

Wydział Elektryczny  
Katedra Elektrotechniki Teoretycznej i Metrologii

Materiały do wykładu z przedmiotu:  
**Informatyka**  
**Kod: EDS1A1 007**

## WYKŁAD NR 6

Opracował: **dr inż. Jarosław Forenc**  
**Białystok 2018**

Materiały zostały opracowane w ramach projektu „PB2020 - Zintegrowany Program Rozwoju Politechniki Białostockiej” realizowanego w ramach Działania 3.5 Programu Operacyjnego Wiedza, Edukacja, Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

## Plan wykładu nr 6

- Algorytmy komputerowe
  - definicje, sposoby opisu
  - rekurencja
  - złożoność obliczeniowa
  - algorytmy sortowania (proste wstawianie, proste wybieranie, bąbelkowe, quick-sort)
- Klasyfikacja systemów komputerowych (Flynn)

## Algorytm - definicje

### Definicja 1

- Skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego zadania

### Definicja 2

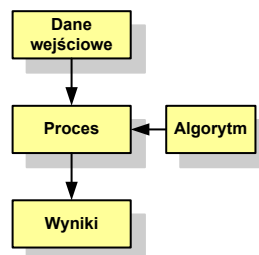
- Opis rozwiązania problemu wyrażony za pomocą operacji zrozumiałych i możliwych do zrealizowania przez wykonawcę

### Definicja 3

- Ścisłe określona procedura obliczeniowa, która dla właściwych danych wejściowych zwraca żądane dane wyjściowe zwane wynikiem działania algorytmu

### Definicja 4

- Metoda rozwiązania zadania



## Algorytmy

- Słowo „algorytm” pochodzi od nazwiska Muhammada ibn-Musy al-Chuwarizmiego (po łacinie pisanego jako **Algorismus**), matematyka perskiego z IX wieku
- Badaniem algorytmów zajmuje się **algorytmika**
- Algorytm może zostać **zaimplementowany** w postaci **programu komputerowego**
- Przetłumaczenie algorytmu na wybrany język programowania nazywane jest też **kodowaniem** algorytmu
- Ten sam algorytm może być zaimplementowany (zakodowany) w różny sposób przy użyciu różnych języków programowania.

## Podstawowe cechy algorytmu

- Posiada dane wejściowe (w ilości większej lub równej zero) pochodzące z dobrze zdefiniowanego zbioru
- Zwraca wynik
- Jest precyzyjnie zdefiniowany (każdy krok algorytmu musi być jednoznacznie określony)
- Poprawność (dla każdego z założonego dopuszczalnego zestawu danych wejściowych)
- Kończy działanie po skończonej liczbie kroków (powinna istnieć poprawnie działająca reguła stopu algorytmu)
- Efektywność (jak najkrótszy czas wykonania i jak najmniejsze zapotrzebowanie na pamięć).

## Sposoby opisu algorytmów

1. Opis słowny w języku naturalnym lub w postaci listy kroków (opis w punktach)
2. Schemat blokowy
3. Pseudokod (nieformalna odmiana języka programowania)
4. Wybrany język programowania

## Opis słowny algorytmu

- Podanie kolejnych czynności, które należy wykonać, aby otrzymać oczekiwany efekt końcowy
- Przypomina przepis kulinarny z książki kucharskiej lub instrukcję obsługi urządzenia, np.

**Algorytm:** Tortilla („Podróże kulinarne” R. Makłowicza)

**Dane wejściowe:** 0,5 kg ziemniaków, 100 g kiełbasy Chorizo, 8 jajek

**Dane wyjściowe:** gotowa Tortilla

**Opis algorytmu:** Ziemniaki obrać i pokroić w plasterki. Kiełbasę pokroić w plasterki. Ziemniaki wrzucić na gorącą oliwę na patelni i przyrumienić z obu stron. Kiełbasę wrzucić na gorącą oliwę na patelni i przyrumienić z obu stron. Ubić jajka i dodać do połączonych ziemniaków i kiełbasy. Dodać sól i pieprz. Usmażyć z obu stron wielki omelet nadsiewany chipsami ziemniaczanymi z kiełbaską.

## Lista kroków

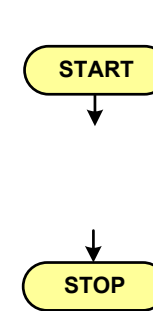
- Uporządkowany opis wszystkich czynności, jakie należy wykonać podczas realizacji algorytmu
- **Krok** jest to pojedyncza czynność realizowana w algorytmie
- Kroki w algorytmie są numerowane, operacje wykonywane są zgodnie z rosnącą numeracją kroków
- Jedynym odstępstwem od powyższej reguły są operacje skoku (warunkowe lub bezwarunkowe), w których jawnie określa się numer kolejnego kroku
- **Przykład** (instrukcja otwierania wózka-specerówki):
  - Krok 1:** Zwolnij element blokujący wózek
  - Krok 2:** Rozkładaj wózek w kierunku kółek
  - Krok 3:** Naciskając nogą dolny element blokujący aż do zatrzaśnięcia, rozłóż wózek do pozycji przewozowej

## Schemat blokowy

- Zawiera plan algorytmu przedstawiony w postaci graficznej
- Na schemacie umieszczane są **bloki** oraz **linie przepływu** (strzałki)
- Blok zawiera informację o wykonywanej operacji
- Linie przepływu (strzałki) określają kolejność wykonywania bloków algorytmu
- Przykład: wyznaczenie roku urodzenia na podstawie wieku (**algorytm liniowy**)

