



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



Wydział Elektryczny  
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Materiały do wykładu z przedmiotu:  
**Informatyka**  
**Kod: EDS1B1007**

**WYKŁAD NR 4**

**Opracował: dr inż. Jarosław Forenc**

**Białystok 2023**

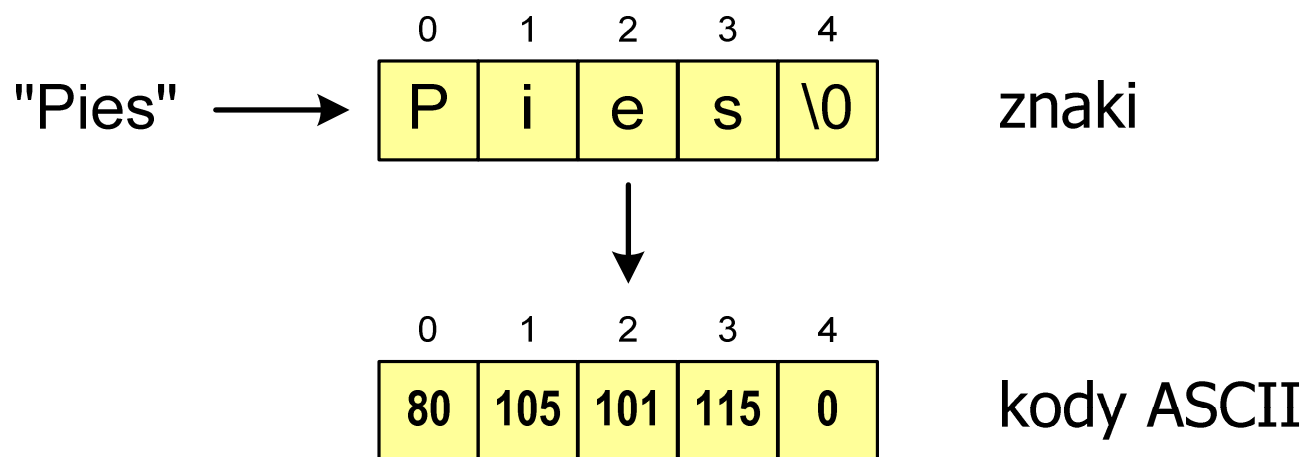
Materiały zostały opracowane w ramach projektu „PB2020 - Zintegrowany Program Rozwoju Politechniki Białostockiej” realizowanego w ramach Działania 3.5 Programu Operacyjnego Wiedza, Edukacja, Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

## Plan wykładu nr 4

- Łańcuchy znaków
  - wczytanie tekstu i znaku
  - plik nagłówkowy string.h
  - macierz elementów typu char
  
- Algorytmy komputerowe
  - definicje, sposoby opisu
  - rekurencja
  - złożoność obliczeniowa
  - algorytmy sortowania (proste wstawianie, proste wybieranie, bąbelkowe, szybkie - quick-sort)

