Wprowadzenie do programowania **przykłady algorytmów**

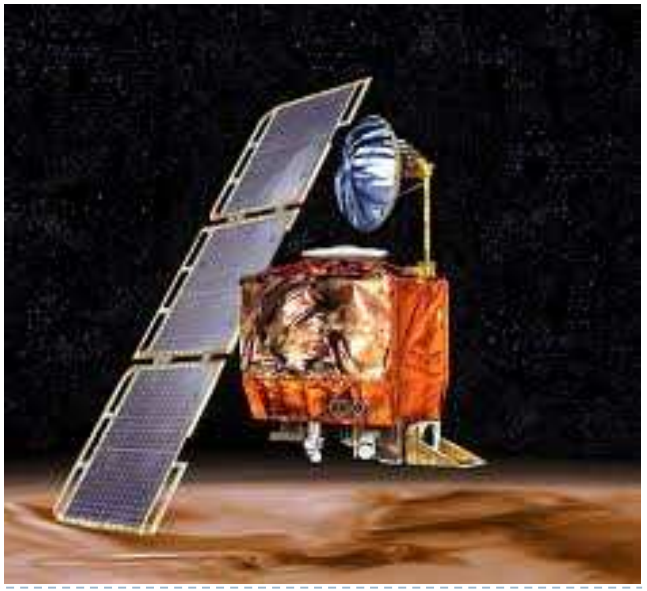
dr hab. inż. Mikołaj Morzy

plan wykładu

- cechy algorytmów
- sposoby zapisu algorytmów
- klasyfikacja algorytmów
- przykłady algorytmów
  - sumowanie
  - przeszukiwanie ciągu liczb
  - sortowanie
- algorytmy rekurencyjne

cechy algorytmu

- kompletność
  - algorytm uwzględnia wszystkie możliwe przypadki które mogą wystąpić w trakcie realizacji algorytmu
  - algorytm przewiduje wystąpienia błędów (numerycznych, logicznych, itd.) oraz posiada system reakcji (komunikaty o błędach, odpowiednie zakończenie działania)



cechy algorytmu

- skończoność
  - algorytm zapewnia osiągnięcie rozwiązania w skończonej liczbie kroków (czyli w skończonym czasie)
  - dokładna liczba kroków nie jest znana *a priori*
  - algorytm posiada warunek zakończenia

n	∑(n)
1	1
2	4
3	6
4	13
5	4096
6	10^{1439}
10	$3\uparrow\uparrow\uparrow 3$
12	$4096\uparrow\uparrow 166$

3 podniesione do potęgi 3 3^3 razy,
tj. 3 z 7 625 597 484 987 potęgami

cechy algorytmu

- jednoznaczność
  - dla tych samych danych wejściowych algorytm musi dawać zawsze te same wyniki
  - algorytm jest niezależny od momentu jego uruchomienia, wpływu innych programów, sprzętu, kodowania znaków, itp.

z czego składa się algorytm?

- **dane**
  - obiekty podlegające przekształceniu podczas działania
- **wynik**
  - ostateczny rezultat działania algorytmu
- **instrukcje**
  - opis sekwencji czynności które muszą zostać wykonane w ściśle określonej kolejności

algorytm zapisany przy pomocy języka programowania nazywamy programem

sposoby zapisu algorytmu

- lista kroków (opis słowny)
  - najbardziej naturalny sposób zapisu algorytmu
- zapis graficzny (schemat blokowy)
  - zestaw symbolicznych elementów odpowiadających czynnościom
- w pseudojęzyku programowania
- w konkretnym języku programowania

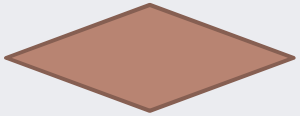
przykład listy kroków

1. Czy słuchawka jest odłożona? Jeśli tak, to przejdź do (2), jeśli nie, to odłóż słuchawkę.
2. Podnieś słuchawkę.
3. Wybierz cyfrę 6.
4. Wybierz cyfrę 1.
5. Wybierz cyfrę 6.
6. Wykonaj czynność cztery razy
    1. wybierz cyfrę 2
7. Czy połączyłeś się z fryzjerem? Jeśli tak, to przejdź do (8), w przeciwnym wypadku przejdź do (9)
8. Umów się na strzyżenie włosów.
9. Odłóż słuchawkę.

schematy blokowe

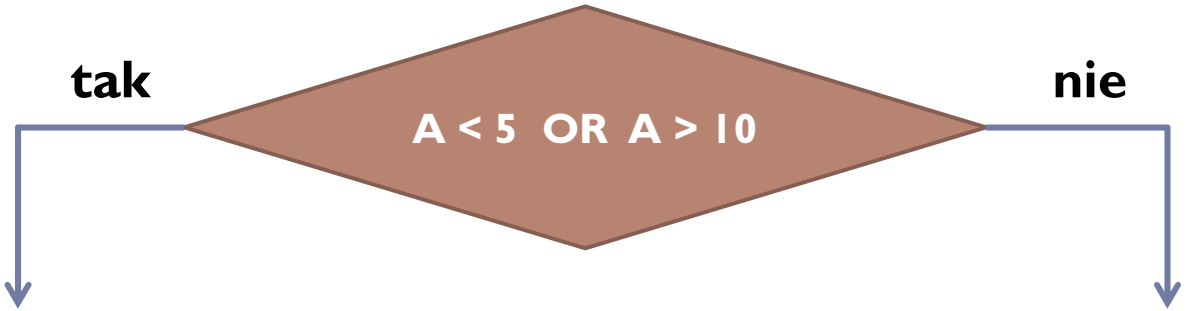
- dla skomplikowanych algorytmów zapis w postaci języka naturalnego szybko staje się nieczytelny i zagmatwany
- schemat blokowy to graficzny zapis algorytmu
  - operacje przedstawione są za pomocą odpowiednio połączonych bloków
  - połączenia określają kolejność i sposób wykonywania operacji
  - istnieje standardowy, ogólnie przyjęty system oznaczeń graficznych dla schematów blokowych

symbole używane w schematach blokowych

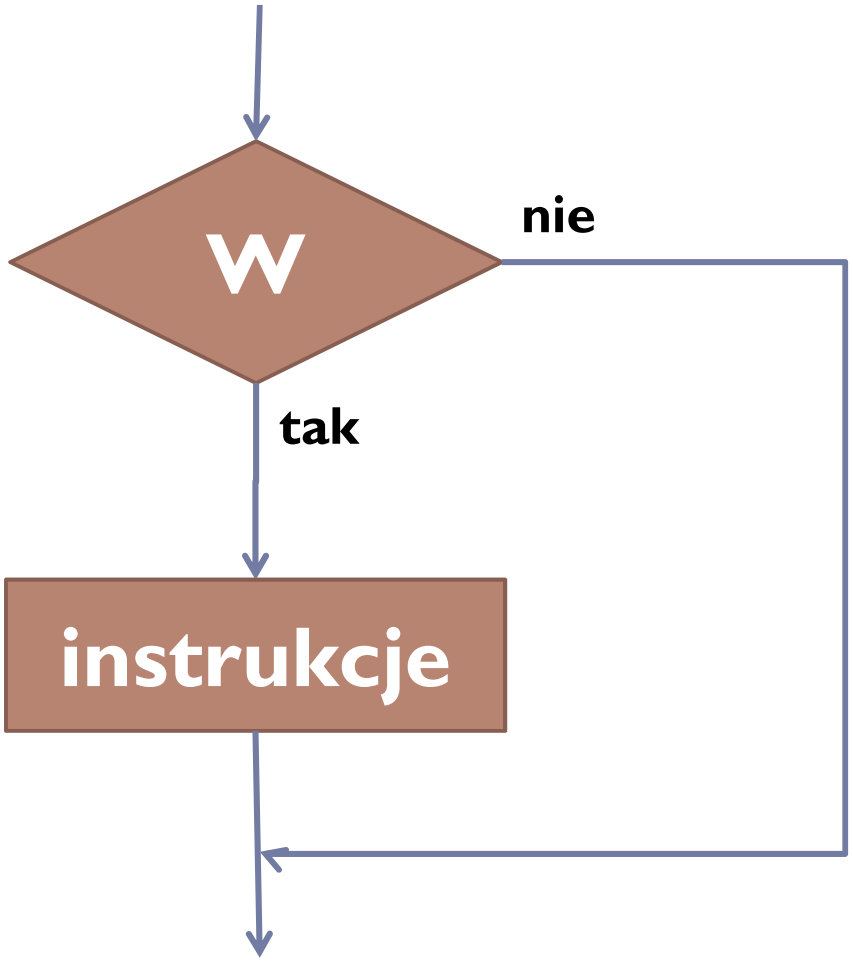
symbol	nazwa	opis
	początek, koniec	oznacza miejsce rozpoczęcia i zakończenia algorytmu
	operator	działanie do wykonania
	wejście/wyjście	wprowadzanie danych lub wyświetlanie danych
	element decyzyjny	wybór jednej z alternatywnych dróg działania
	łącznik	połączenie dwóch fragmentów sieci działania
	komentarz	miejsce na komentarz
	linia	połączenie kolejnych symboli działania

operacja wyboru

- na schemacie blokowym operacje wyboru jednej z alternatywnych dróg działania realizujemy za pomocą skrzynki warunkowej
- wewnątrz skrzynki warunkowej umieszczamy warunek logiczny
  - operatory: **= <> < > <= >= OR AND**