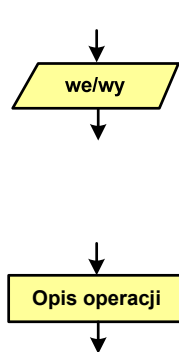


## Schemat blokowy - symbole graficzne



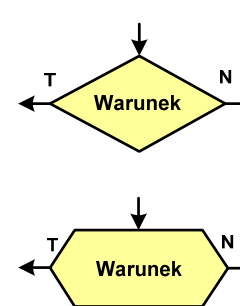
- **blok startowy**, początek algorytmu
- wskazuje miejsce rozpoczęcia algorytmu
- ma jedno wyjście
- może występować tylko jeden raz
- **blok końcowy**, koniec algorytmu
- wskazuje miejsce zakończenia algorytmu
- ma jedno wejście
- musi występować przynajmniej jeden raz

## Schemat blokowy - symbole graficzne

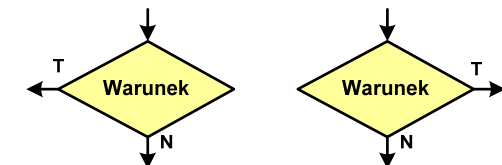


- **blok wejścia-wyjścia**
- poprzez ten blok wprowadzane są (czytane) dane wejściowe i wyprowadzane (zapisywane) wyniki
- ma jedno wejście i jedno wyjście
- **blok wykonawczy**, blok funkcyjny, opis procesu
- zawiera jedno lub kilka poleceń (elementarnych instrukcji) wykonywanych w podanej kolejności
- instrukcją może być np. operacja arytmetyczna, podstawienie
- ma jedno wejście i jedno wyjście

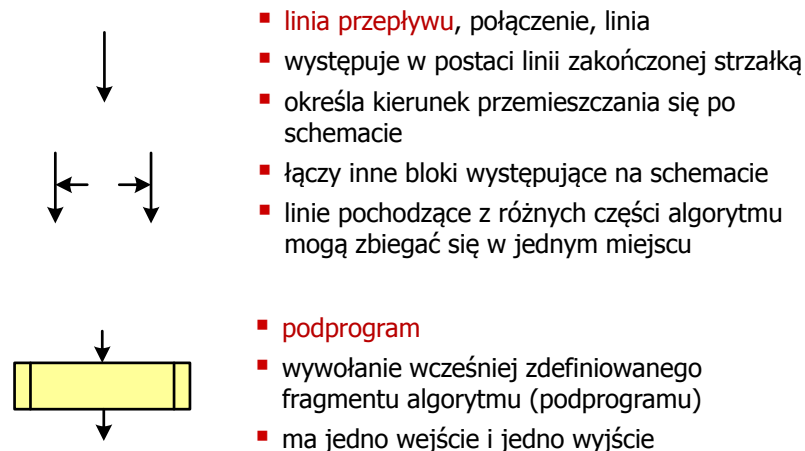
## Schemat blokowy - symbole graficzne



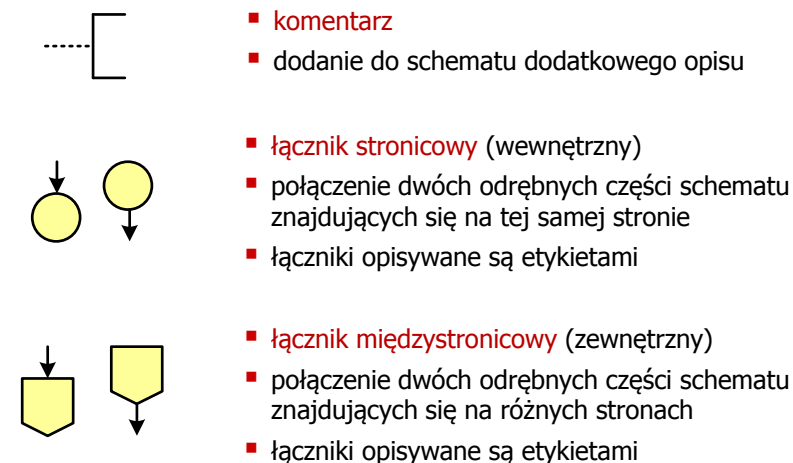
- **blok warunkowy** (decyzyjny, porównujący)
- wewnątrz bloku umieszcza się warunek logiczny
- na podstawie warunku określana jest tylko jedna droga wyjściowa
- połączenia wychodzące z bloku:
  - **T** lub **TAK** - gdy warunek jest prawdziwy
  - **N** lub **NIE** - gdy warunek nie jest prawdziwy
- wyjścia mogą być skierowane na boki lub w dół



## Schemat blokowy - symbole graficzne



## Schemat blokowy - symbole graficzne



## Pseudokod i język programowania

### Pseudokod:

- Pseudokod (pseudojęzyk) - uproszczona wersja języka programowania
- Często zawiera zwroty pochodzące z języków programowania
- Zapis w pseudokodzie może być łatwo przetłumaczony na wybrany język programowania

### Opis w języku programowania:

- Zapis programu w konkretnym języku programowania
- Stosowane języki: Pascal, C, C++, Matlab, Python (kiedyś - Fortran, Basic)

