**Maska Podsieci:** 255.255.255.224

- Adres IP binarnie: 11000000.10101000.11001000.01001011
- Maska Podsieci binarnie: 11111111.11111111.11111111.11100000

$$\begin{array}{r} 11000000.10101000.11001000.01001011 \\ \text{AND } 11111111.11111111.11111111.11100000 \\ \hline 11000000.10101000.11001000.01000000 \end{array}$$

**Adres Sieci binarnie:** 11000000.10101000.11001000.01000000

**Adres Sieci:** 192.168.200.64

c)

**Adres IP:** 10.200.15.132

**Maska Podsieci:** 255.255.255.128

- Adres IP binarnie: 00001010.11001000.00001111.10000100
- Maska Podsieci binarnie: 11111111.11111111.11111111.10000000

00001010.11001000.00001111.10000100  
AND 11111111.11111111.11111111.10000000  
-----  
00001010.11001000.00001111.10000000

**Adres Sieci binarnie:** 00001010.11001000.00001111.10000000

**Adres Sieci:** 10.200.15.128