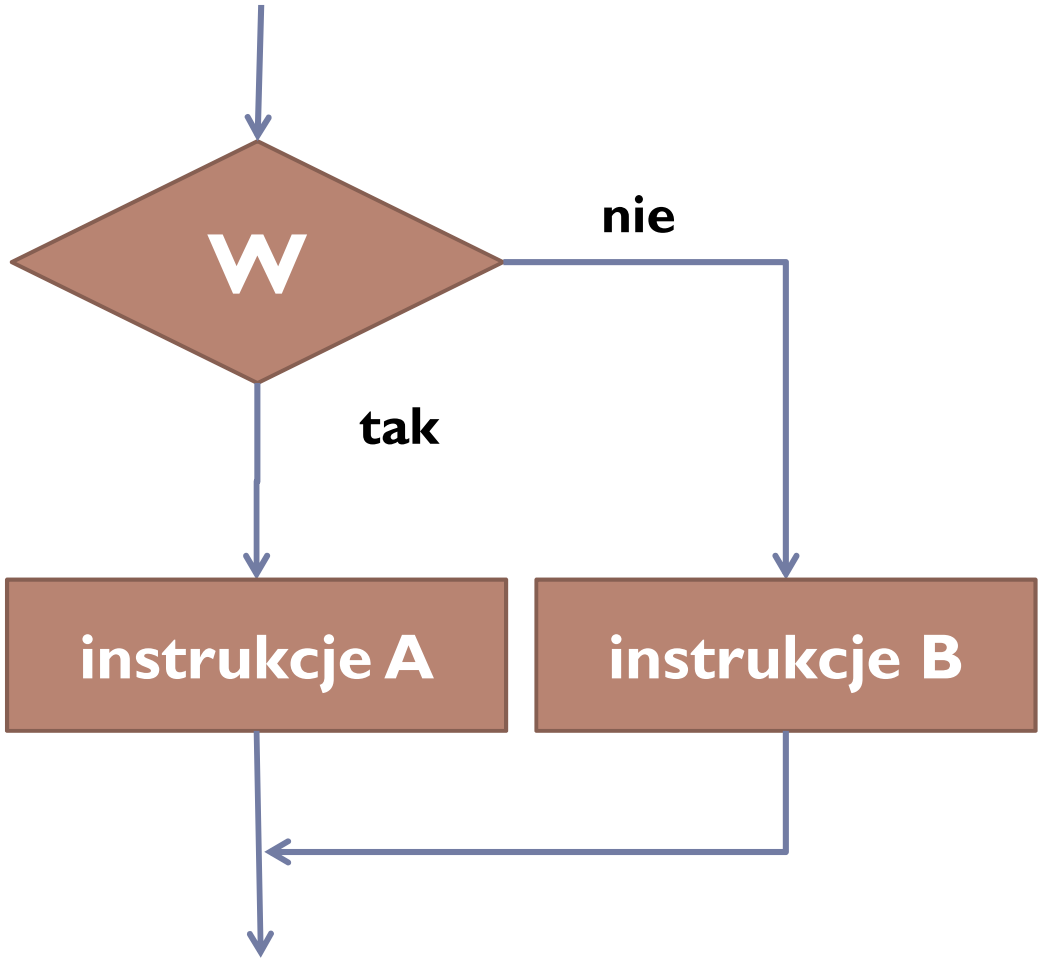


instrukcja warunkowa



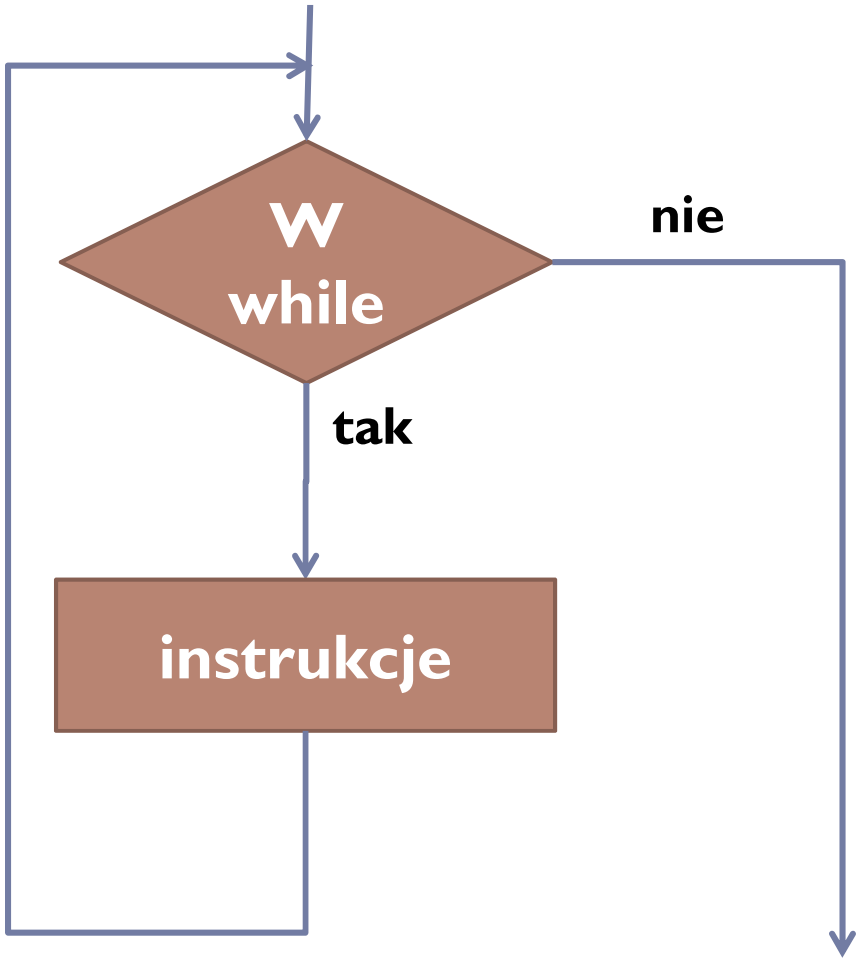
- instrukcje są realizowane jeśli spełniony jest warunek **W**

instrukcja warunkowa



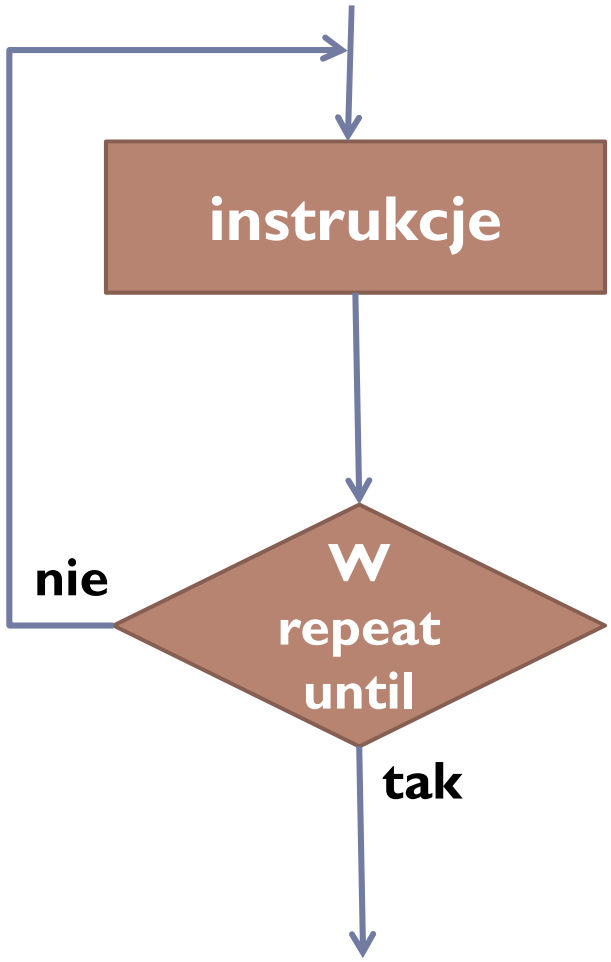
- jeśli spełniony jest warunek **W** to realizowane są instrukcje A, w przeciwnym wypadku realizowane są instrukcje B

instrukcja pętli "dopóki"



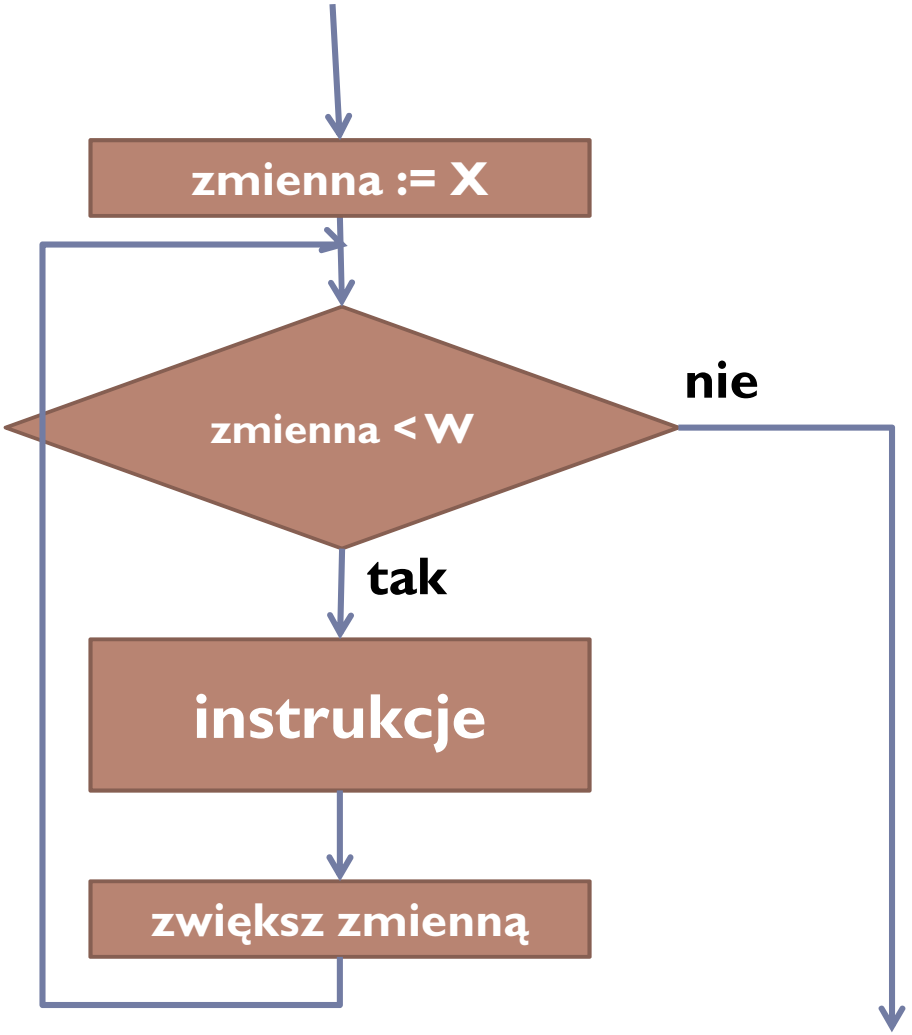
- dopóki spełniony jest warunek **W** to instrukcje są powtarzane

instrukcja pętli "powtarzaj aż"



- instrukcje są powtarzane aż do spełnienia jest warunek **W**

instrukcja pętli "powtarzaj *n* raz"