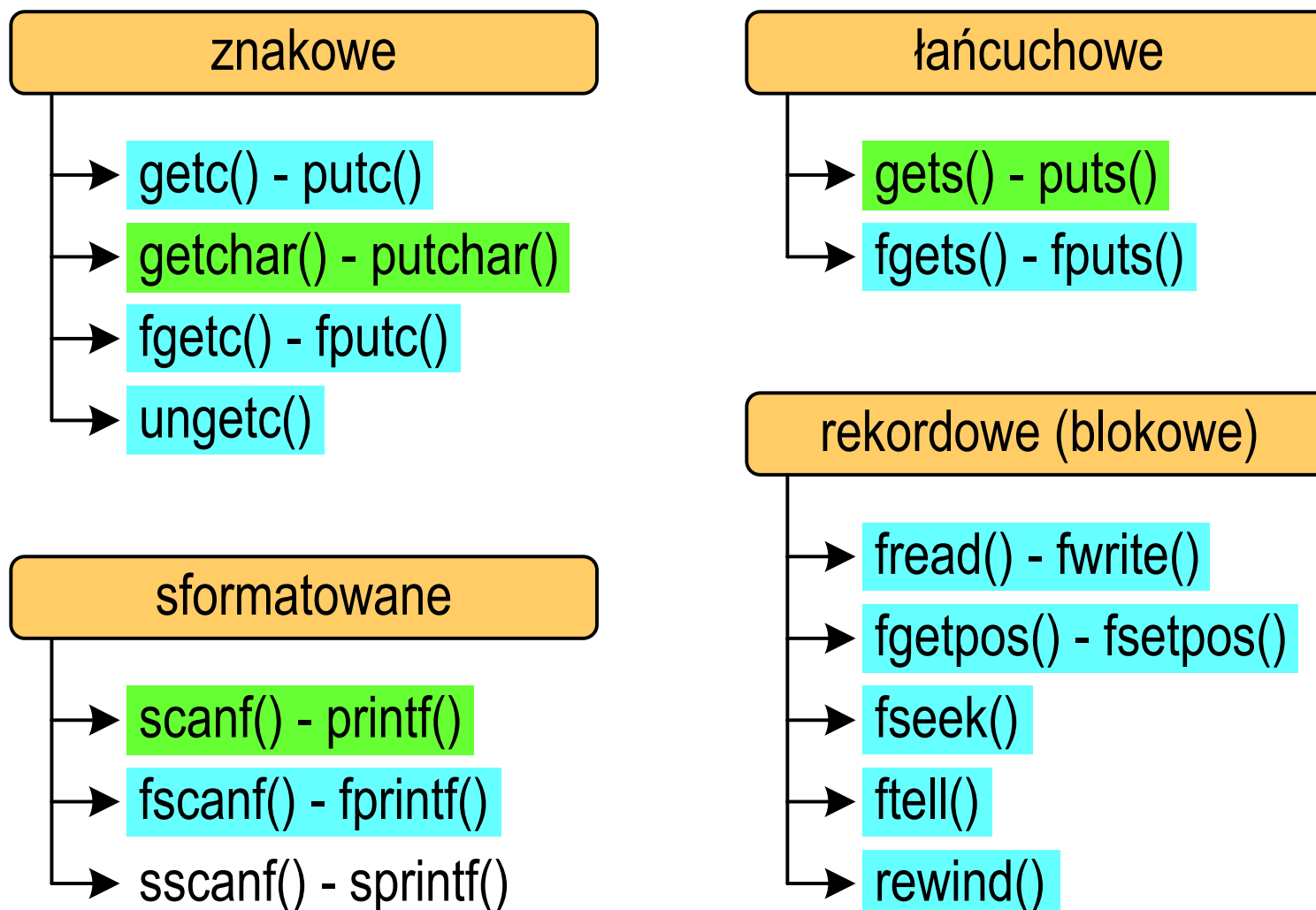
## Język C - łańcuchy znaków

- łańcuch znaków to tablica elementów typu **char** przechowująca kody ASCII znaków (czyli liczby)



```
char txt1[5] = "Pies";  
char txt2[5] = {'P', 'i', 'e', 's'};  
char txt3[5] = {80, 105, 101, 115};
```

# Język C - standardowe funkcje wejścia-wyjścia



## Język C - wczytanie tekstu

- Do wczytania tekstu funkcją `scanf()` stosowany jest specyfikator `%s`

```
char napis[15];  
scanf("%s", napis);
```

brak znaku `&`

- W specyfikatorze formatu `%s` można podać szerokość

```
char napis[15];  
scanf("%10s", napis);
```

- W powyższym przykładzie `scanf()` zakończy wczytywanie tekstu po pierwszym białym znaku (spacja, tabulacja, enter) lub w momencie pobrania 10 znaków

## Język C - wczytanie tekstu

- W przypadku wprowadzenia tekstu "To jest napis", funkcja `scanf()` zapamięta tylko wyraz "To"
- Zapamiętanie całego wiersza tekstu (do naciśnięcia klawisza `Enter`) wymaga użycia funkcji `gets()`

`gets()`

`char *gets(char *s);`

- Funkcja `gets()` wprowadza wiersz (ciąg znaków zakończony `'\n'`) ze strumienia `stdin` (klawiatura) i umieszcza w obszarze pamięci wskazywanym przez wskaźnik `s` zastępując `'\n'` znakiem `'\0'`

```
char napis[15];  
gets(napis);
```

## Język C - wczytanie znaku

- Wczytanie jednego znaku funkcją `scanf()` wymaga specyfikatora formatu `%c` (przed zmienną `znak` musi wystąpić operator `&`)

```
int znak;  
scanf ("%c", &znak);
```

- Do wczytania znaku można zastosować także funkcję `getchar()`

<code>getchar()</code>	<code>int getchar(void);</code>
------------------------	---------------------------------

```
int znak;  
znak = getchar();
```

## Język C - plik nagłówkowy string.h

`strcpy()`

`char *strcpy(char *s1, const char *s2);`

- Kopiuje łańcuch `s2` do łańcucha `s1`

`strlen()`

`size_t strlen(const char *s);`

- Zwraca długość łańcucha znaków, nie uwzględnia znaku `'\0'`

`strcat()`

`char *strcat(char *s1, const char *s2);`

- Dołącza do łańcucha `s1` łańcuch `s2`



## Język C - plik nagłówkowy string.h

`strcmp()`

```
int strcmp(const char *s1, const char *s2);
```

- Porównuje łańcuchy `s1` i `s2` z rozróżnianiem wielkości liter

`strcmpi()`

```
int strcmpi(const char *s1, const char *s2);
```

- Porównuje łańcuchy `s1` i `s2` bez rozróżniania wielkości liter

`strchr()`

```
char *strchr(const char *s, int c);
```

- Szuka w łańcuchu `s` znaku `c`

## Język C - plik nagłówkowy string.h

`strlwr()`

`char *strlwr(char *s);`

- Zamienia w łańcuchu **s** wielkie litery na małe

`strupr()`

`char *strupr(char *s);`

- Zamienia w łańcuchu **s** małe litery na wielkie

`strrev()`

`char *strrev(char *s);`

- Odwraca kolejność znaków w łańcuchu **s**

## Język C - plik nagłówkowy string.h (przykład)

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char napis1[] = "Tekst w buforze", napis2[20];

    printf("napis1: %s \n", napis1);
    int dlugosc = strlen(napis1);
    printf("liczba znakow w napis1: %d \n", dlugosc);
    strcpy(napis2, napis1);
    printf("napis2: %s \n", napis2);
    strrev(napis2);
    printf("napis2 (odwr): %s \n", napis2);

    return 0;
}
```

## Język C - plik nagłówkowy string.h (przykład)

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char napis1[] = "Tekst w buforze";

    printf("napis1: %s \n", napis1);
    int dlugosc = strlen(napis1);
    printf("liczba znakow w napis1: %d \n", dlugosc);
    strcpy(napis2, napis1);
    printf("napis2: %s \n", napis2);
    strrev(napis2);
    printf("napis2 (odwr): %s \n", napis2);

    return 0;
}
```

```
napis1: Tekst w buforze
liczba znakow w napis1: 15
napis2: Tekst w buforze
napis2 (odwr): ezrofub w tskeT
```