## Największy wspólny dzielnik - algorytm Euklidesa

- NWD - największa liczba naturalna dzieląca (bez reszty) dwie (lub więcej) liczby całkowite

$$\text{NWD}(1675, 3752) = ?$$

Algorytm Euklidesa - przykład

a	b	Dzielenie większej liczby przez mniejszą	Zamiana
1675	3752	$b/a = 3752/1675 = 2$ reszta 402	$b = 402$
1675	402	$a/b = 1675/402 = 4$ reszta 67	$a = 67$
67	402	$b/a = 402/67 = 6$ reszta 0	$b = 0$
67	0	KONIEC	

$$\text{NWD}(1675, 3752) = 67$$

## Algorytm Euklidesa - lista kroków

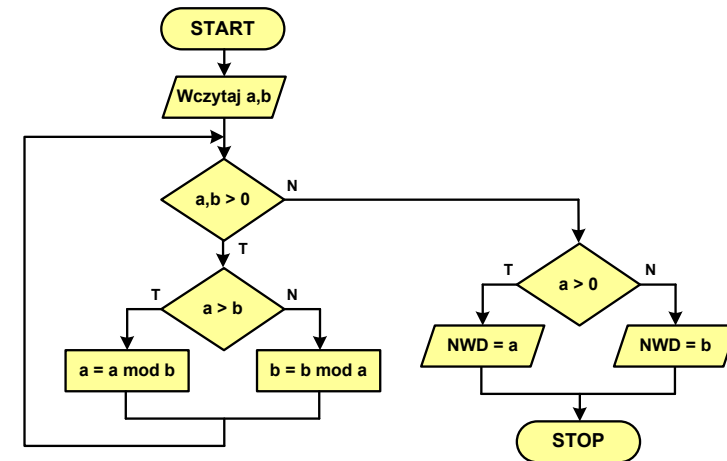
Dane wejściowe: niezerowe liczby naturalne  $a$  i  $b$

Dane wyjściowe:  $NWD(a,b)$

Kolejne kroki:

1. Czytaj liczby  $a$  i  $b$
2. Dopóki  $a$  i  $b$  są większe od zera, powtarzaj **krok 3**, a następnie przejdź do **kroku 4**
3. Jeśli  $a$  jest większe od  $b$ , to weź za  $a$  resztę z dzielenia  $a$  przez  $b$ , w przeciwnym razie weź za  $b$  resztę z dzielenia  $b$  przez  $a$
4. Przyjmij jako największy wspólny dzielnik tę z liczb  $a$  i  $b$ , która pozostała większa od zera
5. Drukuj  $NWD(a,b)$

## Algorytm Euklidesa - schemat blokowy



## Algorytm Euklidesa - pseudokod

```
NWD(a,b)
while a>0 i b>0
do if a>b
    then a ← a mod b
    else b ← b mod a
if a>0
then return a
else return b
```

## Algorytm Euklidesa - język programowania (C)

```
#include <stdio.h>

int main(void)
{
    int a = 1675, b = 3752, NWD;

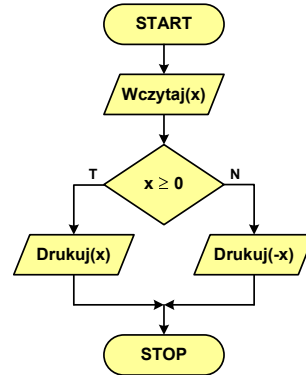
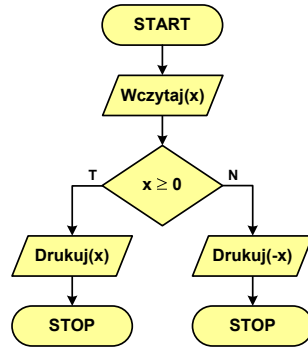
    while (a>0 && b>0)
        if (a>b)
            a = a % b;
        else
            b = b % a;

    if (a>0)
        NWD = a;
    else
        NWD = b;

    printf("NWD = %d\n", NWD);
}
```

## Wartość bezwzględna liczby - schemat blokowy

$$|x| = \begin{cases} x & \text{dla } x \geq 0 \\ -x & \text{dla } x < 0 \end{cases}$$



## Równanie kwadratowe - schemat blokowy

$$ax^2 + bx + c = 0$$

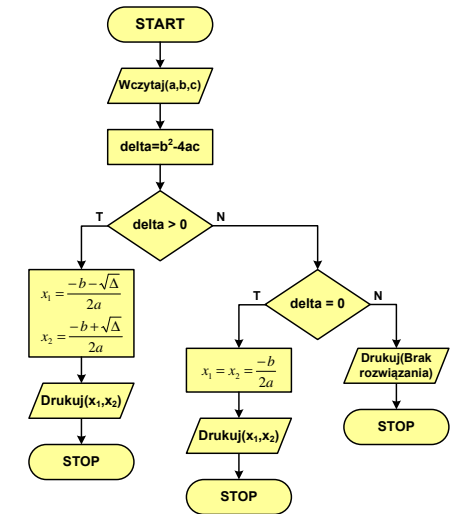
$$\Delta = b^2 - 4ac$$

$$\Delta > 0:$$

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a}, \quad x_2 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$\Delta = 0:$$

$$x_1 = x_2 = \frac{-b}{2a}$$



## Silnia - schemat blokowy

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

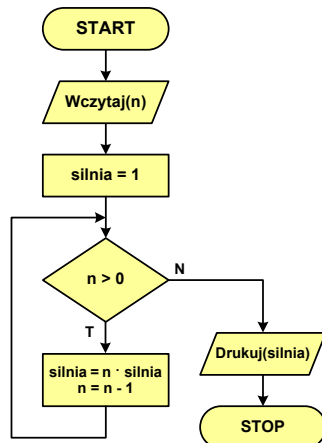
$$0! = 1$$

$$1! = 1$$

$$2! = 1 \cdot 2$$

$$3! = 1 \cdot 2 \cdot 3$$

...