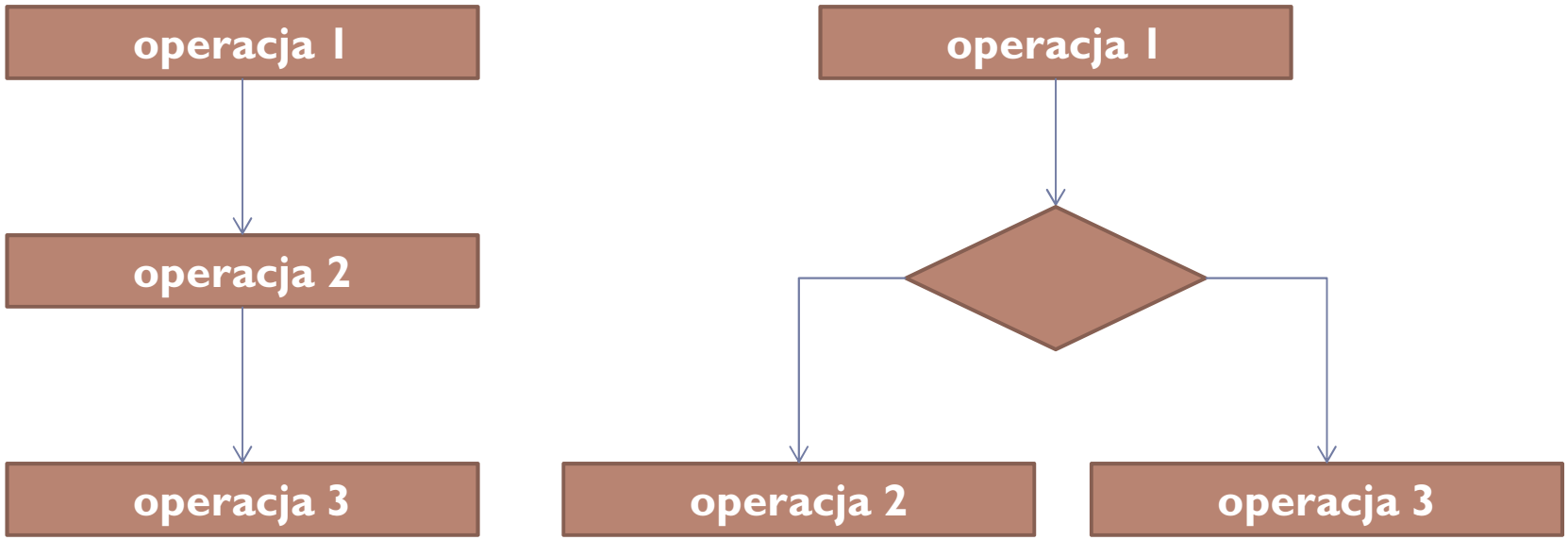


- zainicjalizuj zmienną za pomocą wartości
- powtarzaj określoną liczbę razy
  - jeśli zmienna jest poniżej warunku zmiennej kontrolnej, to wejdź do pętli
  - jeśli warunek jest fałszywy, to następuje koniec przetwarzania

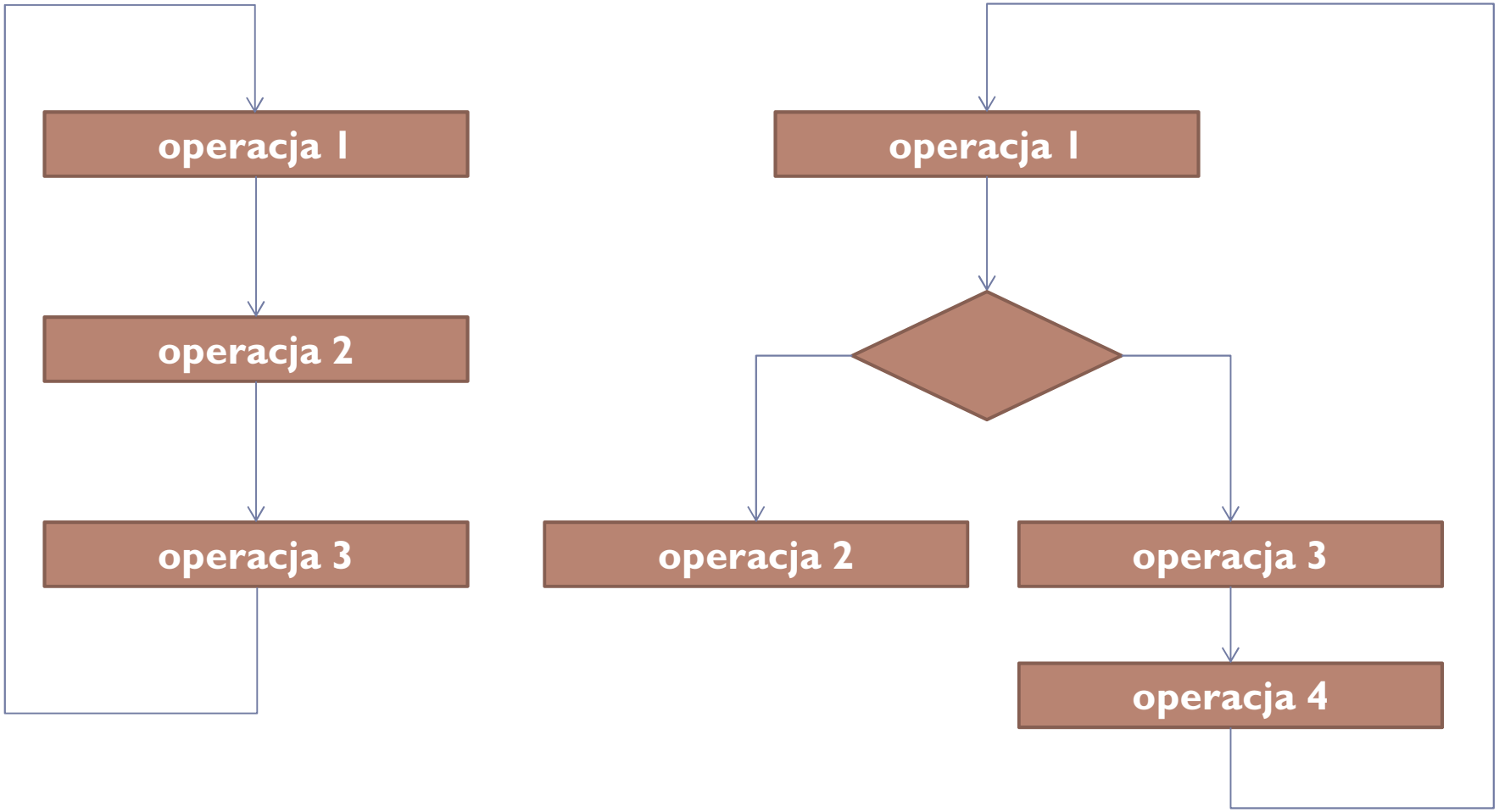
klasyfikacja algorytmów

- algorytmy proste i rozgałęzione
- algorytmy cykliczne i mieszane
- algorytmy sekwencyjne i równoległe/współbieżne
- algorytmy numeryczne i nienumeryczne
- algorytmy rekurencyjne i iteracyjne
  - **iteracja**: wielokrotne wywołanie sekwencji poleceń aż do spełnienia warunku sterującego
  - **rekurencja**: wywołanie procedury z wewnątrz tej samej procedury