# Algorytm - definicje

## Definicja 1

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

## Definicja 2

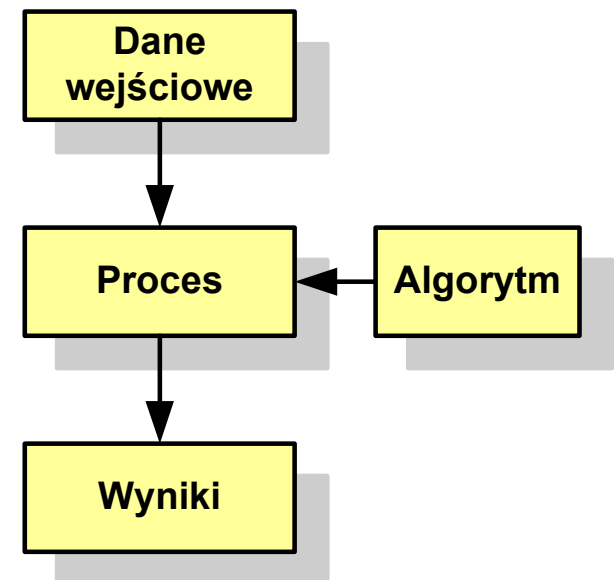
- Opis rozwiązania problemu wyrażony za pomocą operacji zrozumiałych i możliwych do zrealizowania przez wykonawcę

## Definicja 3

- Ściśle 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 matematyka perskiego z IX wieku - Muhammada ibn-Musy **al-Chuwarizmiego** (po łacinie pisanego jako **Algorismus**)
- Badaniem algorytmów zajmuje się **algorytmika**
- „Przetłumaczenie” algorytmu na wybrany język programowania:
  - **implementacja** algorytmu
  - **kodowanie** algorytmu
- Sposoby opisu algorytmów
  - opis słowny w języku naturalnym lub lista kroków (opis w punktach)
  - schemat blokowy
  - pseudokod (nieformalna odmiana języka programowania)
  - 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

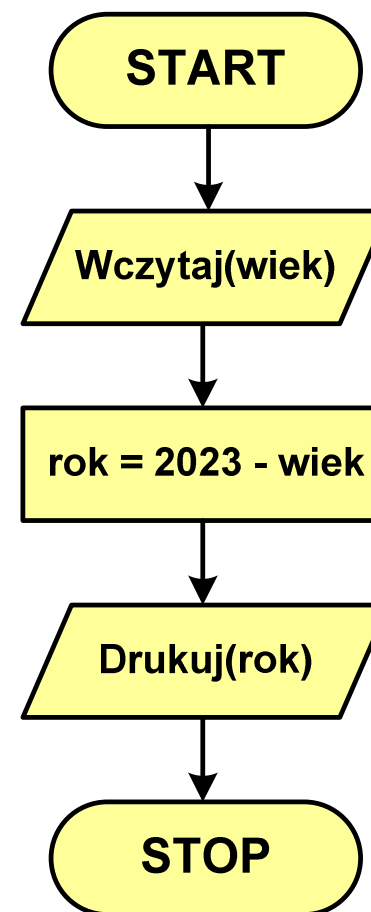
**Opis algorytmu:** Ziemniaki obrać i pokroić w plasterki. Kiełbasę pokroić w plasterki. Ziemniaki wrzucić na gorącą oliwę na patelni i przyrumienić z obu stron. Kiełbasę wrzucić na gorącą oliwę na patelni i przyrumienić z obu stron. Ubić jajka i dodać do połączonych ziemniaków i kiełbasy. Dodać sól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

## Lista kroków

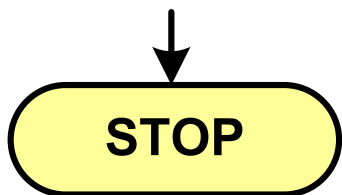
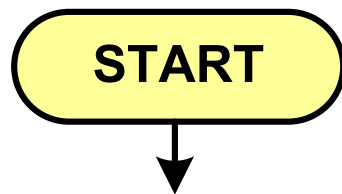
- Uporządkowany opis wszystkich czynności, jakie należy wykonać podczas realizacji algorytmu
- **Krok** jest to pojedyncza czynność realizowana w algorytmie
- Kroki w algorytmie są numerowane, operacje wykonywane są zgodnie z rosnącą numeracją kroków
- Jedynym odstępstwem od powyższej reguły są operacje 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: wyznaczanie roku urodzenia na podstawie wieku (**algorytm liniowy**)

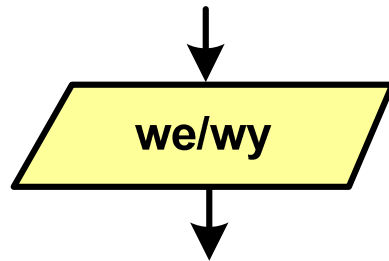


## Schemat blokowy - symbole graficzne

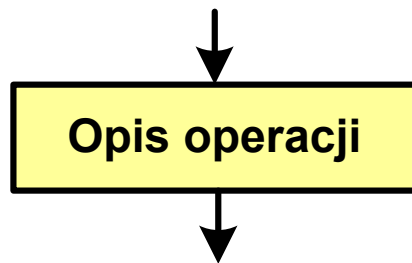


- **blok startowy**, początek algorytmu
  - wskazuje miejsce rozpoczęcia algorytmu
  - ma jedno wyjście
  - może występować tylko jeden raz
- 
- **blok końcowy**, koniec algorytmu
  - wskazuje miejsce zakończenia algorytmu
  - ma jedno wejście
  - musi występować przynajmniej jeden raz