## Rekurencja

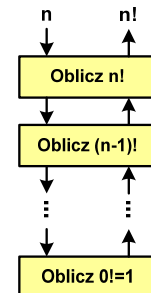
- **Rekurencja** lub **rekursja** - jest to odwoływanie się funkcji lub definicji do samej siebie
- Rozwiązanie danego problemu wyraża się za pomocą rozwiązań tego samego problemu, ale dla danych o mniejszych rozmiarach
- W matematyce mechanizm rekurencji stosowany jest do definiowania lub opisywania algorytmów

- Silnia:

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n(n-1)! & \text{dla } n \geq 1 \end{cases}$$

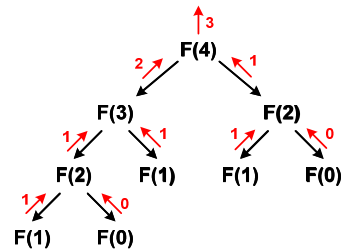
```

int silnia(int n)
{
    return n==0 ? 1 : n*silnia(n-1);
}
    
```



## Rekurencja - ciąg Fibonacciego

$$F_n = \begin{cases} 0 & \text{dla } n=0 \\ 1 & \text{dla } n=1 \\ F_{n-1} + F_{n-2} & \text{dla } n > 1 \end{cases}$$



```
int F(int n)
{
    if (n==0) return 0;
    else
        if (n==1) return 1;
        else
            return F(n-1) + F(n-2);
}
```

## Rekurencja - algorytm Euklidesa

$$NWD(a,b) = \begin{cases} a & \text{dla } b=0 \\ NWD(b, a \bmod b) & \text{dla } b \geq 1 \end{cases}$$

```
int NWD(int a, int b)
{
    if (b==0)
        return a;
    else
        return NWD(b, a % b);
}
```

## Złożoność obliczeniowa

- W celu rozwiązania danego problemu obliczeniowego szukamy algorytmu najbardziej **efektywnego** czyli:
  - najszybszego (najkrótszy czas otrzymania wyniku)
  - o możliwie małym zapotrzebowaniu na pamięć
- **Problem:** Jak ocenić, który z dwóch różnych algorytmów rozwiązujących to samo zadanie jest efektywniejszy?
- Do oceny efektywności służy **złożoność obliczeniowa algorytmu (koszt algorytmu)**
- Złożoność obliczeniowa algorytmu to ilość zasobów potrzebnych do jego działania (czas, pamięć)

## Złożoność obliczeniowa

### Złożoność czasowa

- Czas wykonania algorytmu wyrażony w **liczbie wykonywanych operacji** (jednostkach czasu, liczbie cykli procesora) w zależności od wielkości danych
- Jej miarą jest zazwyczaj liczba podstawowych operacji (dominujących) - pozostałe operacje są pomijane
- Podstawowe operacje: porównanie, podstawienie, operacja arytmetyczna

### Złożoność pamięciowa

- Jest miarą wykorzystania pamięci (liczba komórek pamięci)
- Wyrażana jest w liczbie bajtów lub liczbie zmiennych określonego typu w zależności od wielkości danych

## Złożoność obliczeniowa

- Jeśli wykonanie algorytmu zależy od zestawu danych wejściowych, to wyróżnia się:

### Złożoność optymistyczna

- Odpowiada danym najbardziej sprzyjającym dla algorytmu

### Złożoność średnia (oczekiwana)

- Złożoność uśredniona po wszystkich możliwych zestawach danych, występująca dla „typowych” (losowych) danych wejściowych

### Złożoność pesymistyczna

- Odpowiada danym najbardziej niesprzyjającym dla algorytmu
- Przykład: poszukiwanie określonej wartości w N-elementowej, nieposortowanej tablicy liczb

## Złożoność obliczeniowa

- Złożoność obliczeniowa algorytmu jest **funkcją** opisującą zależność między liczbą danych a liczbą operacji wykonywanych przez ten algorytm
- W praktyce stosuje się oszacowanie powyższej funkcji - są to tzw. notacje (klasy złożoności):
  - $O$  (duże  $O$ )
  - $\Omega$  (omega)
  - $\Theta$  (theta)

## Notacja $O$ („duże $O$ ”)

- Wyraża złożoność matematyczną algorytmu
- Do wyznaczenia złożoności bierze się pod uwagę tylko liczbę dominujących operacji wykonywanych w algorytmie
- W funkcji opisującej złożoność bierze się pod uwagę tylko najistotniejszy składnik, np.

$$f(n) = n^2 + 2n \rightarrow O(n^2) \quad f(n) = n^2 + n - 5 \rightarrow O(n^2)$$

- Po literze  $O$  występuje wyrażenie w nawiasach zawierające literę  $n$ , która oznacza liczbę elementów, na których działa algorytm
- W powyższych przykładach dla dużego  $n$  wpływ składnika liniowego i stałego na wartość funkcji jest nieistotny w porównaniu ze składnikiem głównym  $n^2$

## Notacja $O$ („duże $O$ ”)

- Porównanie najczęściej występujących złożoności:

Elementy (n)	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$
10	3	10	33	100	1 000	1024
100	7	100	664	10 000	1 000 000	$1,27 \cdot 10^{30}$
1 000	10	1 000	9 966	1 000 000	$10^9$	$1,07 \cdot 10^{301}$
10 000	13	10 000	132 877	$10^8$	$10^{12}$	$1,99 \cdot 10^{3010}$

- $O(\log n)$  - logarytmiczna (np. przeszukiwanie binarne)
- $O(n)$  - liniowa (np. porównywanie łańcuchów znaków)
- $O(n \log n)$  - liniowo-logarytmiczna (np. sortowanie szybkie)
- $O(n^2)$  - kwadratowa (np. proste algorytmy sortowania)
- $O(n^3)$  - sześcienna (np. mnożenie macierzy)
- $O(2^n)$  - wykładnicza (np. problem komiwojażera)