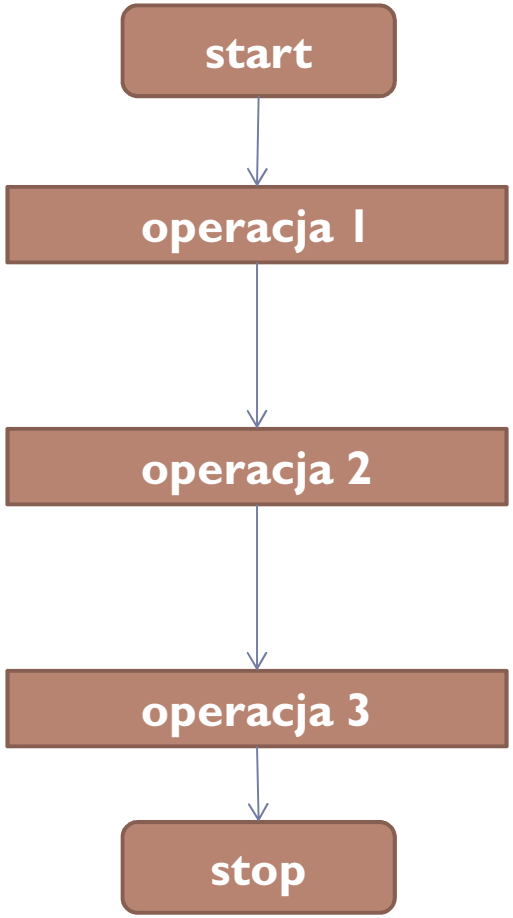
algorytm prosty i rozgałęziony

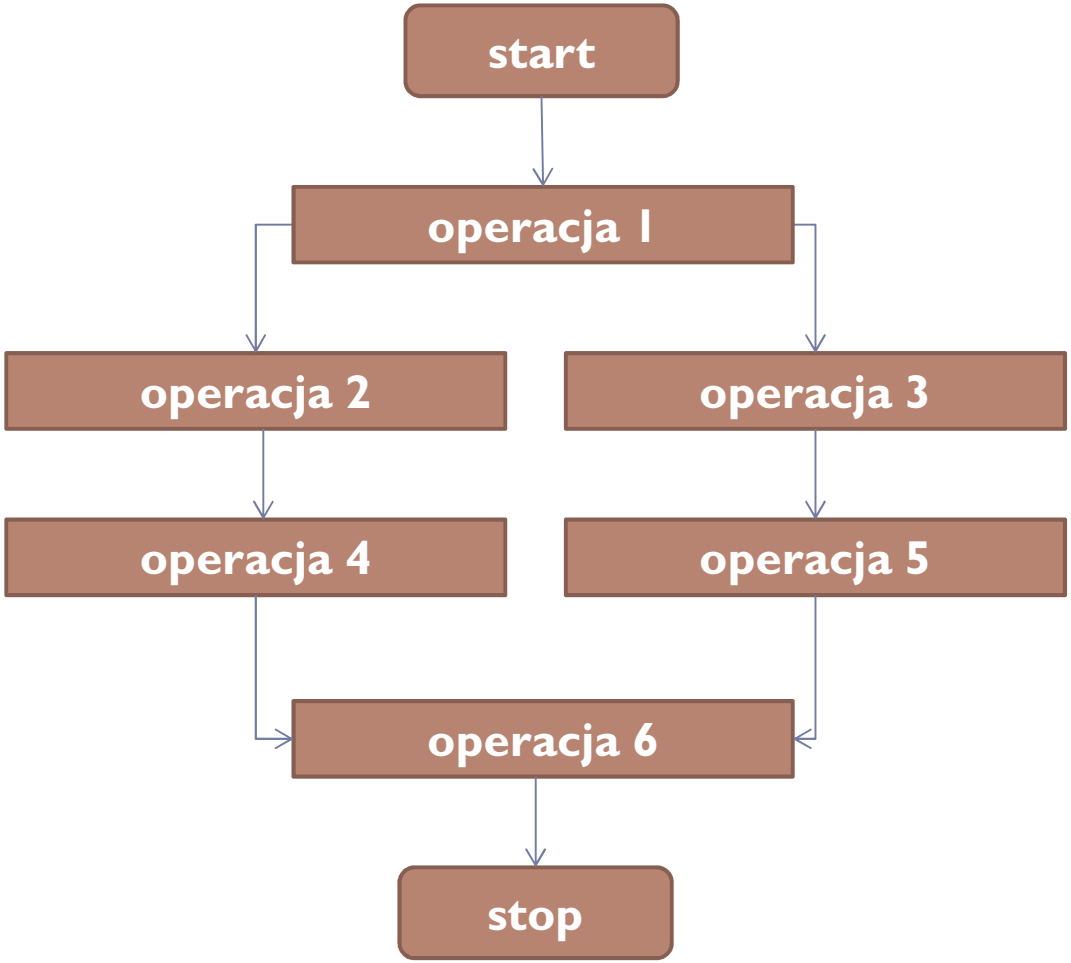


algorytm cykliczny i mieszany

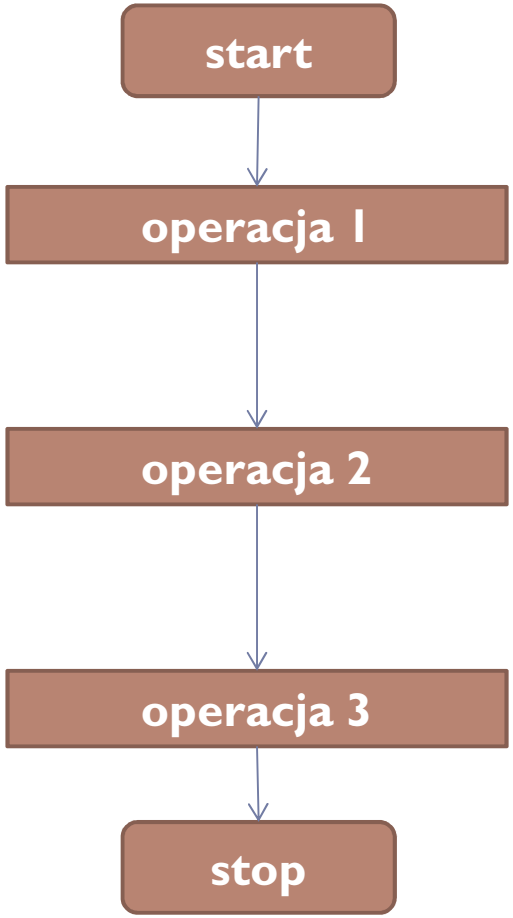


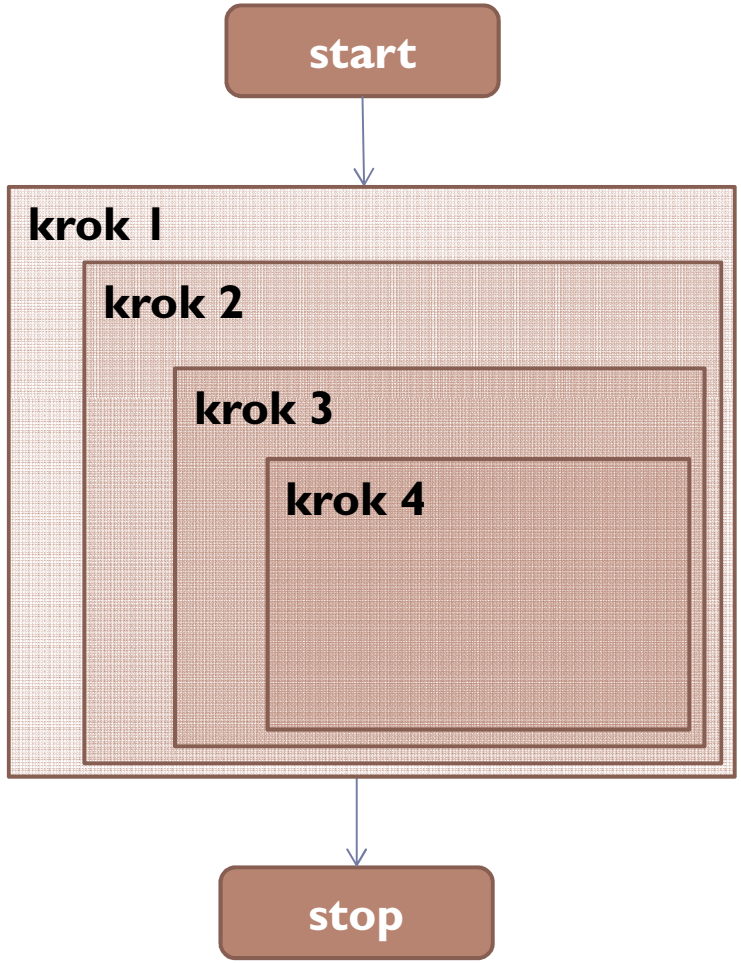
algorytm sekwencyjny i równoległy





algorytm iteracyjny i rekurencyjny

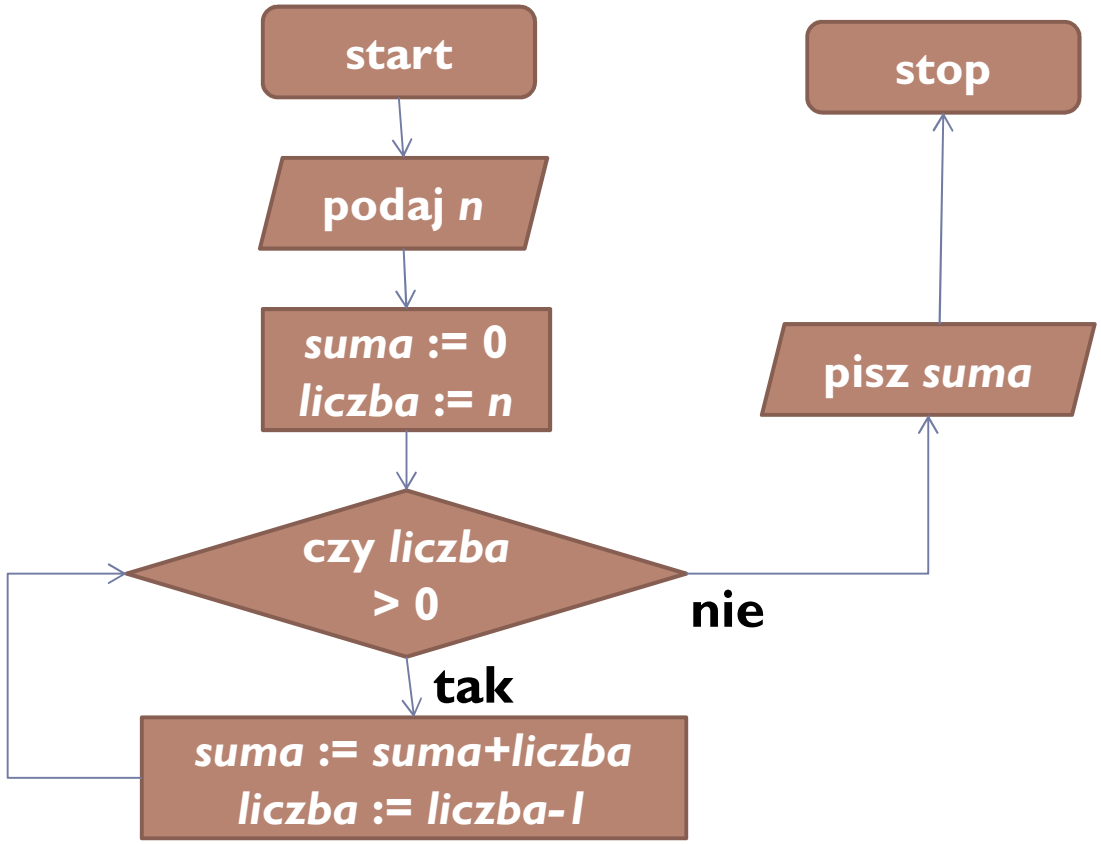




przegląd podstawowych algorytmów

