INFORMATYKA

SORTOWANIE DANYCH

http://www.infoceram.agh.edu.pl

SORTOWANIE

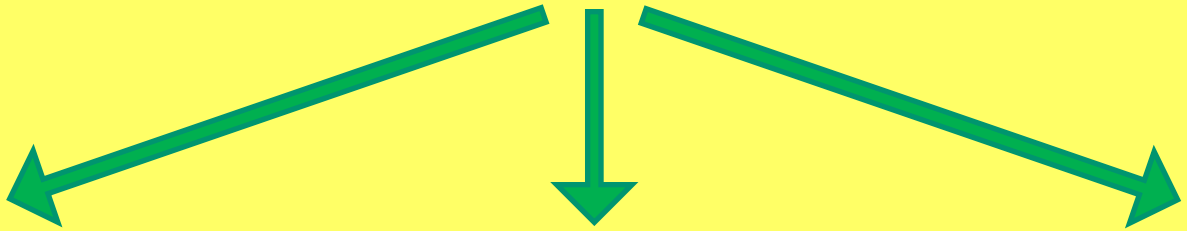
Jest to proces ustawiania zbioru obiektów w określonym porządku. Sortowanie stosowane jest w celu ułatwienia późniejszego wyszukania konkretnego elementu danego zbioru. Sortowanie jest w wielu dziedzinach podstawową, powszechnie spotykaną działalnością. Sortowanie jest szczególnie istotne w procesie przetwarzania danych. Szczególnym przypadkiem sortowania jest sortowanie względem wartości każdego elementu, np. sortowanie liczb, słów, itp.