## Schemat blokowy - symbole graficzne

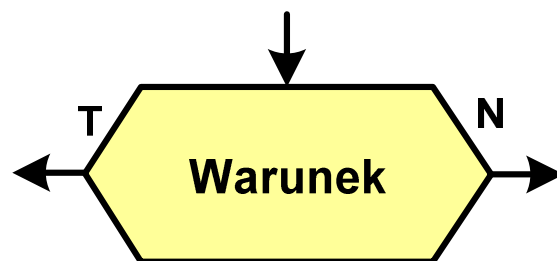
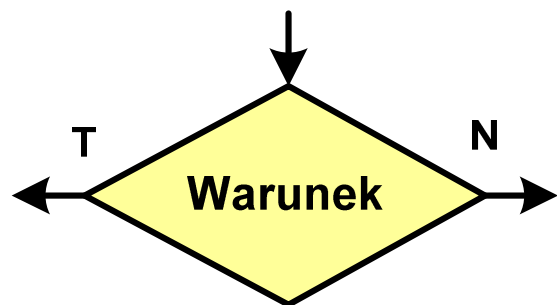


- **blok wejścia-wyjścia**
- poprzez ten blok wprowadzane są (czytane) dane wejściowe i wyprowadzane (zapisywane) wyniki
- ma jedno wejście i jedno wyjście

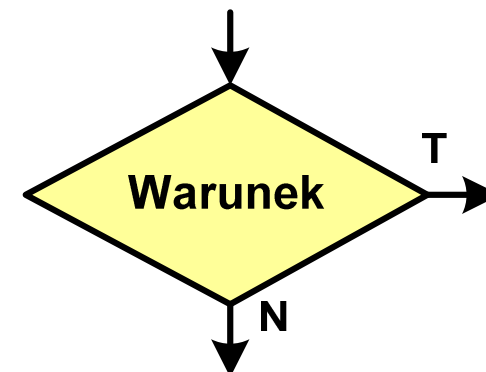
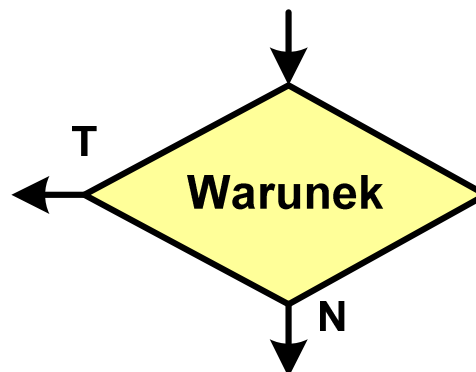


- **blok wykonawczy**, blok funkcyjny, opis procesu
- zawiera jedno lub kilka poleceń (elementarnych instrukcji) wykonywanych w podanej kolejności
- instrukcją może być np. operacja arytmetyczna, podstawienie
- ma jedno wejście i jedno wyjście

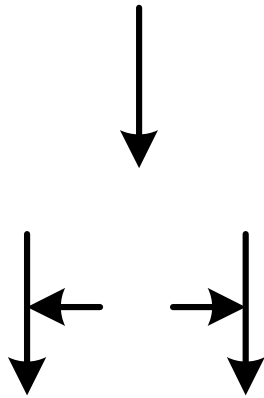
## Schemat blokowy - symbole graficzne



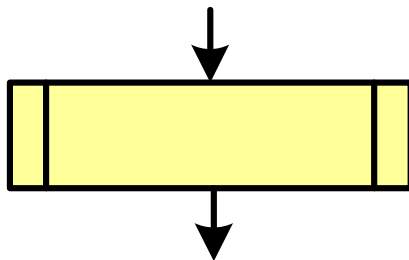
- **blok warunkowy** (decyzyjny, porównujący)
- wewnątrz bloku umieszcza się warunek logiczny
- na podstawie warunku określana jest tylko jedna droga wyjściowa
- połączenia wychodzące z bloku:
  - **T** lub **TAK** - gdy warunek jest prawdziwy
  - **N** lub **NIE** - gdy warunek nie jest prawdziwy
-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



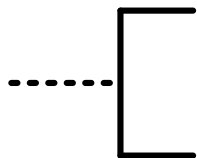
## Schemat blokowy - symbole graficzne



- **linia przepływu**, połączenie, linia
- występuje w postaci linii zakończonej strzałką
- określa kierunek przemieszczania się po schemacie
- linie pochodzące z różnych części algorytmu mogą zbiegać się w jednym miejscu



- **podprogram**
- wywołanie wcześniej zdefiniowanego fragmentu algorytmu (podprogramu)
- ma jedno wejście i jedno wyjście



- **komentarz**
- dodanie do schematu dodatkowego opisu



# 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: C, C++, Matlab, Python  
(kiedyś - Pascal, 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