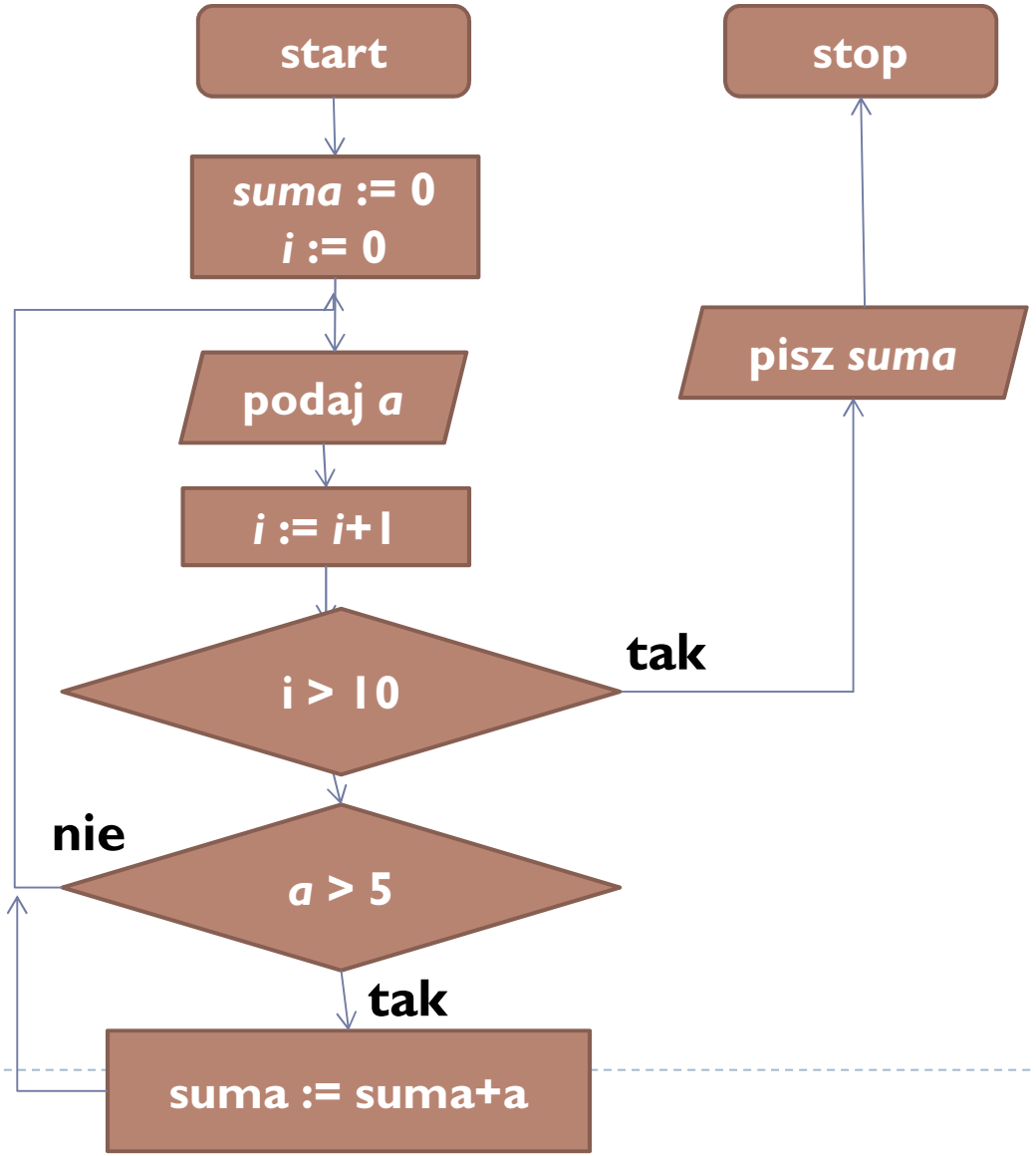
sumowanie liczb naturalnych

- zadanie
  - zsumuj liczby naturalne z zakresu 1.. *n*



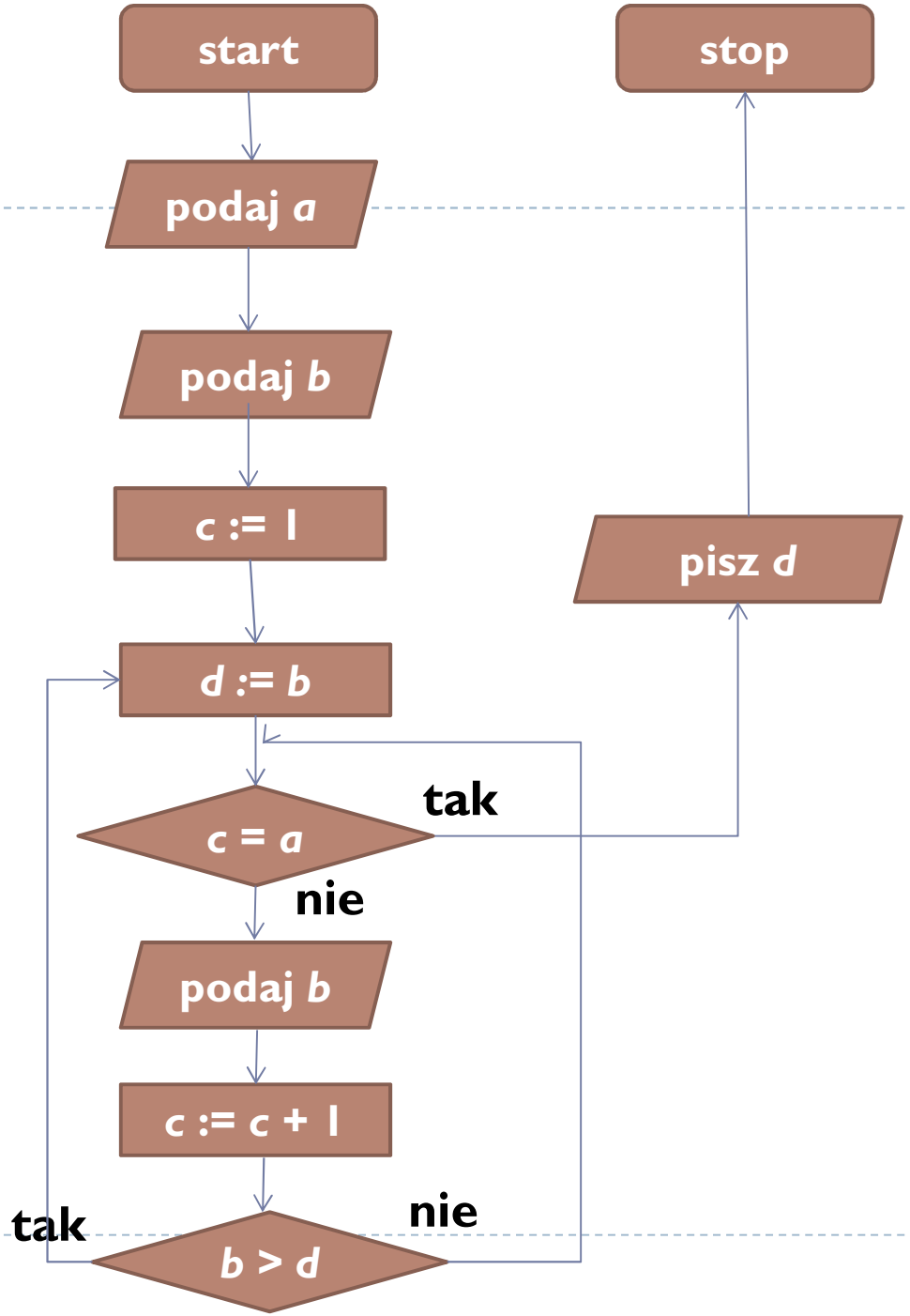
sumowanie liczb naturalnych

- zadanie
  - wylicz sumę z podanych 10 liczb, uwzględnij tylko liczby większe od 5



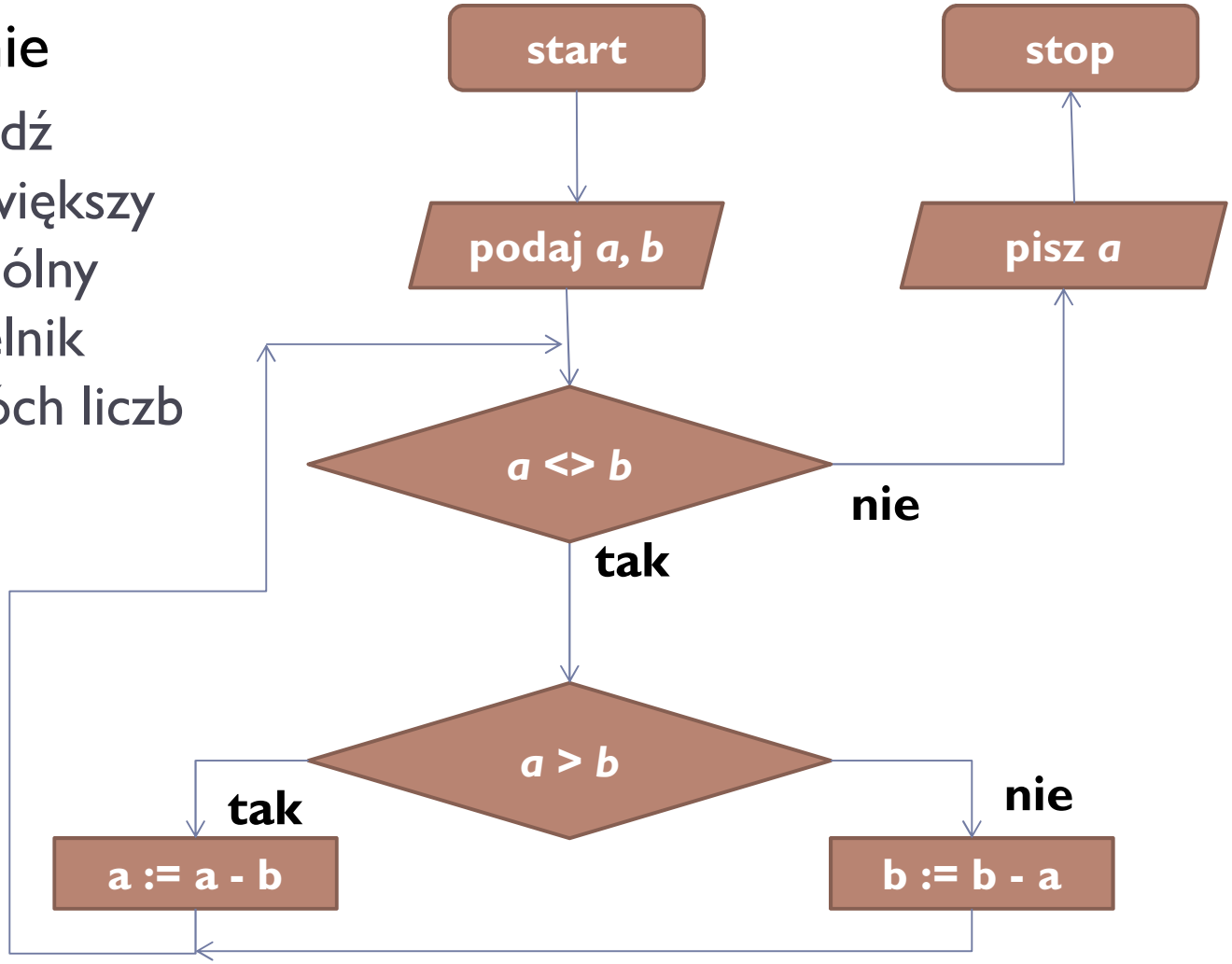
zagadka

- zadanie
  - co robi ten algorytm?
  - jak go poprawić?