PRZYKŁADY SORTOWANIA
- segregacja śmieci

PRZYKŁADY SORTOWANIA
- segregacja śmieci

PRZYKŁADY SORTOWANIA
- segregacja śmieci

Śmiecie nieposegregowane

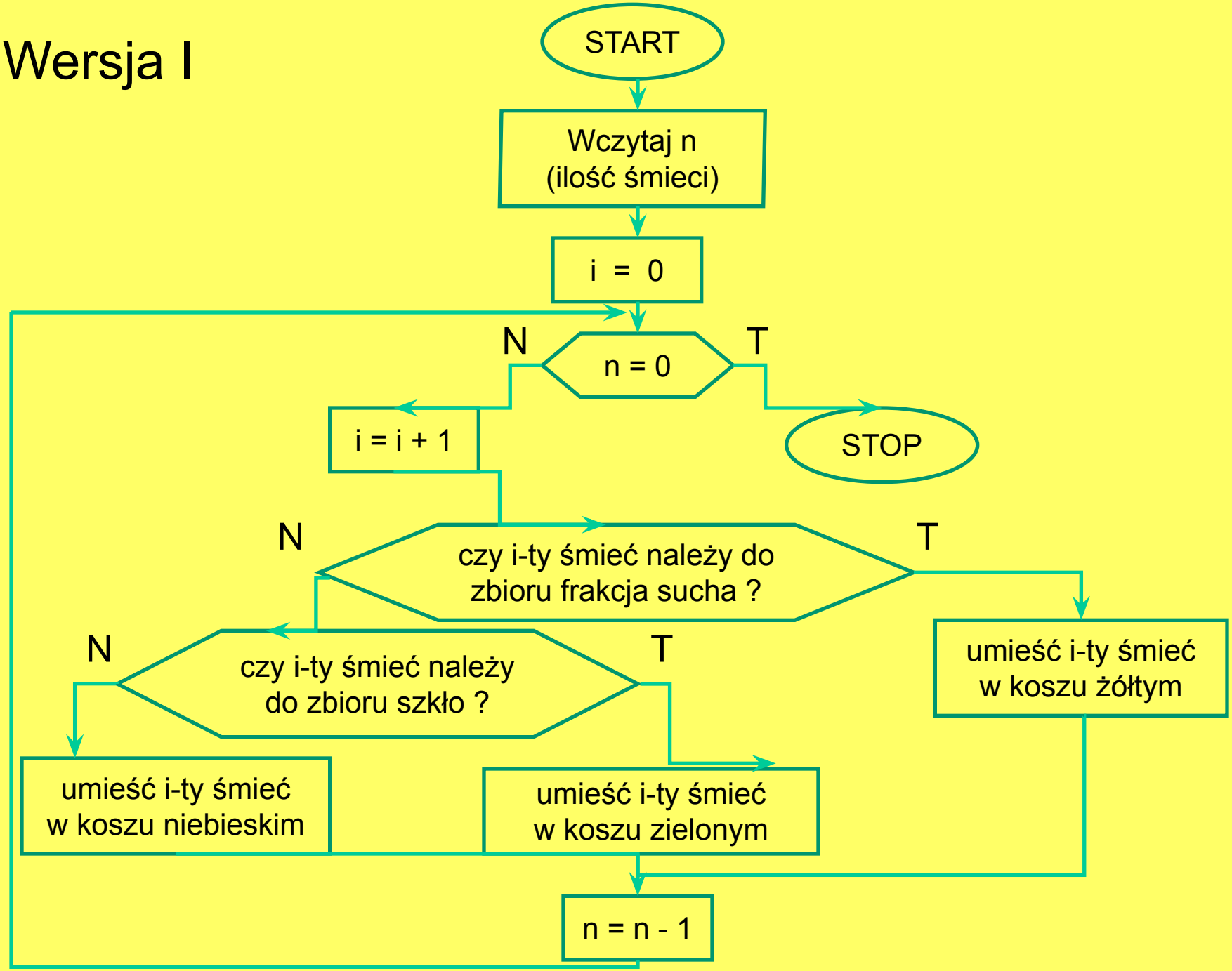


Frakcja sucha

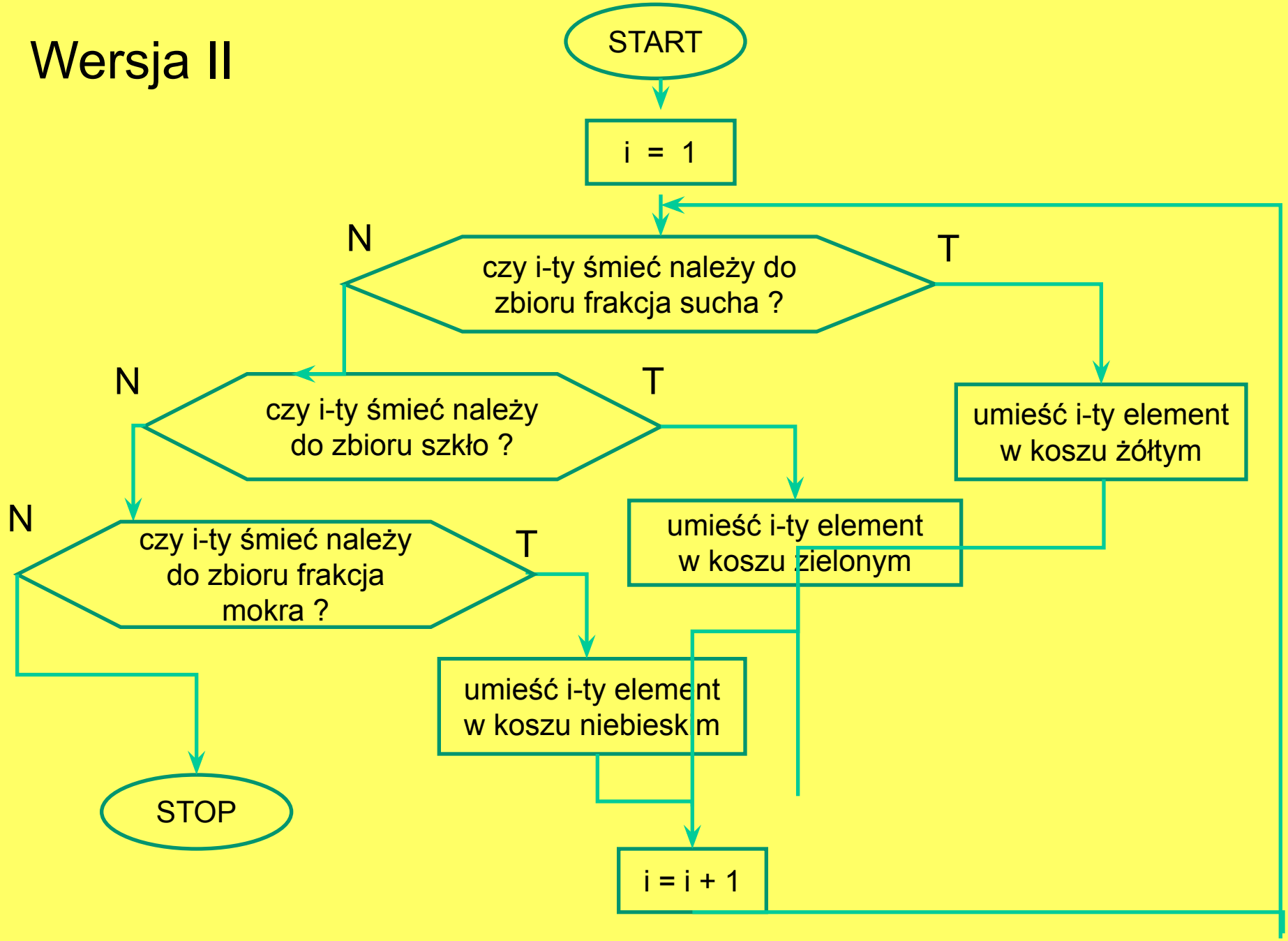
Frakcja mokra

Szkło

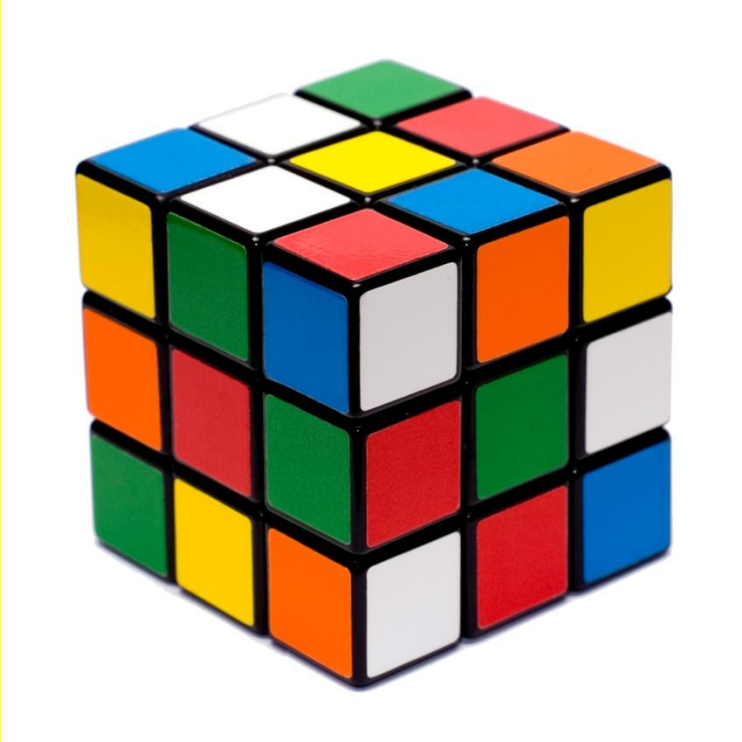
Wersja I

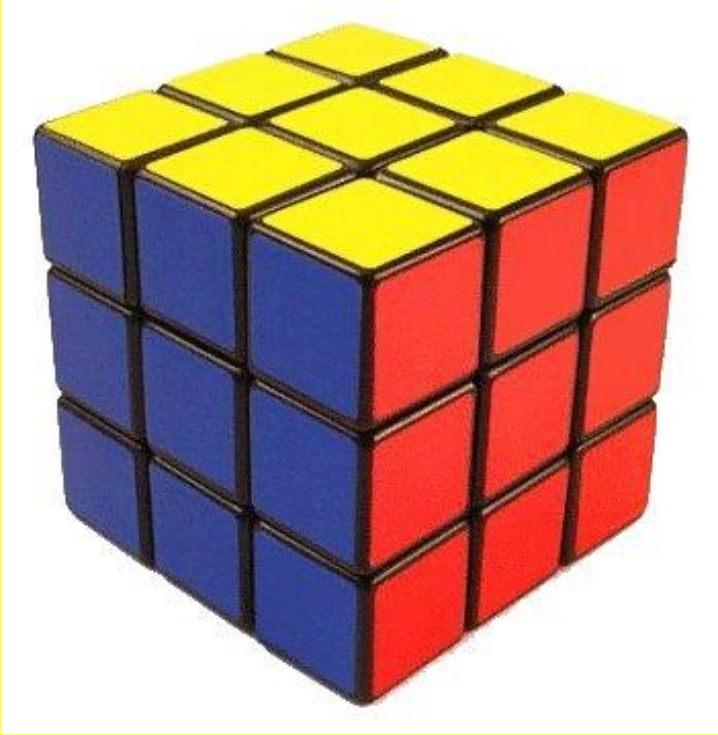

START
Wczytaj n
(ilość śmieci)
i = 0
N
n = 0
T
i = i + 1
STOP
N
czy i-ty śmieć należy do
zbioru frakcja sucha ?
T
N
czy i-ty śmieć należy
do zbioru szkło ?
T
umieść i-ty śmieć
w koszu żółtym
umieść i-ty śmieć
w koszu niebieskim
umieść i-ty śmieć
w koszu zielonym
n = n - 1

Wersja II


START
i = 1
czy i-ty śmieć należy do zbioru frakcja sucha ?
N
T
czy i-ty śmieć należy do zbioru szkło ?
N
T
umieść i-ty element w koszu żółtym
czy i-ty śmieć należy do zbioru frakcja mokra ?
N
T
umieść i-ty element w koszu zielonym
umieść i-ty element w koszu niebieskim
STOP
i = i + 1

PRZYKŁADY SORTOWANIA
- układanie kostki rubika





PRZYKŁADY SORTOWANIA
- układanie kostki rubika

http://kostkarubika.info/files/kurs_układania_kostki_rubika_offline.pdf

ALGORYTMY SORTOWANIA

Algorytmy Stabilne – tj. takie, w których elementy o równej wartości występują po posortowaniu w tej samej kolejności jaką miały w zbiorze nieposortowanym.

- sortowanie bąbelkowe
- sortowanie przez wstawianie
- sortowanie przez scalanie
- sortowanie przez zliczanie
- sortowanie kubełkowe
- sortowanie pozycyjne
- sortowanie biblioteczne

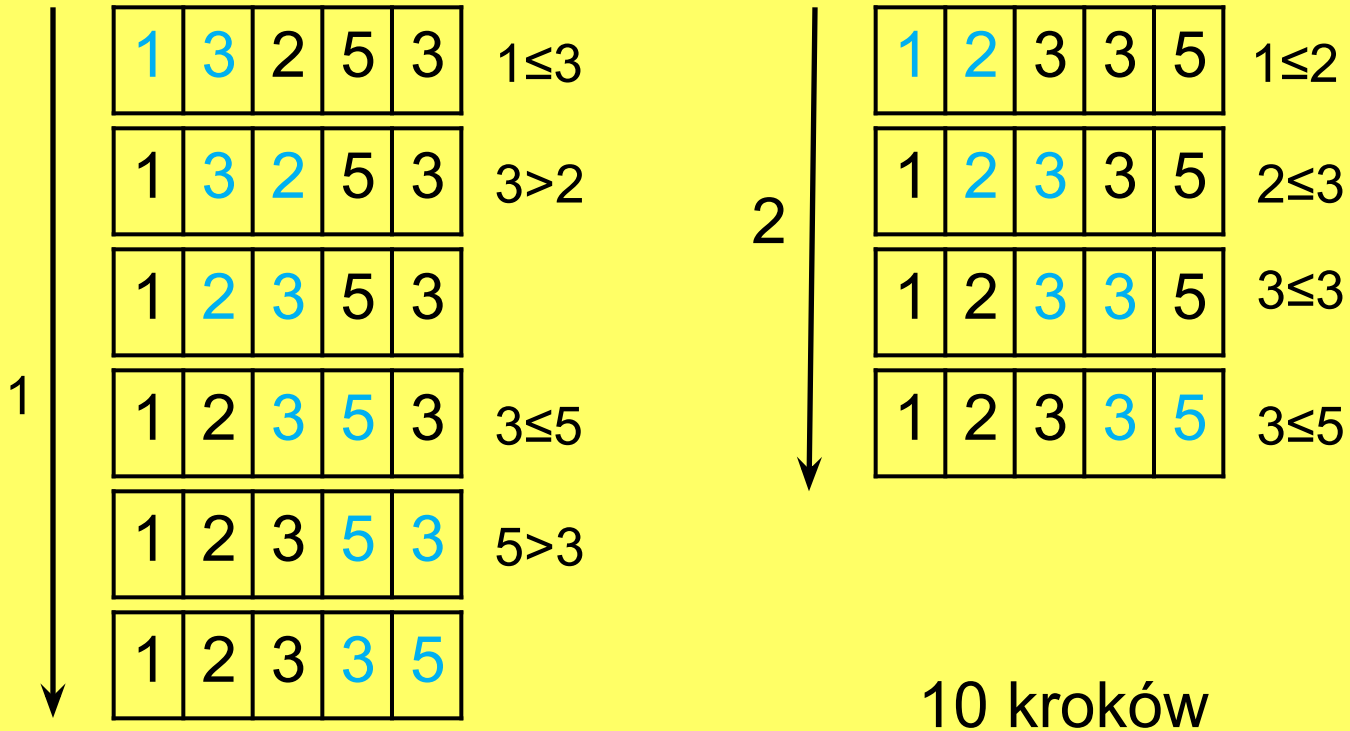
Algorytmy Niestabilne – tj. takie, w których elementy o równej wartości nie występują po posortowaniu w tej samej kolejności jaką miały w zbiorze nieposortowanym.