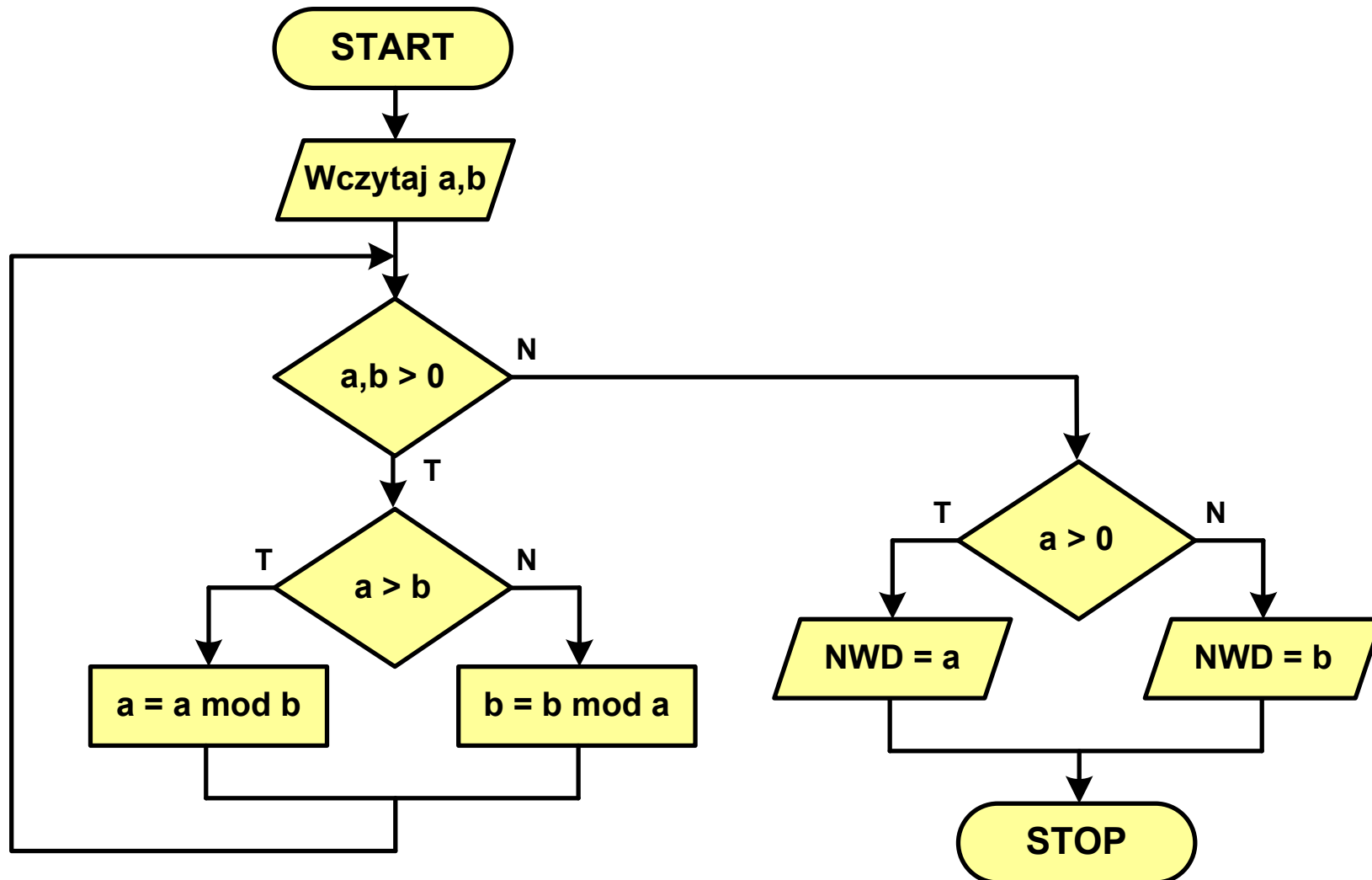
Dane wejściowe: niezerowe liczby naturalne  $a$  i  $b$

Dane wyjściowe:  $\text{NWD}(a,b)$

Kolejne kroki:

1. Czytaj liczby  $a$  i  $b$
2. Dopóki  $a$  i  $b$  są większe od zera, powtarzaj **krok 3**, a w przeciwnym przypadku przejdź do **kroku 4**
3. Jeśli  $a$  jest większe od  $b$ , to weź za  $a$  resztę z dzielenia  $a$  przez  $b$ , w przeciwnym przypadku 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  $\text{NWD}(a,b)$

# 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;

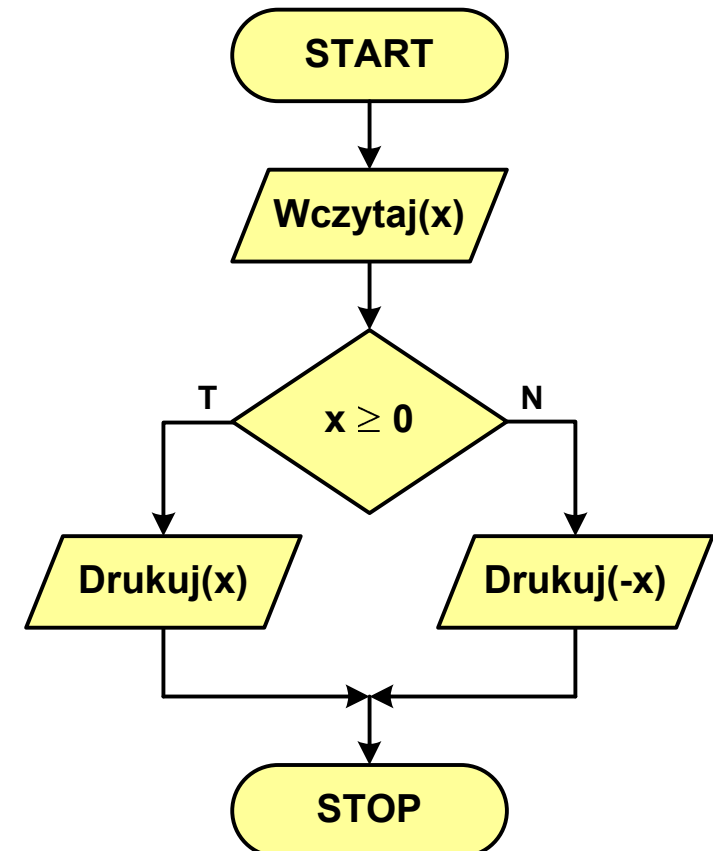
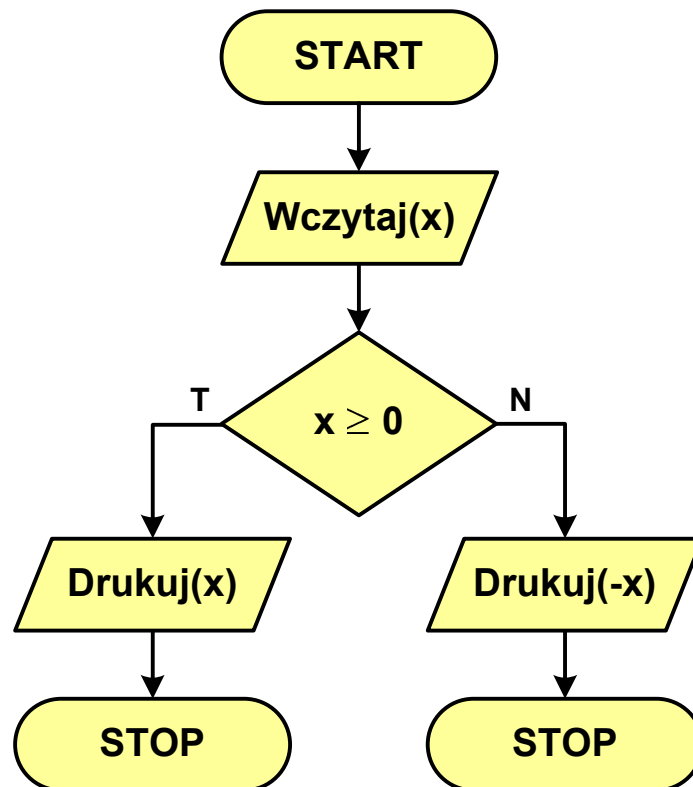
    while (a>0 && b>0)
        if (a>b)
            a = a % b;
        else
            b = b % a;