## Sortowanie

- Sortowanie polega na uporządkowaniu zbioru danych względem pewnych cech charakterystycznych każdego elementu tego zbioru (wartości każdego elementu)
- W przypadku liczb, sortowanie polega na znalezieniu kolejności liczb zgodnej z relacją  $\leq$  lub  $\geq$

### Przykład:

- Tablica nieposortowana:
 

6	4	5	2	3	1
---	---	---	---	---	---
- Tablica posortowana zgodnie z relacją  $\leq$  (od najmniejszej do największej liczby):
 

1	2	3	4	5	6
---	---	---	---	---	---
- Tablica posortowana zgodnie z relacją  $\geq$  (od największej do najmniejszej liczby):
 

6	5	4	3	2	1
---	---	---	---	---	---

## Sortowanie

- W przypadku słów sortowanie polega na ustawieniu ich w porządku alfabetycznym (leksykograficznym)

### Przykład:

- Tablica nieposortowana:

Paweł	Piotr	Adrian	Ela	Ola	Henryk
-------	-------	--------	-----	-----	--------

- Tablice posortowane:

Adrian	Ela	Henryk	Ola	Paweł	Piotr
--------	-----	--------	-----	-------	-------

Piotr	Paweł	Ola	Henryk	Ela	Adrian
-------	-------	-----	--------	-----	--------

## Sortowanie

- W praktyce sortowanie sprowadza się do porządkowanie danych na podstawie porównania - porównywany element to **klucz**

### Przykład:

- Tablica nieposortowana (imię, nazwisko, wiek):

Piotr	Ola	Paweł	Jan	Ela	Magda
Kowalski	Nowak	Wójcik	Kamiński	Król	Mazur
25	18	23	20	22	15

- Tablica posortowana (klucz - nazwisko):

Jan	Piotr	Ela	Magda	Ola	Paweł
Kamiński	Kowalski	Król	Mazur	Nowak	Wójcik
20	25	22	15	18	23

- Tablica posortowana (klucz - wiek):

Magda	Ola	Jan	Ela	Paweł	Piotr
Mazur	Nowak	Kamiński	Król	Wójcik	Kowalski
15	18	20	22	23	25

## Sortowanie

### Po co stosować sortowanie?

- Posortowane elementy można szybciej zlokalizować
- Posortowane elementy można przedstawić w czytelniejszy sposób

### Klasyfikacje algorytmów sortowania

- Złożoność obliczeniowa algorytmu** - zależność liczby wykonywanych operacji od liczebności sortowanego zbioru  $n$
- Złożoność pamięciowa** - wielkość zasobów zajmowanych przez algorytm (sortowanie **w miejscu** - wielkość zbioru danych podczas sortowania nie zmienia się lub jest tylko nieco większa)
- Sortowanie wewnętrzne** (odbywa się w pamięci komputera) i **zewnętrzne** (nie jest możliwe jednoczesne umieszczenie wszystkich elementów zbioru sortowanego w pamięci komputera)

## Klasyfikacje algorytmów sortowania

- Algorytm jest **stabilny**, jeśli podczas sortowania zachowuje kolejność występowania elementów o tym samym kluczu

### Przykład:

- Tablica nieposortowana (imię, nazwisko, wiek):

Piotr	Ola	Paweł	Jan	Ela	Magda
Kowalski	Nowak	Wójcik	Kamiński	Król	Mazur
20	18	23	20	18	15

- Tablica posortowana algorytmem stabilnym (klucz - wiek):

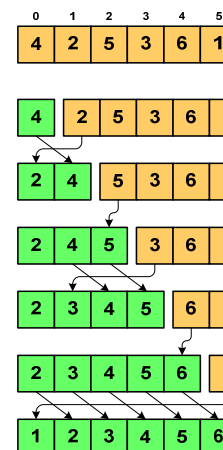
Magda	Ola	Ela	Piotr	Jan	Paweł
Mazur	Nowak	Król	Kowalski	Kamiński	Wójcik
15	18	18	20	20	23

- Tablica posortowana algorytmem niestabilnym (klucz - wiek):

Magda	Ela	Ola	Jan	Piotr	Paweł
Mazur	Król	Nowak	Kamiński	Kowalski	Wójcik
15	18	18	20	20	23

## Proste wstawianie (insertion sort)

### Przykład:



### Program w języku C:

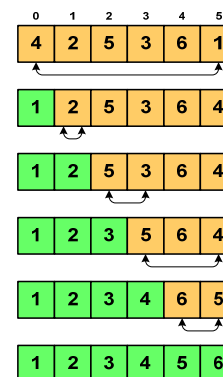
```
int main(void)
{
    int tab[N], i, j, tmp;
    // ...
    for (i=1; i<N; i++)
    {
        j=i;
        tmp=tab[i];
        while (tab[j-1]>tmp && j>0)
        {
            tab[j]=tab[j-1];
            j--;
        }
        tab[j]=tmp;
    }
}
```

## Proste wstawianie (insertion sort)

- Złożoność algorytmu:  $O(n^2)$ 
  - + wydajny dla danych wstępnie posortowanych
  - + wydajny dla zbiorów o niewielkiej liczebności
  - + małe zasoby zajmowane podczas pracy (sortowanie w miejscu)
  - + stabilny
  - + prosty w implementacji
- mała efektywność dla normalnej i dużej ilości danych.