- sortowanie przez wybieranie
- sortowanie Shella
- sortowanie grzebieniowe
- sortowanie szybkie
- sortowanie introspektywne
- sortowanie przez kopcowanie

SORTOWANIE BĄBELKOWE

Prosta metoda sortowania, polegająca na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim się sortuje dane. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.

Przykład I: Posortować rosnąco ciąg liczb: 1, 3, 2, 5, 3



10 kroków

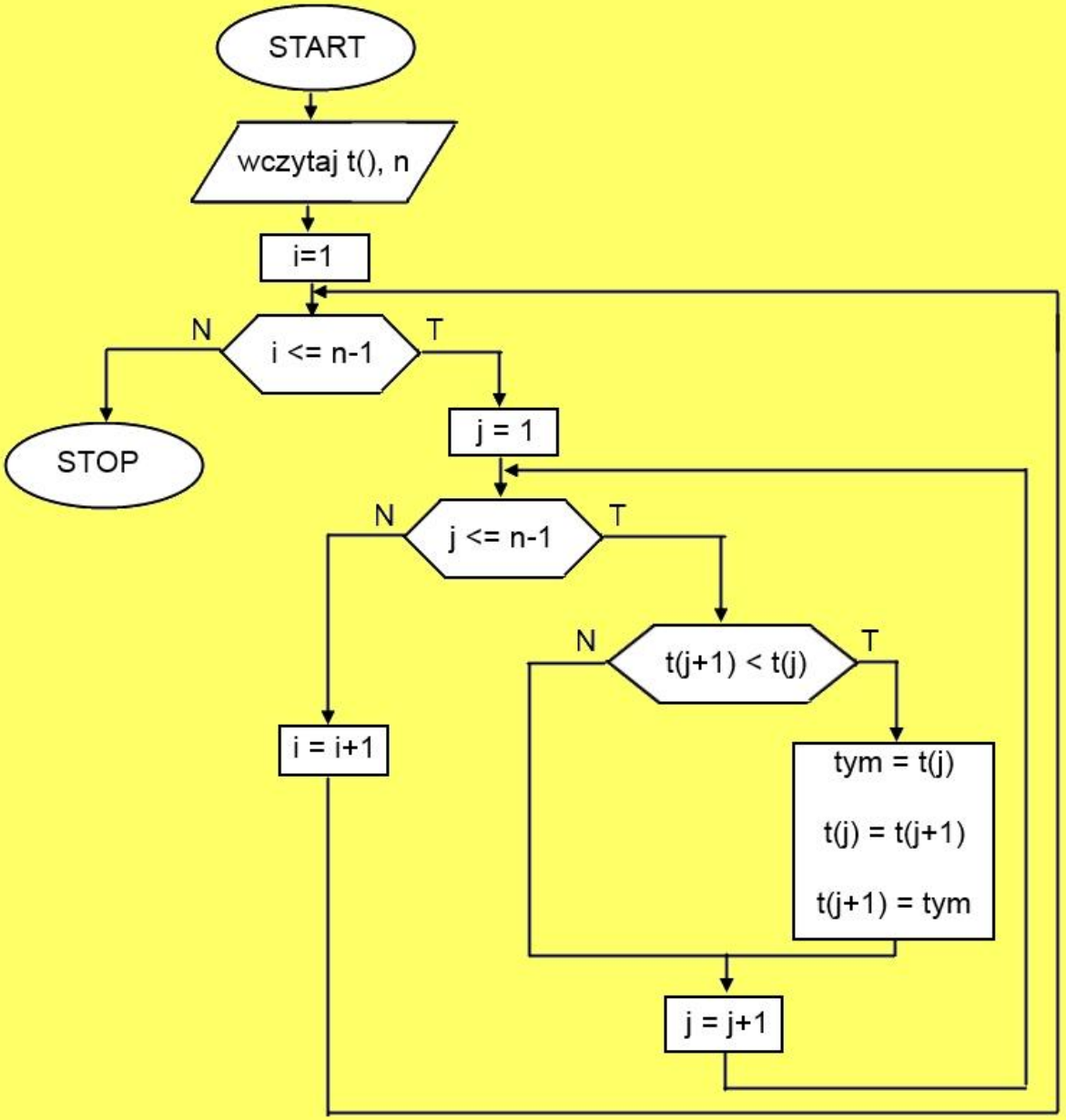
SORTOWANIE BĄBELKOWE

Przykład II: Posortować rosnąco ciąg liczb: 3, 2, 5, 1, 6

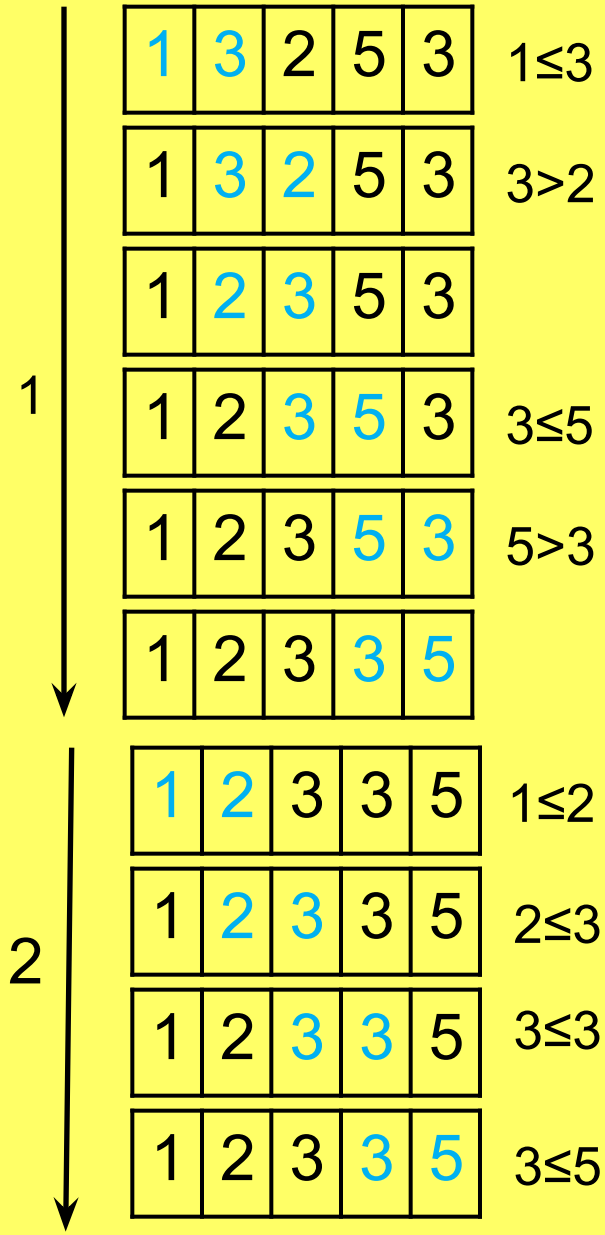
Każda tablica symbolizuje wypchnięcie kolejnego największego elementu na koniec ("wypłynięcie największego bąbelka"). Czerwonym kolorem oznaczono elementy posortowane.