    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

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

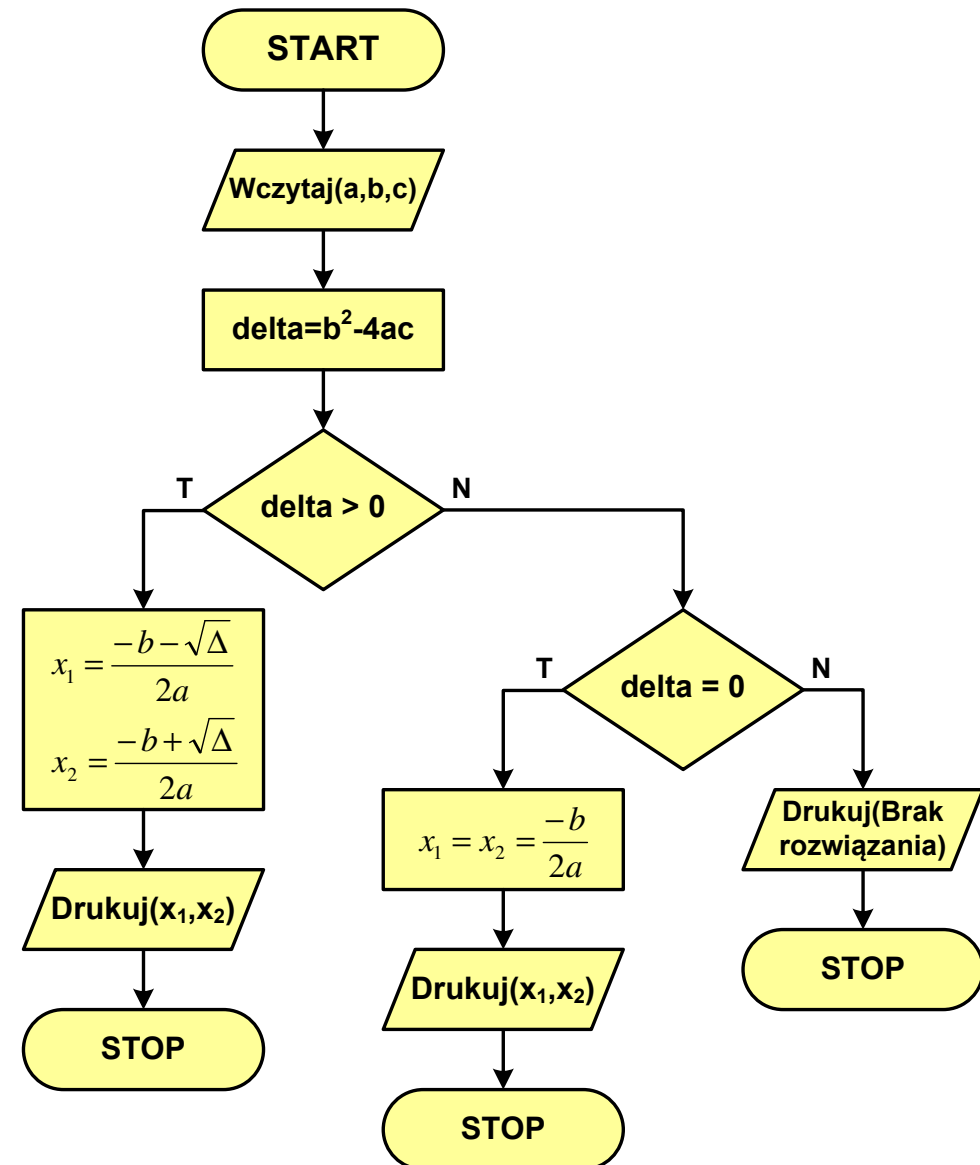
$$\Delta = b^2 - 4ac$$

$\Delta > 0$ :

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

$\Delta = 0$ :