## Proste wybieranie (selection sort)

### Przykład:



### Program w języku C:

```
int main(void)
{
    int tab[N], i, j, k, tmp;
    // ...
    for (i=0; i<N-1; i++)
    {
        k=i;
        for (j=i+1; j<N; j++)
            if (tab[k]>=tab[j])
                k = j;
        tmp = tab[i];
        tab[i] = tab[k];
        tab[k] = tmp;
    }
}
```

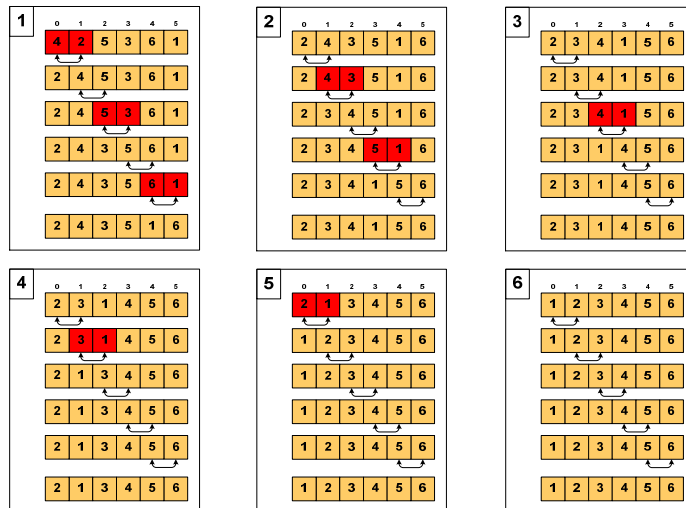
## Proste wybieranie (selection sort)

- Złożoność algorytmu:  $O(n^2)$ 
  - + szybki w sortowaniu niewielkich tablic
  - + małe zasoby zajmowane podczas pracy (sortowanie w miejscu)
  - + prosty w implementacji
- liczba porównań elementów jest niezależna od początkowego rozmieszczenia elementów w tablicy
- w algorytmie może zdarzyć się, że wykonywana jest zamiana tego samego elementu ze sobą.

## Bąbelkowe (bubble sort)

- **Sortowanie bąbelkowe** (ang. bubble sort), nazywane jest także:
  - sortowaniem pęcherzykowym
  - sortowaniem przez prostą zamianę (ang. straight exchange)
- Metoda ta polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności jeśli jest to konieczne
- Nazwa metody wzięła się stąd, że kolejne porównania powodują „wypychanie” kolejnego największego elementu na koniec („wypłynięcie największego bąbelka”)

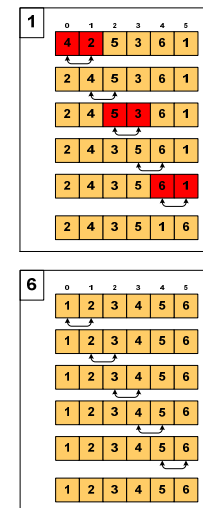
## Bąbelkowe (bubble sort)



## Bąbelkowe (bubble sort)

### Program w języku C:

```
int main(void)
{
    int tab[N], i, j, tmp, koniec;
    // ...
    do {
        koniec=1;
        for (i=0; i<N-1; i++)
            if (tab[i]>tab[i+1])
            {
                tmp=tab[i];
                tab[i]=tab[i+1];
                tab[i+1]=tmp;
                koniec=0;
            }
    } while (!koniec);
}
```



## Bąbelkowe (bubble sort)

- Złożoność algorytmu:  $O(n^2)$ 
  - + prosta realizacja
  - + wysoka efektywność użycia pamięci (sortowanie w miejscu)
  - + stabilny
- mała efektywność.

## Sortowanie szybkie (Quick-Sort) - faza dzielenia

- Tablica jest dzielona na dwie części wokół pewnego elementu  $x$  (nazywanego elementem centralnym)
- Jako element centralny  $x$  najczęściej wybierany jest element środkowy (choć może to być także element losowy)
- Przeglądamy tablicę od lewej strony, aż znajdziemy element  $a_i \geq x$ , a następnie przeglądamy tablicę od prawej strony, aż znajdziemy element  $a_j \leq x$
- Zamieniamy elementy  $a_i$  i  $a_j$  miejscami i kontynuujemy proces przeglądania i zamiany, aż nastąpi spotkanie w środku tablicy
- W ten sposób otrzymujemy tablicę podzieloną na lewą część z wartościami mniejszymi lub równymi  $x$  i na prawą część z wartościami większymi lub równymi  $x$

## Sortowanie szybkie (Quick-Sort) - faza sortowania

- Zawiera dwa rekurencyjne wywołania tej samej funkcji sortowania: dla lewej i dla prawej części posortowanej tablicy
- Rekurencja zatrzymuje się, gdy wielkość tablicy wynosi 1

### Przykład:

- Sortujemy 6-elementową tablicę  $tab$ :

	0	1	2	3	4	5
tab	4	2	5	3	6	1

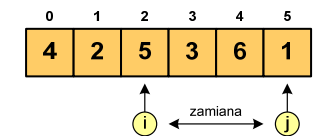
- Wywołanie funkcji  $QS()$  ma postać:

$QS(tab, 0, 5);$

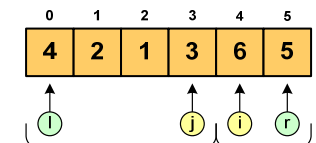
## Sortowanie szybkie (Quick-Sort) - $QS(tab, 0, 5)$

- Element środkowy:  $(0+5)/2 = 2, \quad x = tab[2] = 5$

- Od lewej szukamy  $tab[i] \geq x$ , a od prawej szukamy  $tab[j] \leq x$ , zamieniamy elementy miejscami