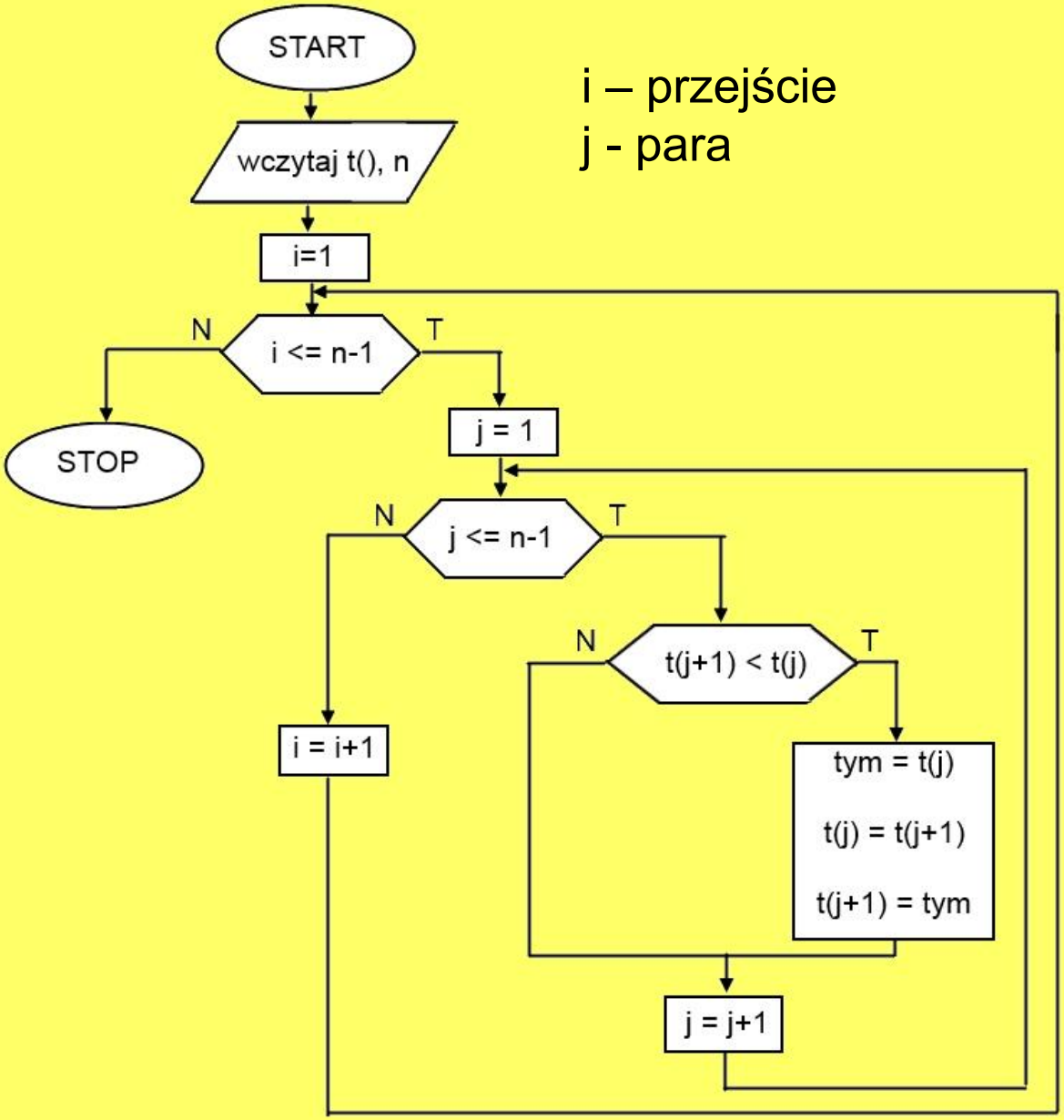
20 kroków

SORTOWANIE BĄBELKOWE



SORTOWANIE BĄBELKOWE





i – przejście
j - para

SORTOWANIE PRZEZ WSTAWIANIE

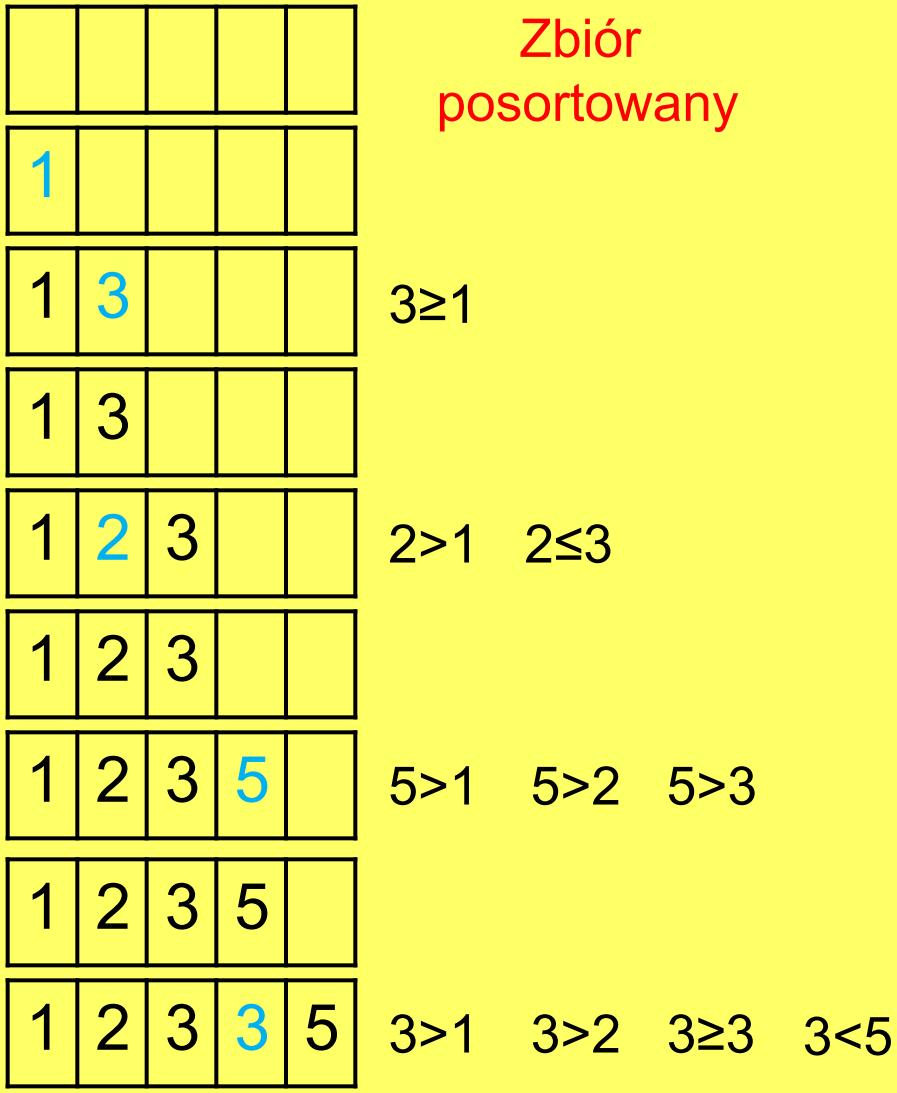
Jeden z najprostszych algorytmów sortowania, odzwierciedlający sposób w jaki ludzie ustawiają karty, tj. kolejne elementy wejściowe są ustawiane na odpowiednie miejsca docelowe. Jest efektywny dla niewielkiej liczby elementów oraz dla danych wstępnie posortowanych.

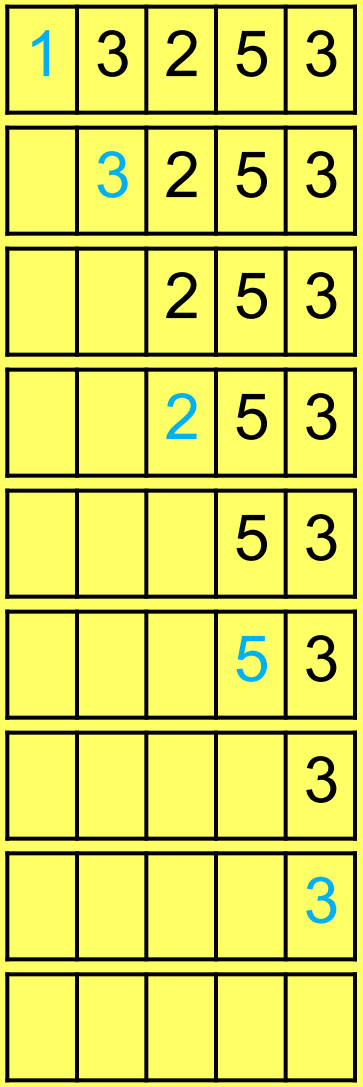
Schemat działania algorytmu

1. Utwórz zbiór elementów posortowanych i przenieś do niego dowolny element ze zbioru nieposortowanego.
2. Weź dowolny element ze zbioru nieposortowanego.
3. Wyciągnięty element porównuj z kolejnymi elementami zbioru posortowanego póki nie napotkasz elementu równego lub elementu większego (jeśli chcemy otrzymać ciąg niemalejący) lub nie znajdziemy się na początku/końcu zbioru uporządkowanego.
4. Wyciągnięty element wstaw w miejsce gdzie skończyłeś porównywać.
5. Jeśli zbiór elementów nieuporządkowanych jest niepusty wróć do punkt 2.