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





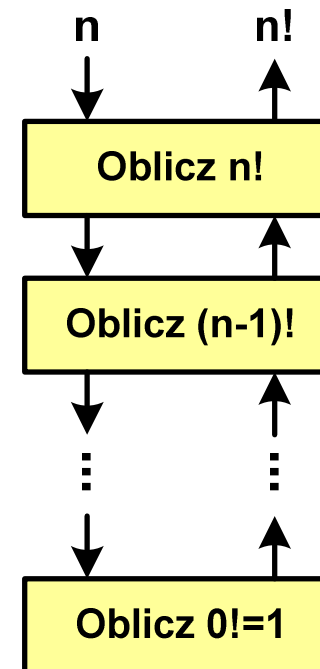
# Rekurencja

- **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:

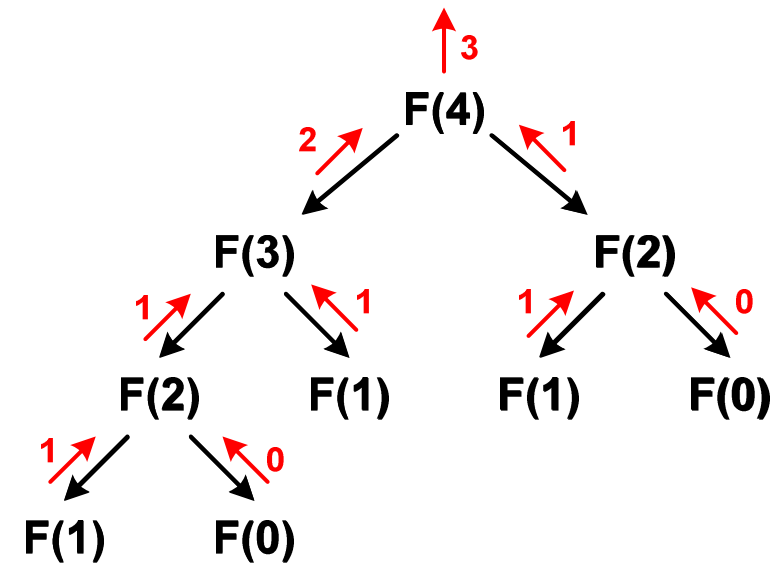
$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n(n-1)! & \text{dla } n \geq 1 \end{cases}$$

```
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;
    if (n==1) return 1;
    return F(n-1) + F(n-2);
}
```

# 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**) czyli ilość zasobów potrzebnych do jego działania (czas, pamięć)
- Miarą złożoności **czasowej** jest liczba podstawowych (dominujących) operacji (porównanie, podstawienie, operacja arytmetyczna) - pozostałe operacje są pomijane
- Miarą złożoności **pamięciowej** jest liczba wykorzystanych komórek pamięci (bajty lub liczba zmiennych określonego typu)

# 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) - najbardziej popularna
  - $\Omega$  (omega)
  - $\Theta$  (theta)

## Notacja $O$ („duże $O$ “)

- Wyraża złożoność matematyczną algorytmu
- Do wyznaczenia złożoności bierze się pod uwagę liczbę dominujących operacji wykonywanych w algorytmie
- Przykład zapisu:  $O(n^2)$ 
  - po literze  $O$  występuje wyrażenie w nawiasach zawierające literę  $n$ , która oznacza liczbę elementów, na których działa algorytm
- W funkcji opisującej złożoność bierze się pod uwagę tylko najistotniejszy składnik, np.

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

- 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):

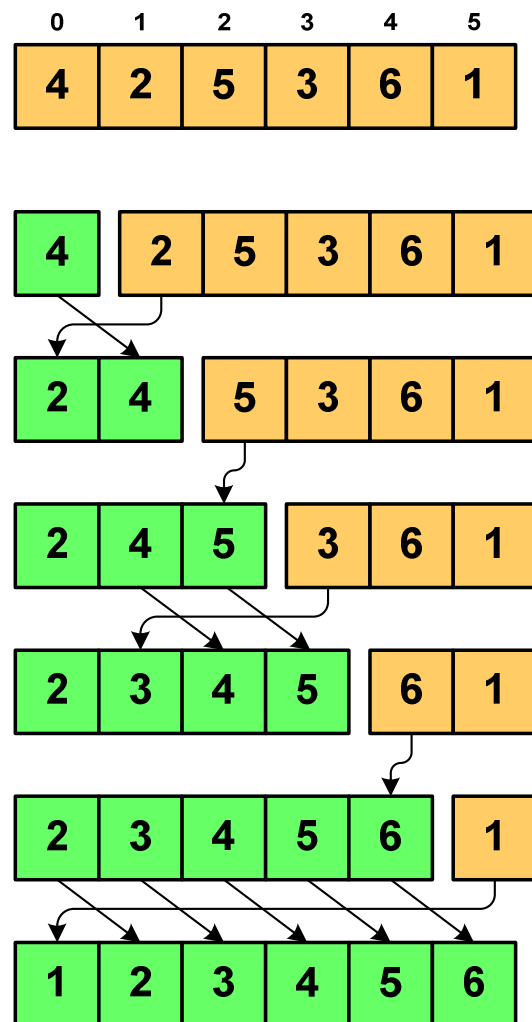
Magda	Ola	Jan	Ela	Paweł	Piotr
Mazur	Nowak	Kamiński	Król	Wójcik	Kowalski
15	18	20	22	23	25

# Sortowanie

- Po co stosować sortowanie?
  - posortowane elementy można szybciej zlokalizować
  - posortowane elementy można przedstawić w czytelniejszy sposób
  
- Przykładowe algorytmy sortowania
  - proste wstawianie (insertion sort)
  - proste wybieranie (selection sort)
  - bąbelkowe (bubble sort)
  - szybkie (quick sort)
  - przez scalanie (merge sort)
  - kubełkowe / przez zliczanie (bucket sort)