- Poszukiwania kończymy, gdy indeksy  $i, j$  mijają się



- Wywołujemy rekurencyjnie funkcję  $QS()$  dla elementów z zakresów  $[l, j]$  i  $[i, r]$ :

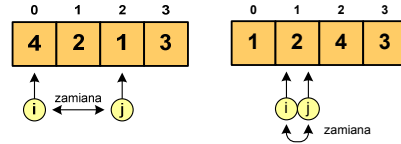
$QS(tab, 0, 3);$

$QS(tab, 4, 5);$

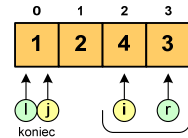
## Sortowanie szybkie (Quick-Sort) - QS(tab,0,3)

- Element środkowy:  $(0+3)/2 = 1$ ,  $x = \text{tab}[1] = 2$

- Od lewej szukamy  $\text{tab}[i] \geq x$ , a od prawej szukamy  $\text{tab}[j] \leq x$ , zamieniamy elementy miejscami



- Poszukiwania kończymy, gdy indeksy  $i, j$  mijają się



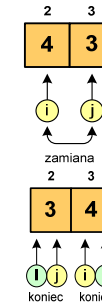
- Wywołanie QS() tylko dla elementów z zakresu [2,3], gdyż po lewej stronie rozmiar tablicy do posortowania wynosi 1:

**QS (tab, 2, 3) ;**

## Sortowanie szybkie (Quick-Sort) - QS(tab,2,3)

- Element środkowy:  $(2+3)/2 = 2$ ,  $x = \text{tab}[2] = 4$

- Od lewej szukamy  $\text{tab}[i] \geq x$ , a od prawej szukamy  $\text{tab}[j] \leq x$ , zamieniamy elementy miejscami



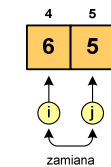
- Poszukiwania kończymy, gdy indeksy  $i, j$  mijają się

- Rozmiar obu tablic do posortowania wynosi 1 więc nie ma nowych wywołań funkcji QS()

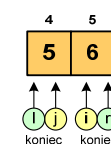
## Sortowanie szybkie (Quick-Sort) - QS(tab,4,5)

- Element środkowy:  $(4+5)/2 = 4$ ,  $x = \text{tab}[4] = 6$

- Od lewej szukamy  $\text{tab}[i] \geq x$ , a od prawej szukamy  $\text{tab}[j] \leq x$ , zamieniamy elementy miejscami



- Poszukiwania kończymy, gdy indeksy  $i, j$  mijają się



- Rozmiar obu tablic do posortowania wynosi 1 więc nie ma nowych wywołań funkcji QS()

## Sortowanie szybkie (Quick-Sort)

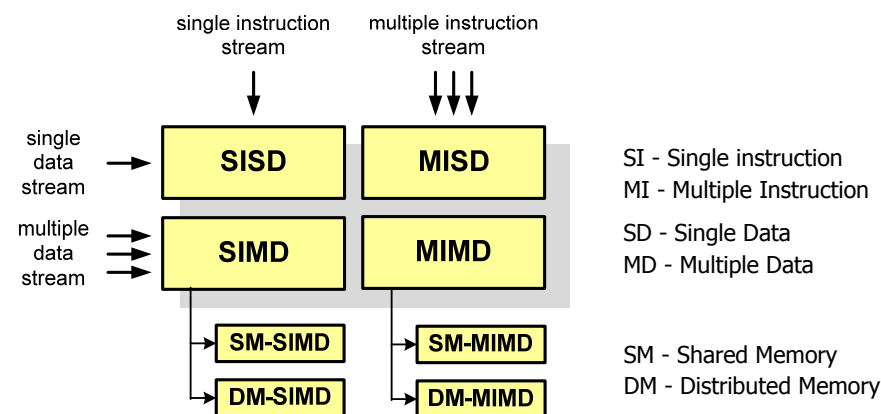
### Funkcja w języku C:

```
void QuickSort(int tab[], int l, int r)
{
    int i, j, x, y;
    i=l;
    j=r;
    x=tab[(l+r)/2];
    do
    {
        while (tab[i]<x) i++;
        while (x<tab[j]) j--;
        if (i<=j)
        {
            y=tab[i];
            tab[i]=tab[j];
            tab[j]=y;
            i++; j--;
        }
    } while (i<=j);
    if (l<j) QuickSort(tab,l,j);
    if (i<r) QuickSort(tab,i,r);
}
```

## Klasyfikacja systemów komputerowych

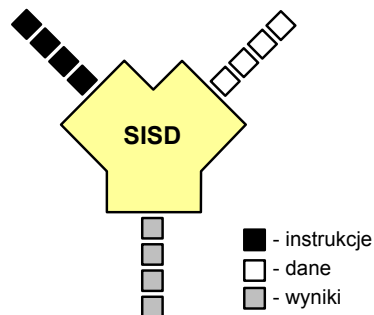
- **Taksonomia Flynna** - pierwsza, najbardziej ogólna klasyfikacja architektur komputerowych (1972):
  - Flynn M.J.: „Some Computer Organizations and Their Effectiveness”, IEEE Transactions on Computers, Vol. C-21, No 9, 1972.
- Opiera się na liczbie przetwarzanych strumieni rozkazów i strumieni danych:
  - **strumień rozkazów** (Instruction Stream) - odpowiednik licznika rozkazów; system złożony z  $n$  procesorów posiada  $n$  liczników rozkazów, a więc  $n$  strumieni rozkazów
  - **strumień danych** (Data Stream) - zbiór operandów, np. system rejestrujący temperaturę mierzoną przez  $n$  czujników posiada  $n$  strumieni danych

## Taksonomia Flynna



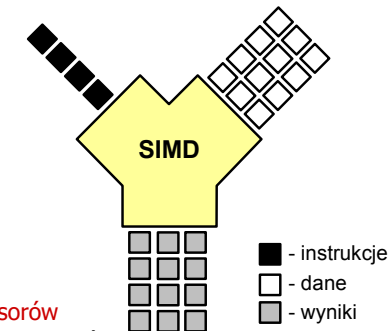
## SISD (Single Instruction, Single Data)

- Jeden wykonywany program przetwarza jeden strumień danych
- Klasyczne komputery zbudowane według **architektury von Neumanna**
- Zawierają:
  - jeden procesor
  - jeden blok pamięci operacyjnej zawierający wykonywany program.