SORTOWANIE PRZEZ WSTAWIANIE

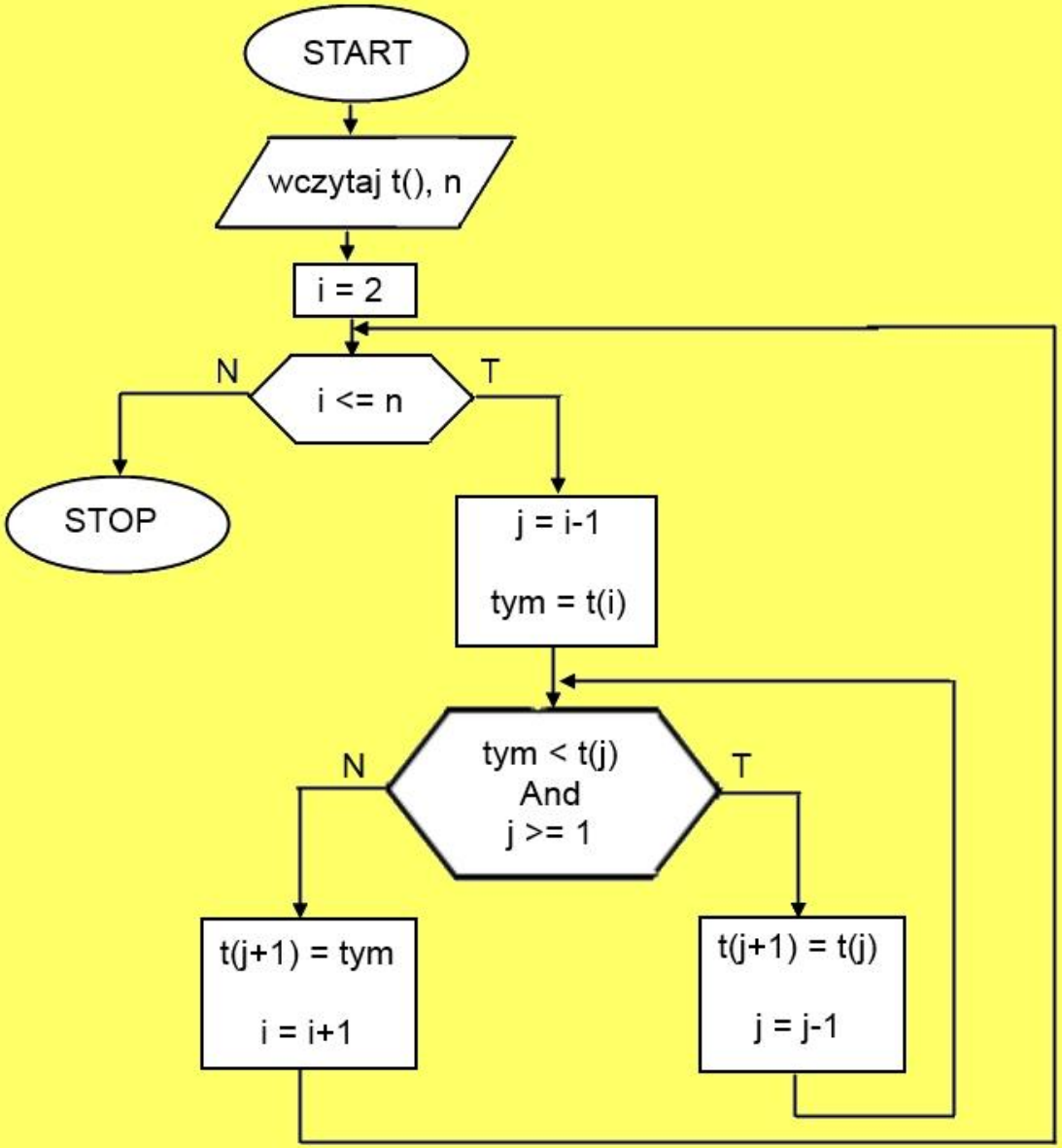
Przykład I: Posortować rosnąco ciąg liczb: 1, 3, 2, 5, 3





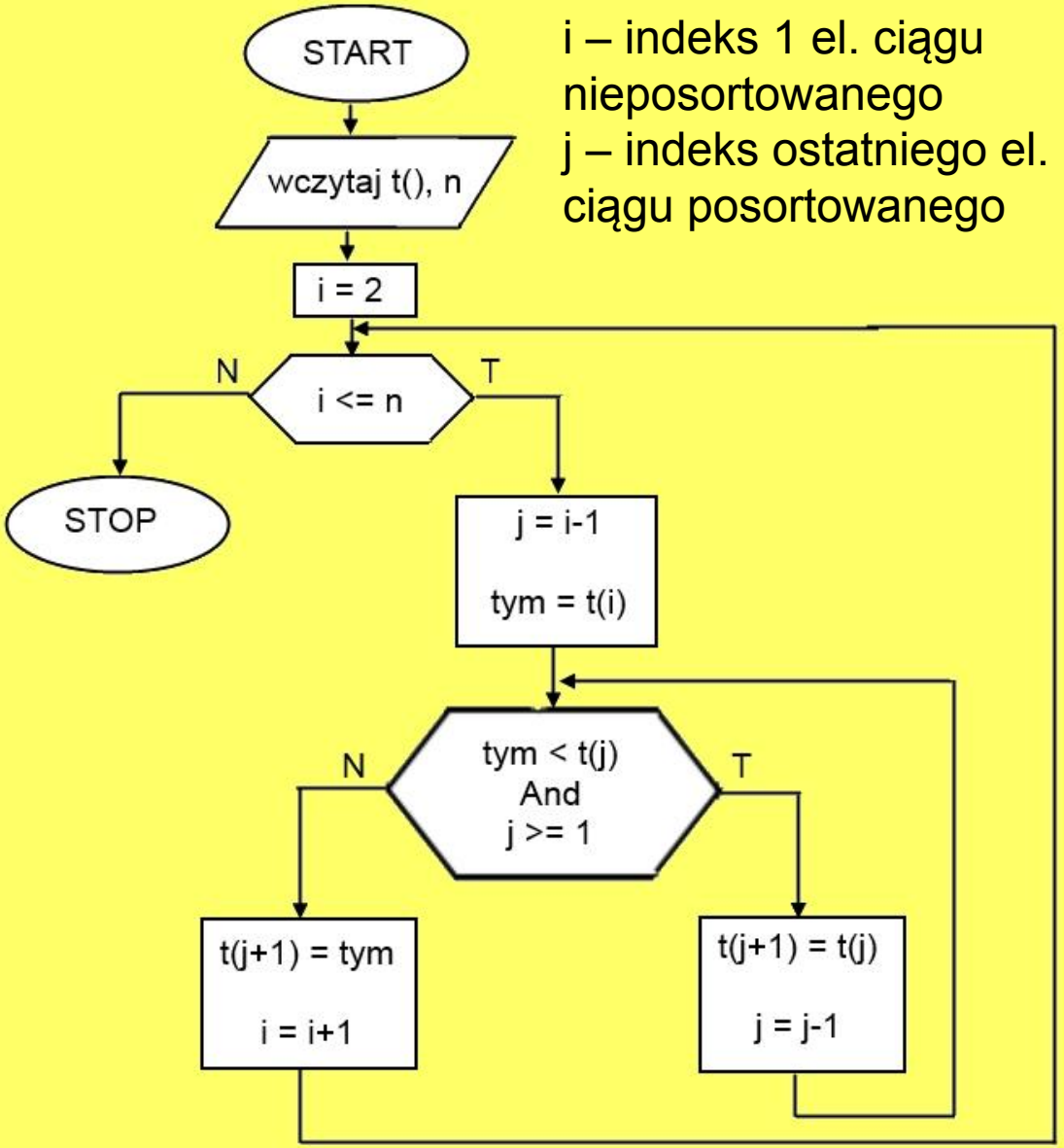
11 kroków

SORTOWANIE PRZEZ WSTAWIANIE



SORTOWANIE PRZEZ WSTAWIANIE

	1 3 2 5 3
1	3 2 5 3
1 3	2 5 3
1 3	2 5 3
1 2 3	5 3
1 2 3	5 3
1 2 3 5	3
1 2 3 5	3
1 2 3 3 5	



i – indeks 1 el. ciągu nieposortowanego
j – indeks ostatniego el. ciągu posortowanego

SORTOWANIE PRZEZ WYBIERANIE

Metoda sortowania polegająca na wyszukaniu elementu mającego się znaleźć na zadanej pozycji i zamianie miejscami z tym, który jest tam obecnie. Operacja jest wykonywana dla wszystkich indeksów sortowanej tablicy.