algorytm Euklidesa

- zadanie
  - znajdź największy wspólny dzielnik dwóch liczb

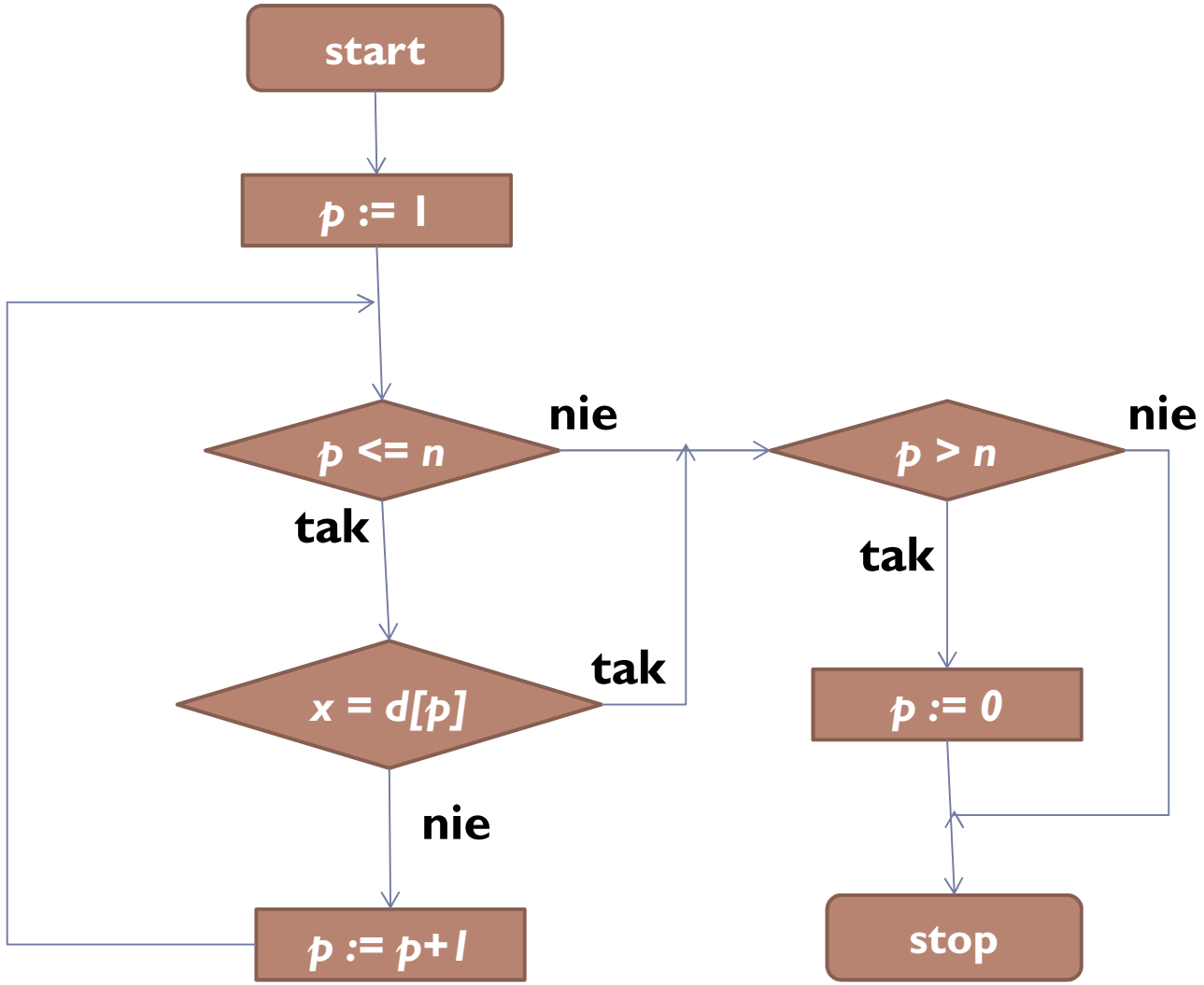


przeszukiwanie sekwencyjne: lista kroków

- dane wejściowe
  - **n** liczba elementów w zbiorze
  - **d[.]** zbiór wejściowy
  - **x** poszukiwana wartość

- algorytm
  1. p := 1
  2. dopóki (p ≤ n) i (d[p] <> x) to
     1. p := p + 1
  3. jeśli p > n to wówczas p := 0
  4. zakończ algorytm

przeszukiwanie sekwencyjne: schemat blokowy



przeszukiwanie z wartownikiem

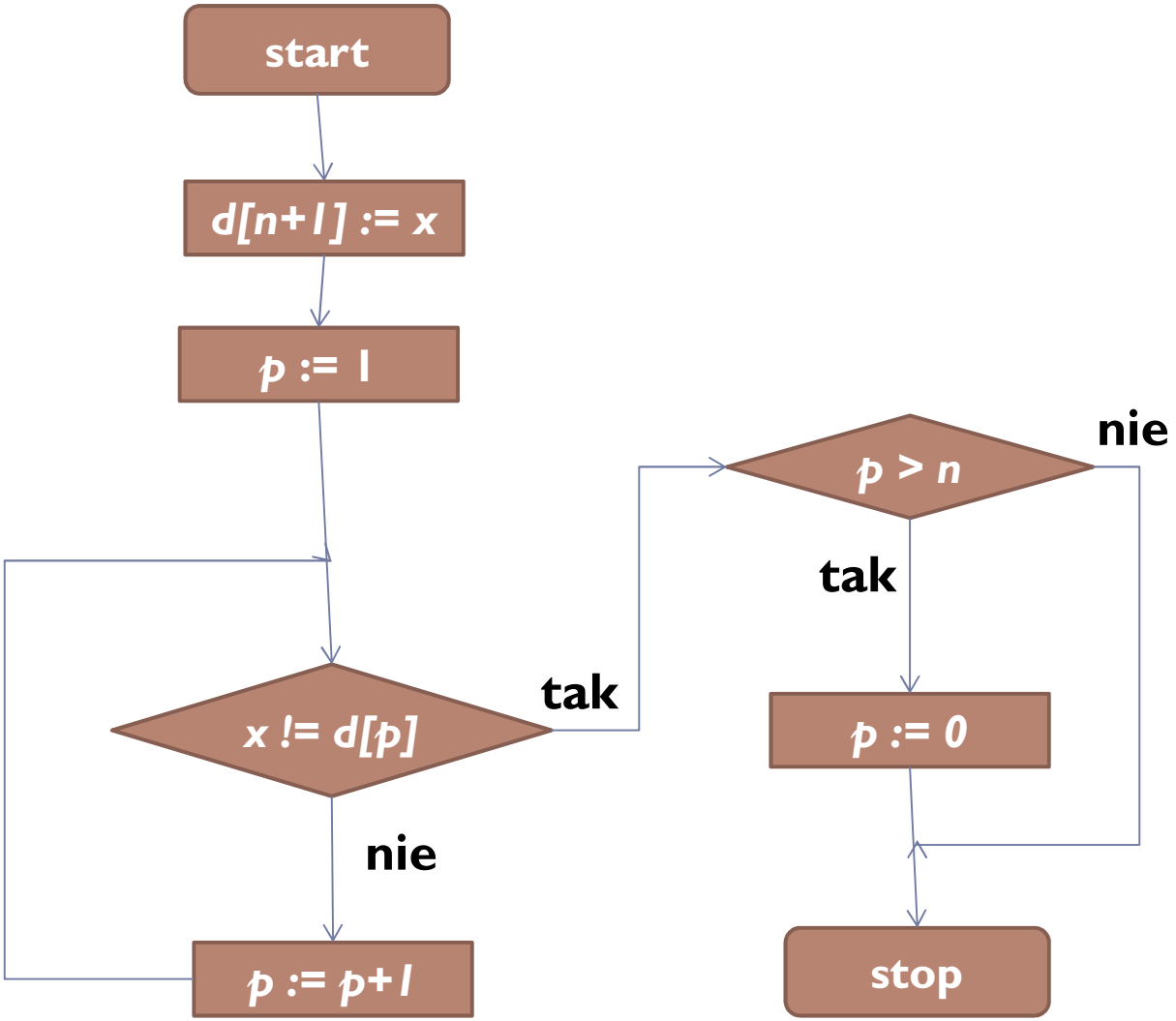
- generalna zasada wyszukiwania
  - warunek sprawdzania zakresu (p <= n) jest potrzebny jedynie w przypadku, gdy zbiór d[.] nie zawiera poszukiwanego elementu (gwarancja zakończenia pętli)
  - jeśli można zagwarantować, że element zawsze zostanie znaleziony, to warunek staje się zbyteczny
  - na końcu zbioru d[.] dodajemy poszukiwany element x. jeśli algorytm zwróci w wyniku pozycję n+1, to wiemy, że znaleziony został wartownik (przeszukiwanie jest nieudane).

przeszukiwanie z wartownikiem: lista kroków

- dane wejściowe
  - **n** liczba elementów w zbiorze
  - **d[.]** zbiór wejściowy
  - **x** poszukiwana wartość

- algorytm
  1. d[n+1] = x
  2. p := 1
  3. dopóki (d[p] <> x) to
     1. p := p + 1
  4. jeśli p > n to wówczas p := 0
  5. zakończ algorytm

przeszukiwanie z wartownikiem: schemat blokowy



wyszukiwanie najczęstszego elementu

- dane wejściowe
  - **n** liczba elementów w zbiorze, $n \in N, n>0$
  - **d[.]** zbiór wejściowy

- algorytm
  1. $w_n := d[1], \quad p_n := 1, \quad l_n := 1$
  2. dla i = 1, 2, …, n wykonuj kroki (3) .. (5)
  3. licznik := 0
  4. dla j = 1, 2, …, n
     1. jeśli d[i] = d[j] to licznik++
  5. jeśli licznik > l_n to o
     1. $w_n := d[i], \quad p_n := i, \quad l_n :=$ licznik
  6. zakończ algorytm

sito Erastotenesa

- najszybszy algorytm wyszukiwania liczb pierwszych w ograniczonym zbiorze
- **idea**: usuń z zadanego zbioru liczb naturalnych wielokrotności tych liczb, które nie zostały usunięte

- przykład: znajdź liczby pierwsze w zbiorze <1..20>

usuń	pozostała sekwencja
1	2 3 4 5 6 … 18 19 20
2	2 3 5 7 9 11 … 19 20
3	2 3 5 7 11 13 17 19

zadanie samodzielne

- zapisz sito Erastotenesa jako listę kroków i za pomocą schematu blokowego

sortowanie bąbelkowe

- sortowanie polega na cyklicznym porównywaniu par elementów sąsiadujących ze sobą w zbiorze
  - jeśli elementy nie spełniają kryterium porządkowania zbioru, to następuje zamiana miejsc między elementami
  - algorytm działa aż do wyczerpania zasobów

I przebieg	II przebieg	II przebieg
4 2 8 6 1	2 4 6 1 8	2 4 1 6 8
2 4 8 6 1	2 4 6 1 8	2 4 1 6 8
2 4 8 6 1	2 4 6 1 8	2 1 4 6 8
2 4 6 8 1	2 4 1 6 8	2 1 4 6 8
2 4 6 1 8	2 4 1 6 8	2 1 4 6 8