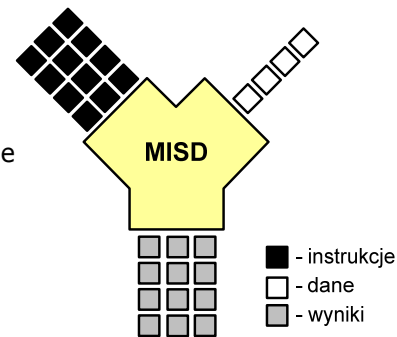
## SIMD (Single Instruction, Multiple Data)

- Jeden wykonywany program przetwarza wiele strumieni danych
- Te same operacje wykonywane są na różnych danych
- Podział:
  - SM-SIMD (Shared Memory SIMD):
    - komputery wektorowe
    - rozszerzenia strumieniowe procesorów (MMX, 3DNow!, SSE, SSE2, SSE3, AVX, ...)
  - DM-SIMD (Distributed Memory SIMD):
    - tablice procesorów
    - procesory kart graficznych (GPGPU)



## MISD (Multiple Instruction, Single Data)

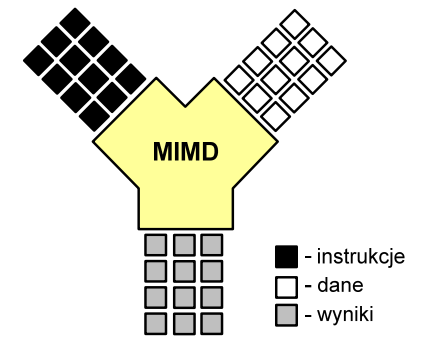
- Wiele równoległe wykonywanych programów przetwarza jednocześnie jeden wspólny strumień danych
- Systemy tego typu nie są spotykane



## MIMD (Multiple Instruction, Multiple Data)

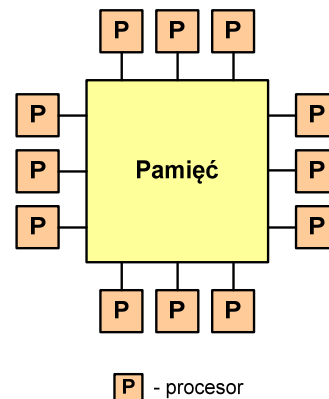
- Równoległe wykonywanych jest wiele programów, z których każdy przetwarza własne strumienie danych

- Podział:
  - SM-MIMD (Shared Memory):
    - wieloprocesory
  - DM-MIMD (Distributed Memory):
    - wielokomputery
    - klastry
    - gridy



## SM-MIMD - Wieloprocesory

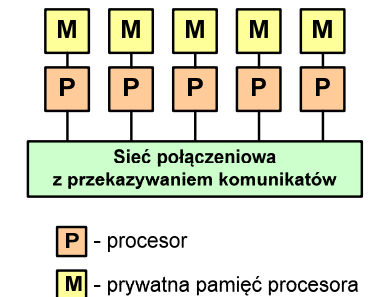
- Systemy z niezbyt dużą liczbą działających niezależnie procesorów
- Każdy procesor ma dostęp do wspólnej przestrzeni adresowej pamięci
- Komunikacja procesorów poprzez uzgodniony obszar wspólnej pamięci
- Do SM-MIMD należą komputery z **procesorami wielordzeniowymi**



## DM-MIMD - Wielokomputery

- Każdy procesor wyposażony jest we własną pamięć operacyjną, niedostępną dla innych procesorów
- Komunikacja między procesorami odbywa się za pomocą sieci poprzez przesyłanie komunikatów

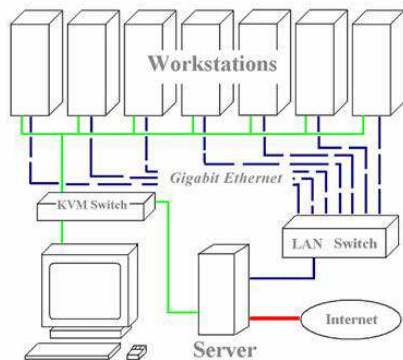
- Biblioteki komunikacyjne:
  - MPI (Message Passing Interface)
  - PVM (Parallel Virtual Machine)



## DM-MIMD - Klastry

- **Klaster (cluster):**
  - równoległy lub rozproszony system składający się z komputerów
  - komputery połączone są siecią
  - używany jest jako pojedynczy, zintegrowany zespół obliczeniowy

- **Węzeł (node)** - pojedynczy komputer przyłączony do klastra i wykonujący zadania obliczeniowe



KVM - Keyboard, Video, Mouse

źródło:  
[http://leda.elfak.ni.ac.rs/projects/SeeGrid/see\\_grid.htm](http://leda.elfak.ni.ac.rs/projects/SeeGrid/see_grid.htm)

## Koniec wykładu nr 6

Dziękuję za uwagę!