Schemat działania algorytmu

1. Wyszukaj minimalną wartość ze zbioru danych wejściowych spośród elementów od i+1 do końca zbioru.
2. Zamień wartość minimalną, z elementem na pozycji i. Gdy zamiast wartości minimalnej wybierana będzie maksymalna, wówczas dane wejściowe będą posortowane od największego do najmniejszego elementu.

UWAGA:

Algorytm można przyspieszyć, gdy zbiór danych jest porządkowany jednocześnie z obu końców, tj. wyszukiwane jest równocześnie minimum i maksimum.

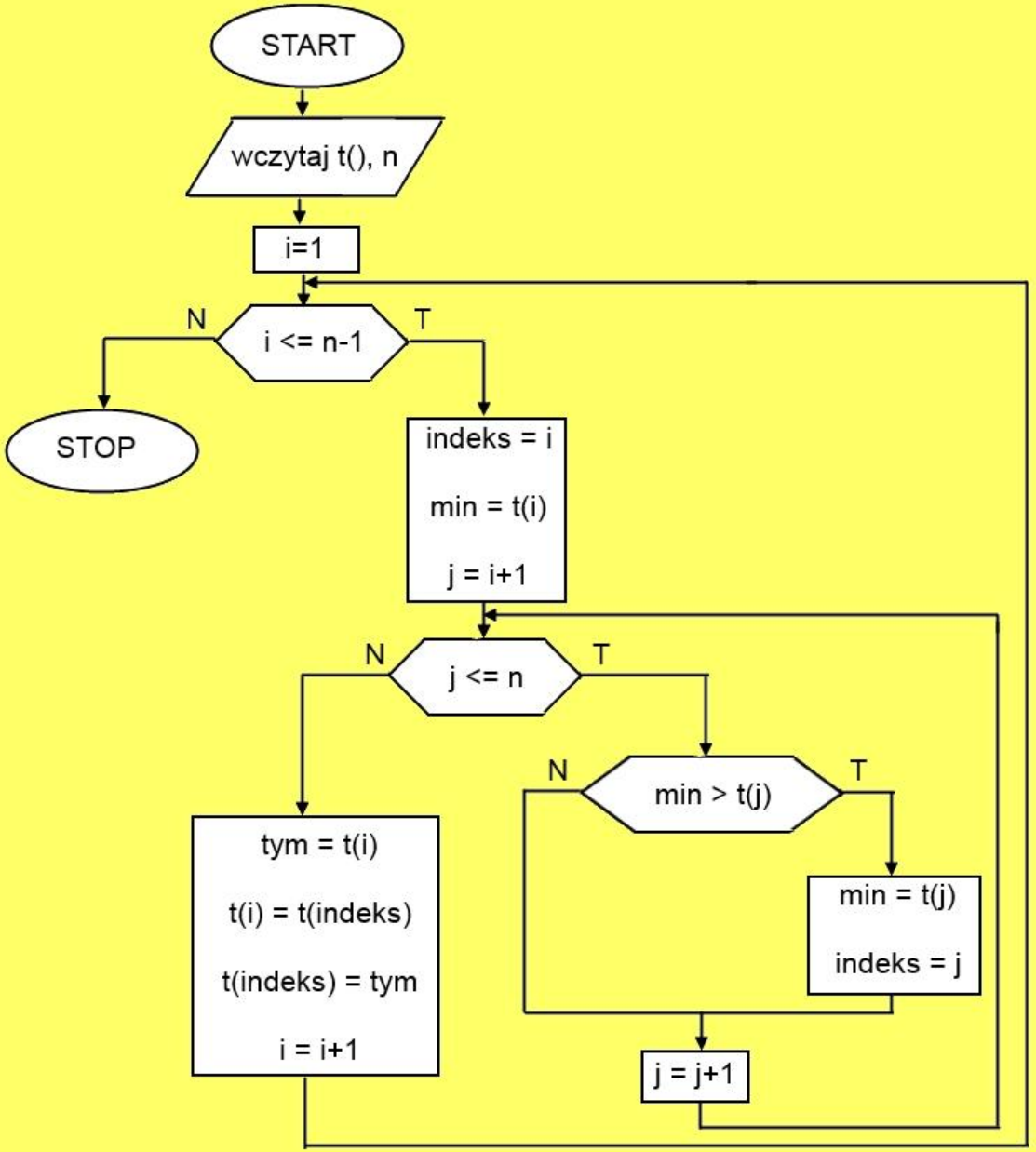
SORTOWANIE PRZEZ WYBIERANIE

Przykład I: Posortować rosnąco ciąg liczb: 1, 3, 2, 5, 3

Numer iteracji i	dane	minimum
0	**1, 3, 2, 5, 3**	1
1	1, **2, 3, 5, 3**	2
2	1, 2, **3, 3, 5**	3
3	1, 2, 3, **3, 5**	3

W tablicy pogrubiono te elementy, wśród których wyszukuje się wartość minimalną, kolorem czerwonym natomiast zaznaczono element znaleziony w wyniku sortowania w danej iteracji.