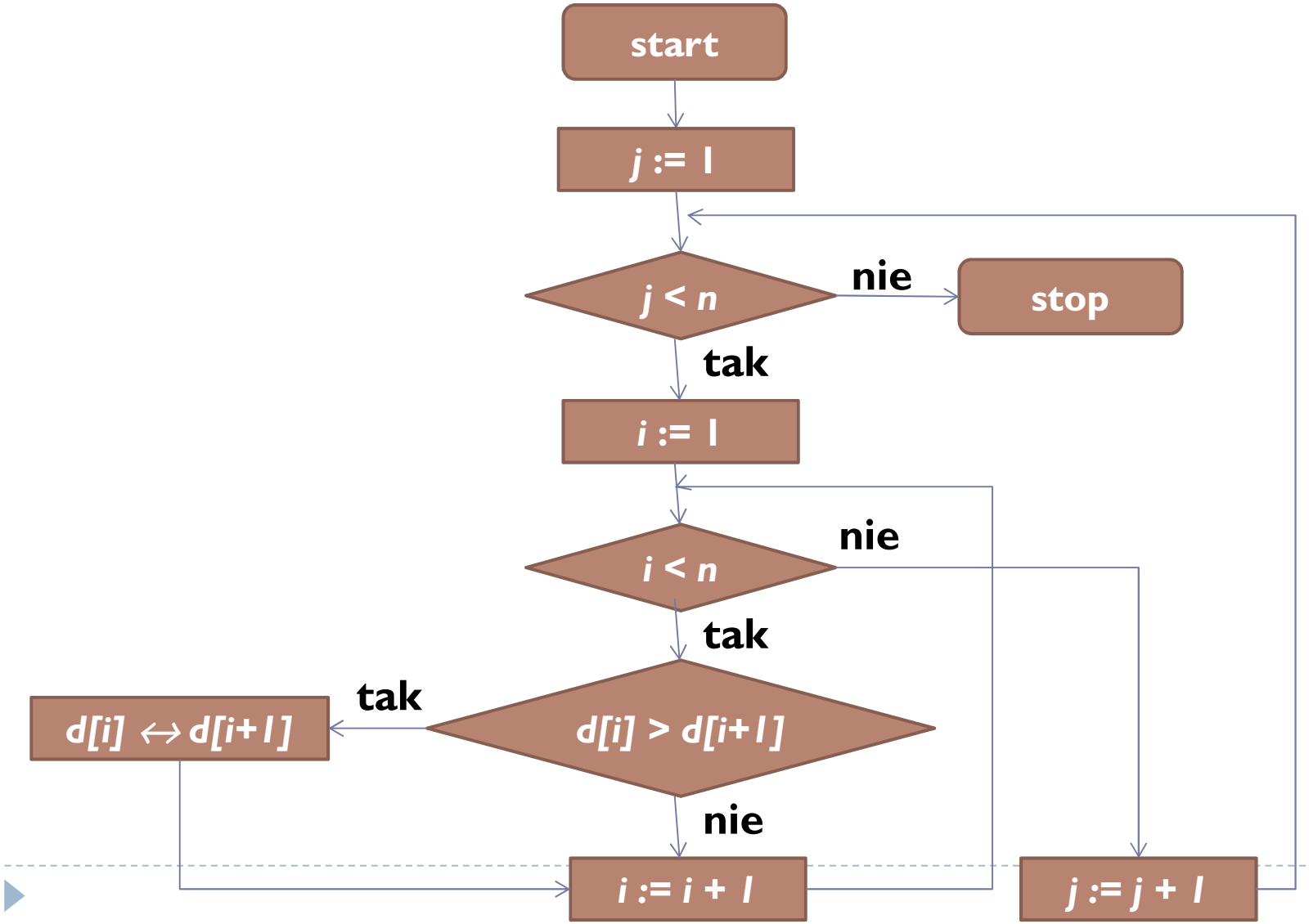
...

sortowanie bąbelkowe: lista kroków

- dane wejściowe
  - **n** liczba elementów w zbiorze, $n \in N$, $n>0$
  - **d[.]** zbiór wejściowy do posortowania

- algorytm
  1. dla j = 1, 2, …, n-1 wykonaj
     1. wykonuj krok (2)
  2. dla i = 1, 2, …, n-1 wykonaj
     1. jeśli d[i] > d[i+1] wówczas zamień d[i] i d[i+1]
  3. zakończ algorytm

sortowanie bąbelkowe: schemat blokowy



zadanie samodzielne

- wykorzystując algorytm sortowania bąbelkowego posortuj zbiór

8 5 2 9 3 6

I przebieg	II przebieg	III przebieg
8 5 2 9 3 6	5 2 8 3 6 9	2 5 3 6 8 9
5 8 2 9 3 6	2 5 8 3 6 9	2 5 3 6 8 9
5 2 8 9 3 6	2 5 8 3 6 9	2 3 5 6 8 9
5 2 8 9 3 6	2 5 3 8 6 9	2 3 5 6 8 9
5 2 8 3 9 6	2 5 3 6 8 9	2 3 5 6 8 9
5 2 8 3 6 9	2 5 3 6 8 9	2 3 5 6 8 9

sortowanie przez wybór

- sortowanie polega na cyklicznym znajdowaniu najmniejszego elementu w zbiorze i wymienianiu tego elementu z pierwszym elementem sortowanej części

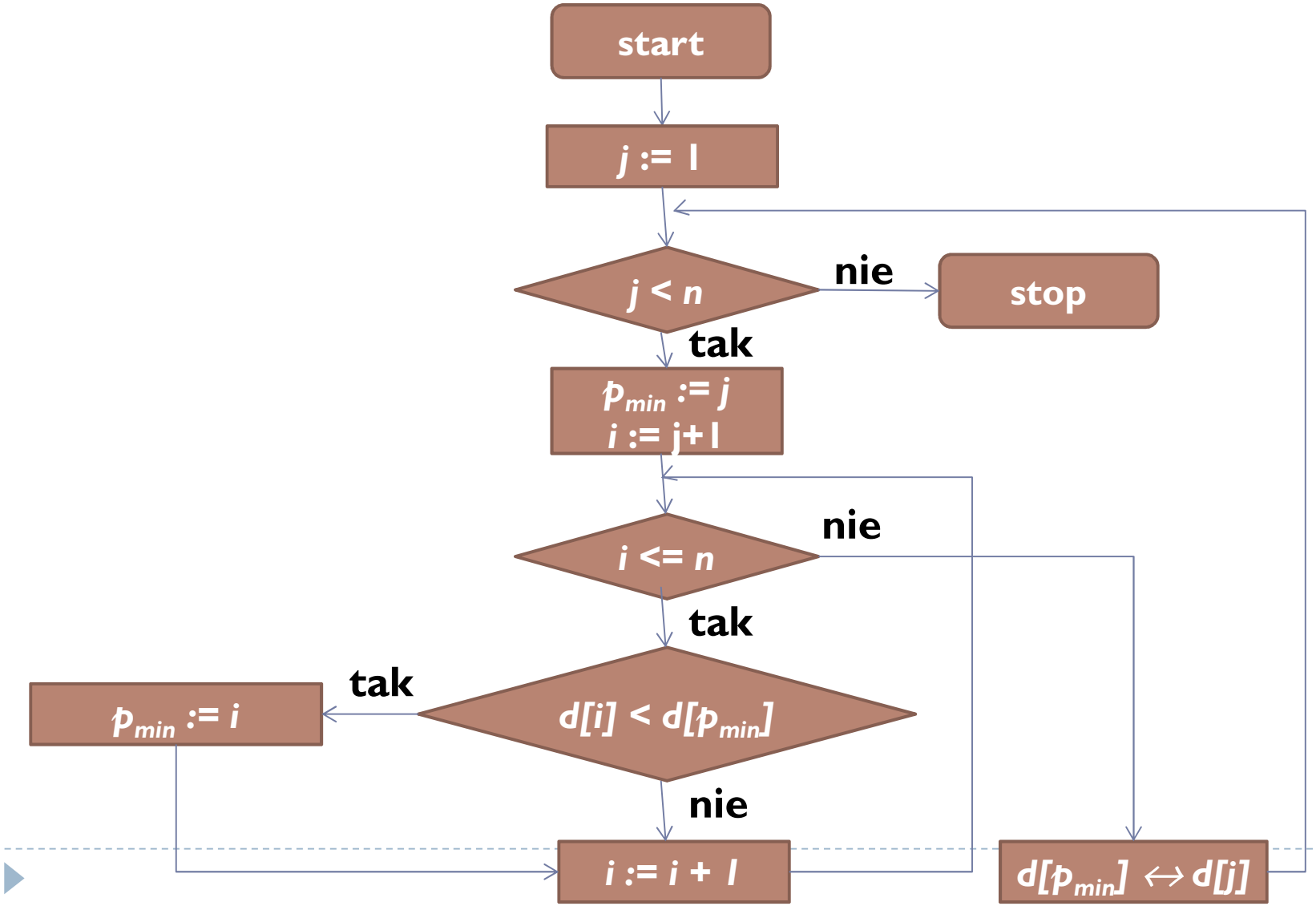
zbiór	operacja
4 7 2 9 3	najmniejszym elementem jest 2
2 7 4 9 3	zamieniamy miejscami 2 i 4
2 7 4 9 3	w pozostałej części zbioru najmniejsza jest 3
2 3 4 9 7	zamieniamy miejscami 3 i 7
2 3 4 9 7	w pozostałej części zbioru najmniejsza jest 4, która już jest na właściwej pozycji
2 3 4 9 7	w pozostałej części zbioru najmniejsza jest 7
2 3 4 7 9	zamieniamy miejscami 7 i 9
2 3 4 7 9	zbiór jest już posortowany

sortowanie przez wybór: lista kroków

- dane wejściowe
  - **n** liczba elementów w zbiorze, $n \in N$, $n>0$
  - **d[.]** zbiór wejściowy do posortowania