# Proste wstawianie (insertion sort)

Przykład:

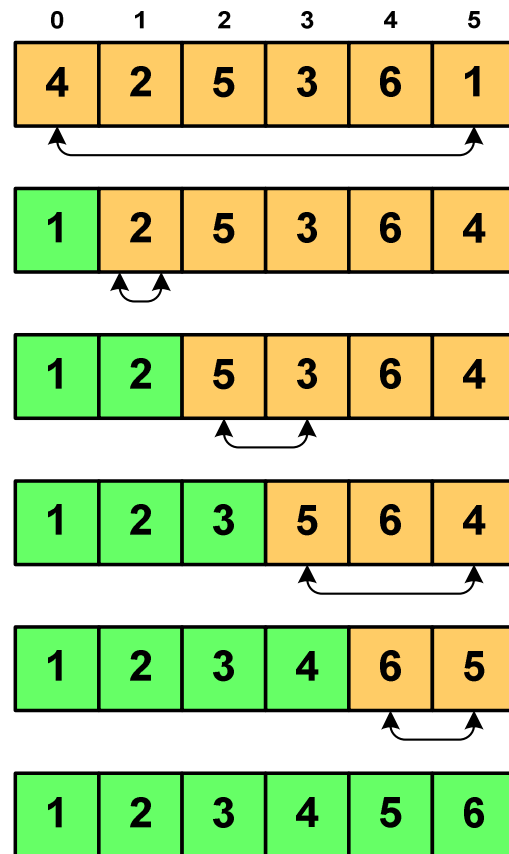


Program w języku C:

```
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 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;
    }
}
```





## 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
<b>tab</b>	<b>4</b>	<b>2</b>	<b>5</b>	<b>3</b>	<b>6</b>	<b>1</b>

- Wywołanie funkcji **QS()** ma postać:

**QS (tab, 0, 5) ;**



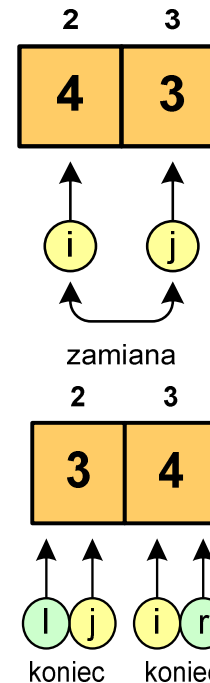




## Sortowanie szybkie (Quick-Sort) - QS(tab,2,3)

- 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

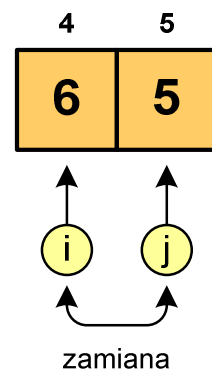


- Rozmiar obu tablic do posortowania wynosi 1 więc nie ma nowych wywołań funkcji QS()

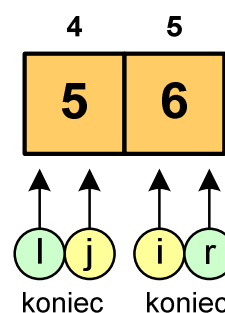
## Sortowanie szybkie (Quick-Sort) - $QS(tab, 4, 5)$

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

- 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



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

## Sortowanie szybkie (Quick-Sort) - funkcja w 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);
}
```

## Funkcja `qsort()` w języku C

- Quick-Sort został zaimplementowany w języku C w funkcji:

QSORT

stdlib.h

```
void qsort(void *baza, size_t n, size_t size,  
          (*funkcja)(const void *element1, const void *element2));
```

- funkcja `qsort()` sortuje metodą Quick-Sort tablicę wskazywaną przez argument `baza` i zawierającą `n` elementów o rozmiarze `size`
- funkcja `qsort()` posługuje się funkcją porównującą `funkcja()`, której argumentami są wskazania do elementów tablicy `baza`
- `funkcja()` powinna zwracać wartości:
  - < 0,    gdy        `*element1 < *element2`
  - == 0,    gdy        `*element1 == *element2`
  - > 0,    gdy        `*element1 > *element2`

## Funkcja qsort() w języku C - przykład (1/2)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 10

void generuj(int tab[])
{
    int i;
    srand(time(NULL));
    for (i=0;i<N;i++)
        tab[i]=rand()%100;
}

void drukuj(int tab[])
{
    int i;
    for (i=0;i<N;i++)
        printf("%2d  ",tab[i]);
    printf("\n");
}
```

## Funkcja qsort() w języku C - przykład (2/2)

```
int funkcja(const void *element1, const void *element2)
{
    if (*(int*)element1 < *(int*)element2) return -1;
    if (*(int*)element1 == *(int*)element2) return 0;
    if (*(int*)element1 > *(int*)element2) return 1;
}

int main()
{
    int tab[N];

    generuj(tab);
    drukuj(tab);

    printf("\nqsort:\n");
    qsort((void*)tab, (size_t)N, sizeof(int), funkcja);
    drukuj(tab);

    return 0;
}
```



## Funkcja qsort() w języku C - przykład (2/2)

```
int funkcja(const  
{  
    if (*(int*)ele  
    if (*(int*)ele  
    if (*(int*)ele  
}
```

```
int main()  
{  
    int tab[N];  
  
    generuj(tab);  
    drukuj(tab);  
  
    printf("\nqsort:\n");  
    qsort((void*)tab, (size_t)N, sizeof(int), funkcja);  
    drukuj(tab);  
  
    return 0;  
}
```

```
65  22  15  26  87  43   3  21  11  73  
  
qsort:  
 3  11  15  21  22  26  43  65  73  87
```

Koniec wykładu nr 4

Dziękuję za uwagę!