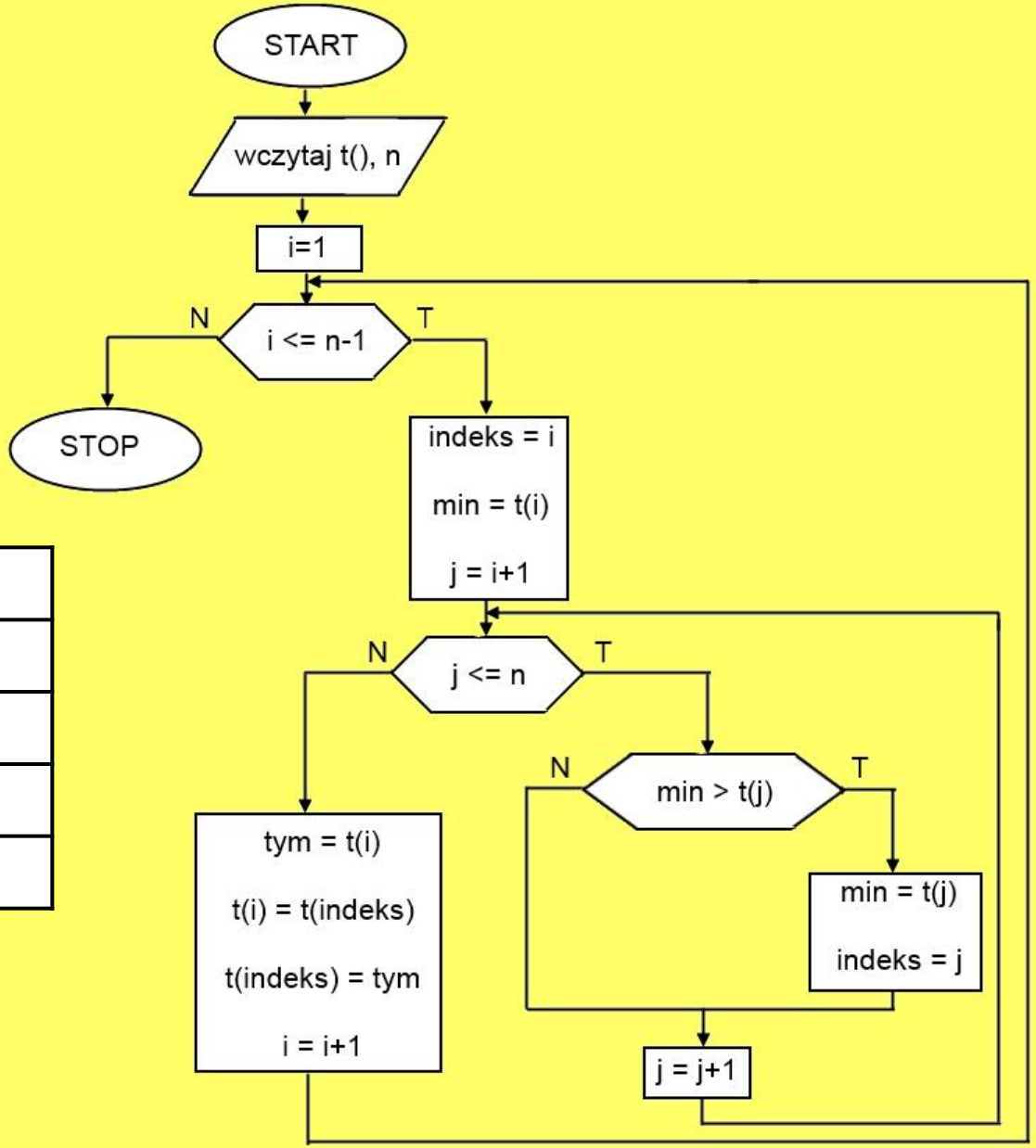
SORTOWANIE PRZEZ WYBIERANIE



SORTOWANIE PRZEZ WYBIERANIE

i – numer iteracji,
j – indeks kolejnego porównywanego el.

Numer iteracji i	dane
1	**1, 3, 2, 5, 3**
2	1, **2, 3, 5, 3**
3	1, 2, **3, 3, 5**
4	1, 2, 3, **3, 5**



OCENA EFEKTYWNOŚCI ALGORYTMÓW

ZŁOŻONOŚĆ ALGORYTMÓW

Jest to suma zasobów niezbędnych do wykonania danego algorytmu. Pod pojęciem zasobów rozumie się takie wielkości jak czas wykonania algorytmu (tzw. złożoność czasowa), pamięć (złożoność pamięciowa) lub liczba procesorów. Na ogół ilość potrzebnych zasobów zależy od liczby i struktury danych wejściowych koniecznych do rozwiązania danego zagadnienia. W informatyce złożoność czasowa ze względu na zwoje znaczenie jest znacznie częściej analizowana niż złożoność pamięciowa.

ZŁOŻONOŚĆ CZASOWA I PAMIĘCIOWA ALGORYTMÓW

Złożoność czasowa

Określa jak długo rozważany algorytm musi pracować w celu rozwiązania danego problemu. Czas pracy algorytmu nie jest wyrażany w sekundach, lecz w jednostkach, którym nie jest przypisana żadna rzeczywista miara, a jej wartość zależy od liczby danych wejściowych oraz zasad opisujących sposób ich przetwarzania przez dany algorytm. Złożoność czasową algorytmów oznacza się dużą literą **O** (notacja Landaua).

Złożoność pamięciowa

Określa zapotrzebowanie na zasoby pamięci przez dany algorytm na podstawie liczby danych wejściowych, przekazanych do algorytmu oraz sposobu ich przetwarzania przez dany algorytm. W przypadku szacowania złożoności pamięciowej rozpatruje się tylko i wyłącznie pamięć, którą należy dodatkowo dodać w trakcie pracy algorytmu tak, aby było możliwe jego wykonanie, co oznacza że do złożoności pamięciowej nie wlicza się rozmiaru danych wejściowych. Miarą złożoności pamięciowej jest zatem dodatkowo dodana liczba pamięci RAM (wyrażana w bajtach).

PORÓWNYWANIE ZŁOŻONOŚCI ALGORYTMÓW

W celu porównania złożoności algorytmów analizowane jest tzw. asymptotyczne tempo wzrostu, czyli zachowanie się funkcji określającej złożoność dla dużych ilości danych wejściowych. Ponadto, złożoności algorytmów różniące się o stałą traktowane są za takie same, co eliminuje m.in. wpływ szybkości działania komputera, na którym dany algorytm jest wykonywany.

Problemy, do rozwiązania których potrzebna jest podobna ilość zasobów łączone są w tzw. klasy złożoności. Na przykład, liniowa złożoność czasowa (pamięciowa), oznacza że czas (zasób dodanej pamięci) konieczny do rozwiązania problemu przez algorytm rośnie liniowo względem rozmiaru danych wejściowych; natomiast kwadratowa złożoność oznacza zależność proporcjonalną do kwadratu rozmiaru danych.

KLASY ZŁOŻONOŚCI ALGORYTMÓW

Złożoność stała - O(1)

Algorytm wykonuje stałą ilość operacji dominujących bez względu na rozmiar danych wejściowych.

Złożoność liniowa - O(n)

Dla każdej danej algorytm wykonuje stałą ilość operacji dominujących. Czas wykonania jest proporcjonalny do liczby n danych wejściowych.

Złożoność kwadratowa - $O(n^2)$

Algorytm dla każdej danej wykonuje ilość operacji dominujących proporcjonalną do liczby wszystkich przetwarzanych danych. Czas wykonania jest proporcjonalny do kwadratu liczby Inne złożoności tego typu $O(n^3)$, $O(n^4)$... noszą nazwę wielomianowych złożoności obliczeniowych.

Złożoność logarytmiczna - $O(\log_2 n)$

W algorytmie zadanie rozmiaru n da się sprowadzić do zadania rozmiaru n/2.

Złożoność liniowo logarytmiczna - $O(n \log_2 n)$

Zadanie rozmiaru n daje się sprowadzić do dwóch podzadań rozmiaru n/2 plus pewna ilość operacji, których liczba jest proporcjonalna do ilości danych n. Tego typu złożoność obliczeniową posiadają dobre algorytmy sortujące.

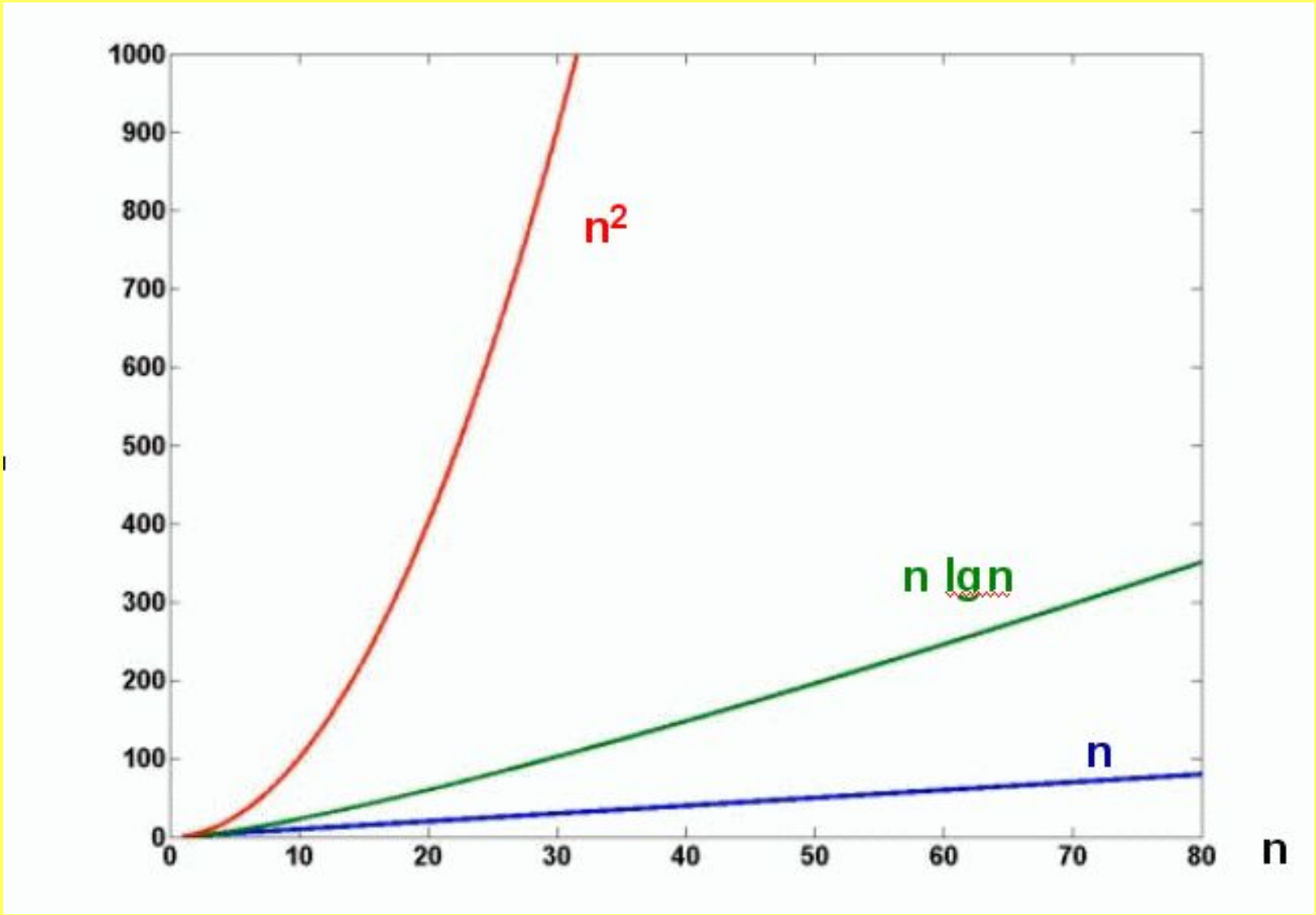
Złożoność wykładnicza - $O(2^n)$, $O(n!)$

Złożoność obliczeniową $O(2^n)$ posiada algorytm, w którym wykonywana jest stała liczba operacji dla każdego podzbioru n danych wejściowych.