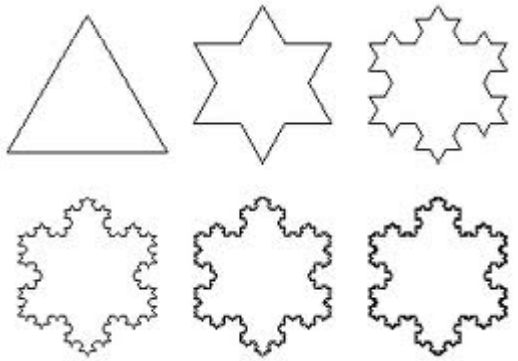
- algorytm
  1. dla j = 1, 2, …, n-1 wykonaj
     1. wykonuj kroki (2-4)
  2. p_{min} := j
  3. dla i = j+1, j+2, …, n wykonaj
     1. jeśli d[i] < d[p_{min}] wówczas p_{min} := i
  4. d[j] := d[p_{min}]
  5. zakończ algorytm

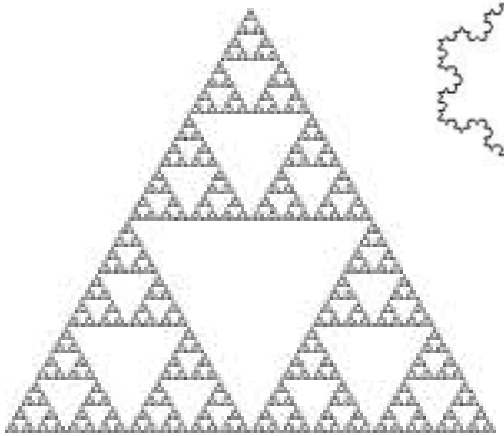
sortowanie przez wybór: schemat blokowy



rekurencja

- **rekurencja**, **rekursja** to zjawisko wywołania, w pewnym kroku algorytmu, całego algorytmu
- **obiekt rekurencyjny** to obiekt, który we fragmencie składa się z samego siebie, lub którego definicja odwołuje się do jego samego
- **funkcja rekurencyjna** to funkcja, która wywołuje samą siebie



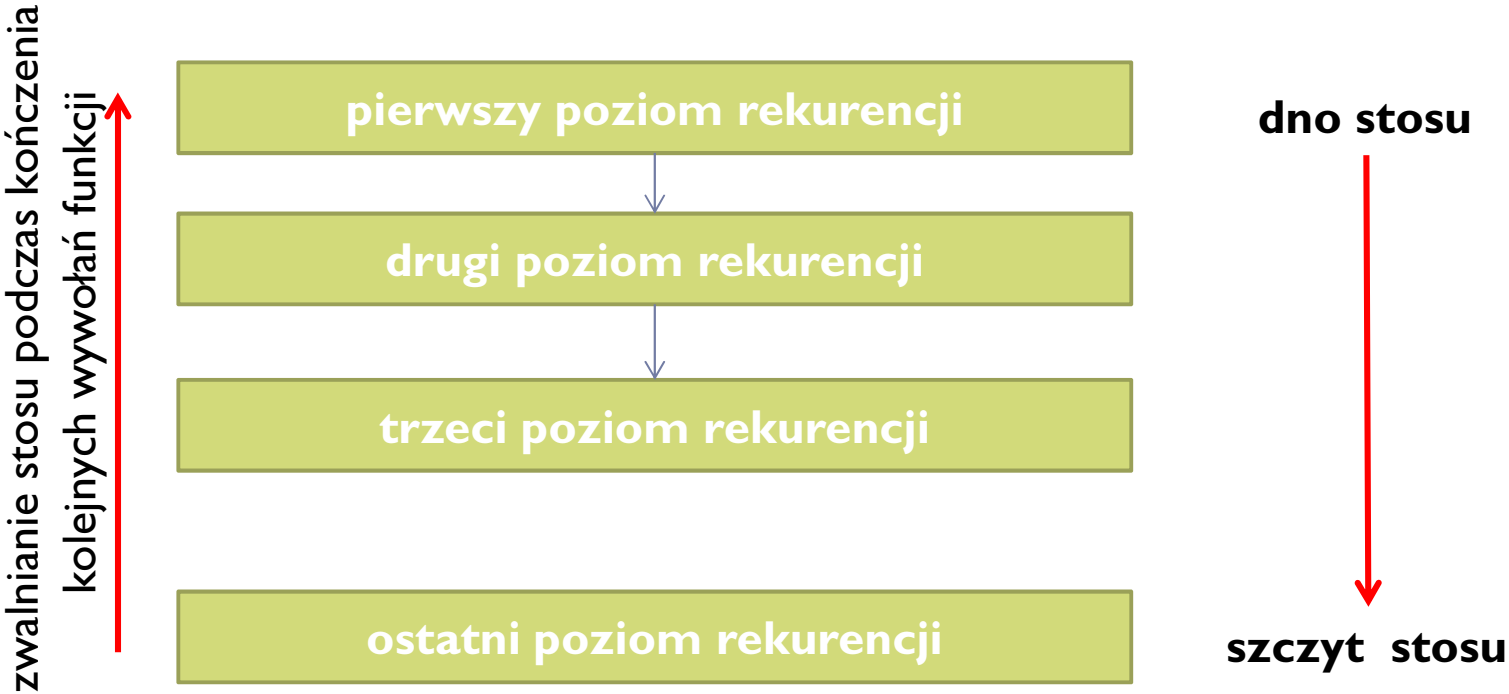


definicja rekurencyjna

- definicja rekurencyjna składa się z dwóch części
  - **warunek początkowy**: wyliczenie elementów podstawowych stanowiących części składowe wszystkich pozostałych elementów zbioru
  - **krok indukcyjny**: reguły umożliwiające konstruowanie nowych obiektów z elementów podstawowych lub obiektów zbudowanych wcześniej, reguły mogą być stosowane wielokrotnie do tworzenia nowych obiektów
  - uwaga: należy uważać, aby nie doszło do zjawiska nieskończonej pętli rekurencyjnych wywołań funkcji

wywołanie funkcji rekurencyjnej

- kolejne wywołania funkcji rekurencyjnej odkładają na stosie rekordy aktywacji procedury
- w trakcie kończenia działania poszczególnych funkcji kolejne rekordy aktywacji są zdejmowane ze stosu



przykład rekurencji: silnia

- definicja funkcji silnia

$$n! = 1 * 2 * \ldots * n$$

- definicja rekurencyjna funkcji silnia

$$n! = \begin{cases} 1 & n = 0 \\ n * (n-1)! & n > 0 \end{cases}$$

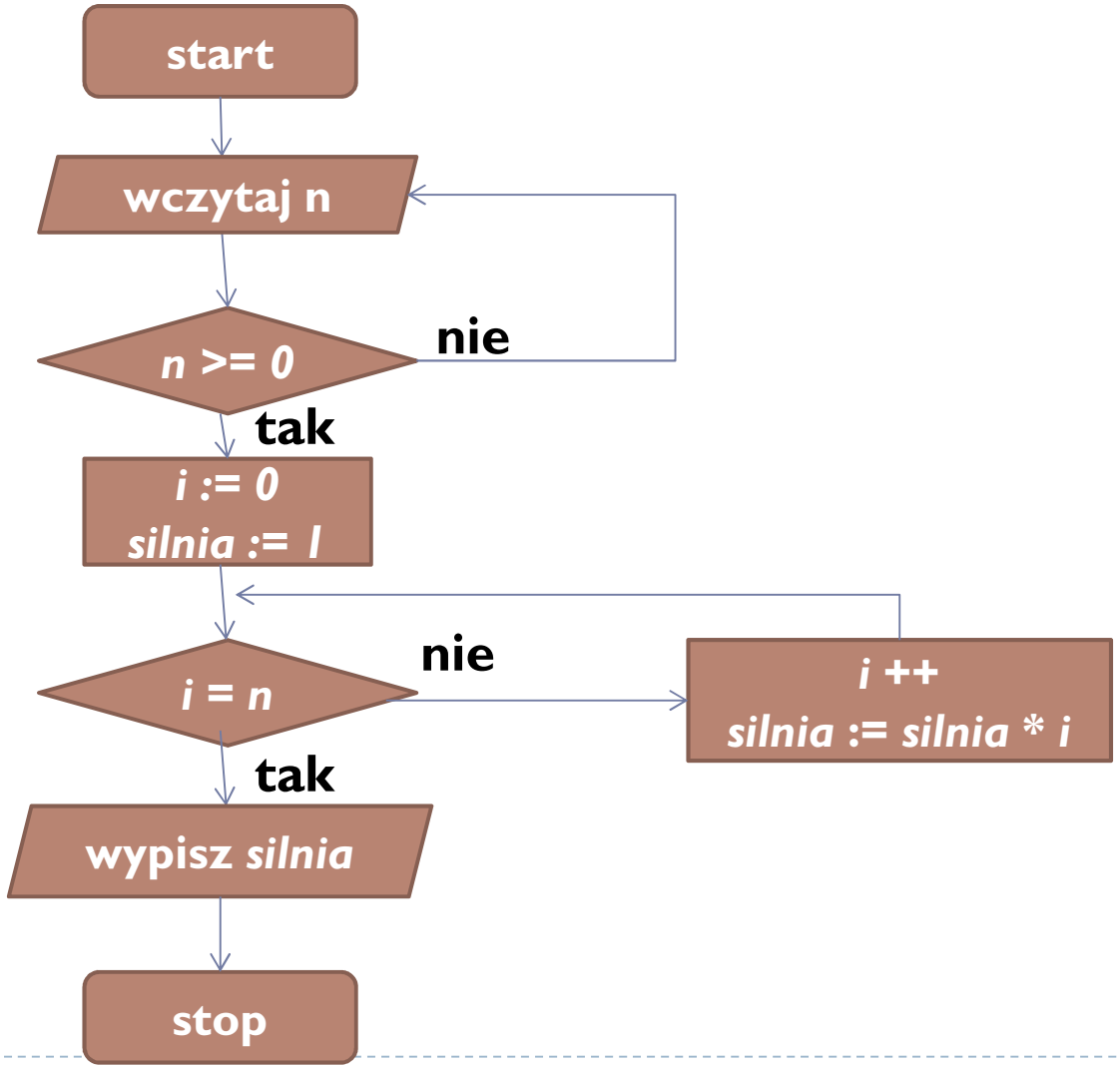
warunek początkowy

krok indukcyjny

- przykład

5! = 5 * 4! = 5 * 4 * 3! = 5 * 4 * 3 * 2! = 5 * 4 * 3 * 2 * 1! =
= 5 * 4 * 3 * 2 * 1 * 0! = 5 * 4 * 3 * 2 * 1 * 1 = 120

rekurencyjna silnia: schemat blokowy



zadanie samodzielne

- opracuj rekurencyjną wersję algorytmu wyznaczania n-tej liczby Fibonacciego
- narysuj schemat blokowy opracowanego algorytmu