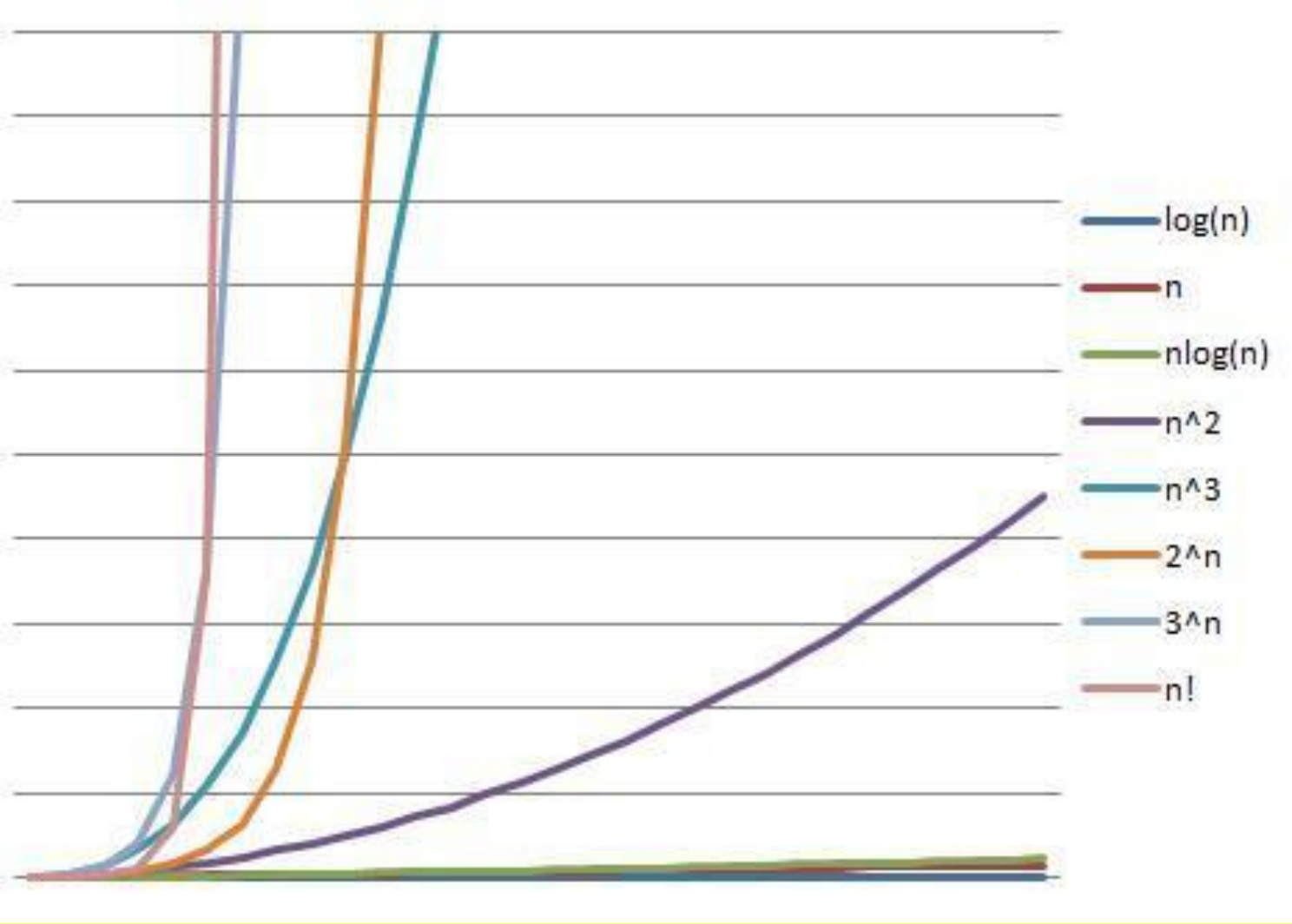
PORÓWNANIE KLAS ZŁOŻONOŚCI ALGORYTMÓW

Porównanie klas złożoności obliczeniowych		
Klasa złożoności obliczeniowej O()	**Nazwa klasy złożoności obliczeniowej**	**Cechy algorytmu**
$O(1)$	Stała	działa prawie natychmiast
$O(\log_2 n)$	Logarytmiczna	niesamowicie szybki
$O(n)$	liniowa	szybki
$O(n\log_2 n)$	liniowo-logarytmiczna	dosyć szybki
$O(n^2)$	kwadratowa	wolny dla dużych n
$O(n^3)$	sześcienna	wolny dla większych n
$O(2^n)$, $O(n!)$	wykładnicza	nierealizowalny dla większych n

ZŁOŻONOŚĆ CZASOWA ALGORYTMÓW



ZŁOŻONOŚĆ CZASOWA ALGORYTMÓW



ALGORYTMY SORTOWANIA

Algorytmy Stabilne

- sortowanie bąbelkowe – O(n^2)
- sortowanie przez wstawianie – O(n^2)
- sortowanie przez scalanie – O(n\log n), wymaga O(n) dodatkowej pamięci
- sortowanie przez zliczanie – O(n+k), wymaga O(n+k) dodatkowej pamięci
- sortowanie kubełkowe – O(n), wymaga O(k) dodatkowej pamięci
- sortowanie pozycyjne – O(d (n+k)), gdzie k to wielkość domeny cyfr, a d szerokość kluczy w cyfrach. Wymaga O(n+k) dodatkowej pamięci
- sortowanie biblioteczne – O(n \log n), pesymistyczny O(n^2)

Algorytmy Niestabilne

- sortowanie przez wybieranie O(n^2) – może być stabilne po odpowiednich zmianach
- sortowanie Shella – złożoność nieznana;
- sortowanie grzebieniowe – złożoność nieznana;
- sortowanie szybkie – Θ(n \log n), pesymistyczny O(n^2); z wykorzystaniem algorytmu selekcji "mediana median" ("magicznych piątek") do wyszukiwania mediany, optymistyczna złożoność to O(n \log n),
- sortowanie introspektywne – O(n \log n);
- sortowanie przez kopcowanie – O(n \log n);

UWAGI KOŃCOWE

1. Najszybsze algorytmy sortujące to algorytmy sortowania dystrybucyjnego. Szybkość ich działania jest okupiona dużym zapotrzebowaniem na pamięć. Algorytmy te mają liniową klasę czasowej złożoności obliczeniowej. W typowych warunkach polecanym, szybkim algorytmem sortującym jest algorytm sortowania szybkiego. Posiada liniowo logarytmiczną klasę czasowej złożoności obliczeniowej, ale dla niekorzystnych danych może się degradować do klasy kwadratowej.
2. Algorytm sortowania przez wstawianie może być również polecany do stosowania z powodu swej prostoty w implementacji i jednocześnie wystarczająco dużej szybkości. Dla zbiorów w znacznym stopniu uporządkowanych wykazuje liniową klasę złożoności obliczeniowej. Dlatego nadaje się np. do sortowania zbioru uporządkowanego, do którego dodajemy nowy element - zbiór będzie posortowany szybciej niż przez algorytm sortowania szybkiego.
3. Nie ma uniwersalnych algorytmów sortujących.
4. Algorytmów sortowania bąbelkowego raczej należy unikać.

KONIEC