

TEORETYCZNE PODSTAWY INFORMATYKI

22/01/2018

WFAiS UJ, Informatyka Stosowana
II stopień studiów

Wykład 14a: Repetytorium

2

Algorytmy i
ich schematy
blokowe

- ▣ **Proste algorytmy iteracyjne**
- ▣ **Algorytmy z wykorzystaniem rekurencji**
- ▣ **Algorytmy sortujące**

Wykład na podstawie skryptu:

D. Nyk, „Algorytmy w przykładach”

http://informatyka.2ap.pl/ftp/3d/algorytmy/podrecznik_algorytmy.pdf

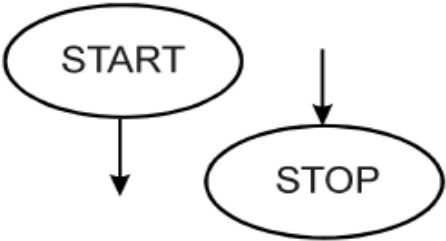

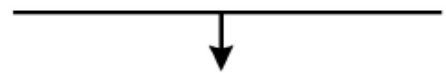
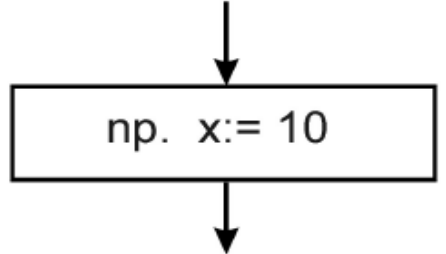
Schemat blokowy

3

- **Przedstawia algorytm w postaci symboli graficznych, podając szczegółowo wszystkie operacje arytmetyczne, logiczne, przesyłania, pomocnicze wraz z kolejnością ich wykonywania**
- **Składa się z wielu elementów z których podstawowy jest blok**
- **Ponizej przedstawione typowe podstawowe bloki programów, istnieją oczywiście jeszcze inne.**

Schemat blokowy

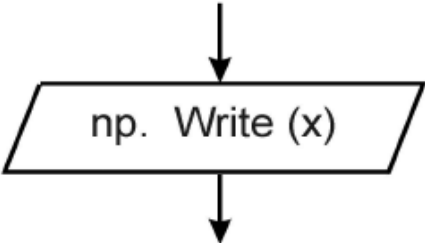
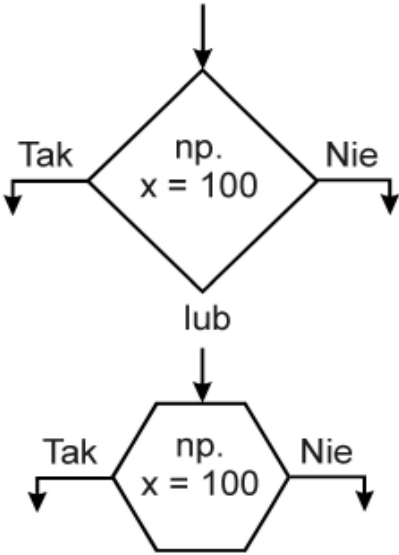
4

Wygląd bloku	Opis
	<p>Bloki graniczne – początek i koniec algorytmu. Mają kształt owalu. Z bloku Start wychodzi tylko jedno połączenie; każdy schemat blokowy musi mieć dokładnie jeden blok START. Każdy schemat blokowy musi mieć co najmniej jeden blok STOP.</p>
	<p>Łącznik pomiędzy blokami – określa kierunek przepływu danych lub kolejność wykonywanych działań (ścieżka sterująca)</p>
	<p>Blok kolekcyjny – łączy kilka różnych dróg algorytmu</p>
	<p>Blok operacyjny – zawiera operację lub grupę operacji, w których wyniku ulega zmianie wartość zmiennej (tu : nadanie zmiennej x wartości 10). Bloki operacyjne mają kształt prostokąta , wchodzi do niego jedno połączenie i wychodzi też jedno.</p>

y

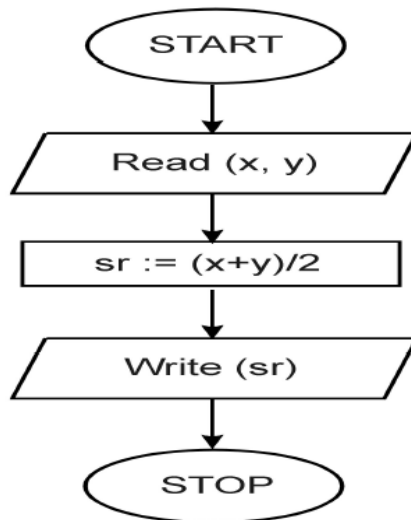
Schemat blokowy

5

	<p>Blok wejścia / wyjścia – blok odpowiedzialny za wykonanie operacji wprowadzania i wyprowadzania danych, wyników, komunikatów. Ma kształt równoległoboku, wchodzi i wychodzi z niego jedno połączenie.</p>
	<p>Blok decyzyjny – określa wybór jednej z dwóch możliwych dróg działania. Ma kształt rombu lub sześciokąta. Wchodzi do niego jedno połączenie, a wychodzą dwa : TAK – gdy warunek wpisany wewnątrz jest spełniony oraz NIE – gdy warunek wpisany wewnątrz nie jest spełniony. Wybór kształtu bloku zależy od nas.</p>

Schemat blokowy i specyfikacja programu

6



Algorytm liczenia średniej

Specyfikacją problemu algorytmicznego nazywamy dokładny opis problemu algorytmicznego, który ma zostać rozwiązany oraz podanie informacji o danych wejściowych i wyjściowych.

Czyli przed naszym algorytmem powinien znaleźć się dodatkowy zapis :

Problem algorytmiczny : obliczenie średniej arytmetycznej dwóch liczb rzeczywistych

Dane wejściowe : $x, y \in \mathbb{R}$

Dane wyjściowe : $sr \in \mathbb{R}$ – średnia liczb x i y

Operandy i operatory

7

- **Stałe i zmienne łączymy operatorami aby otrzymać wyrażenie. Stałe i zmienne nazywamy operandami.**

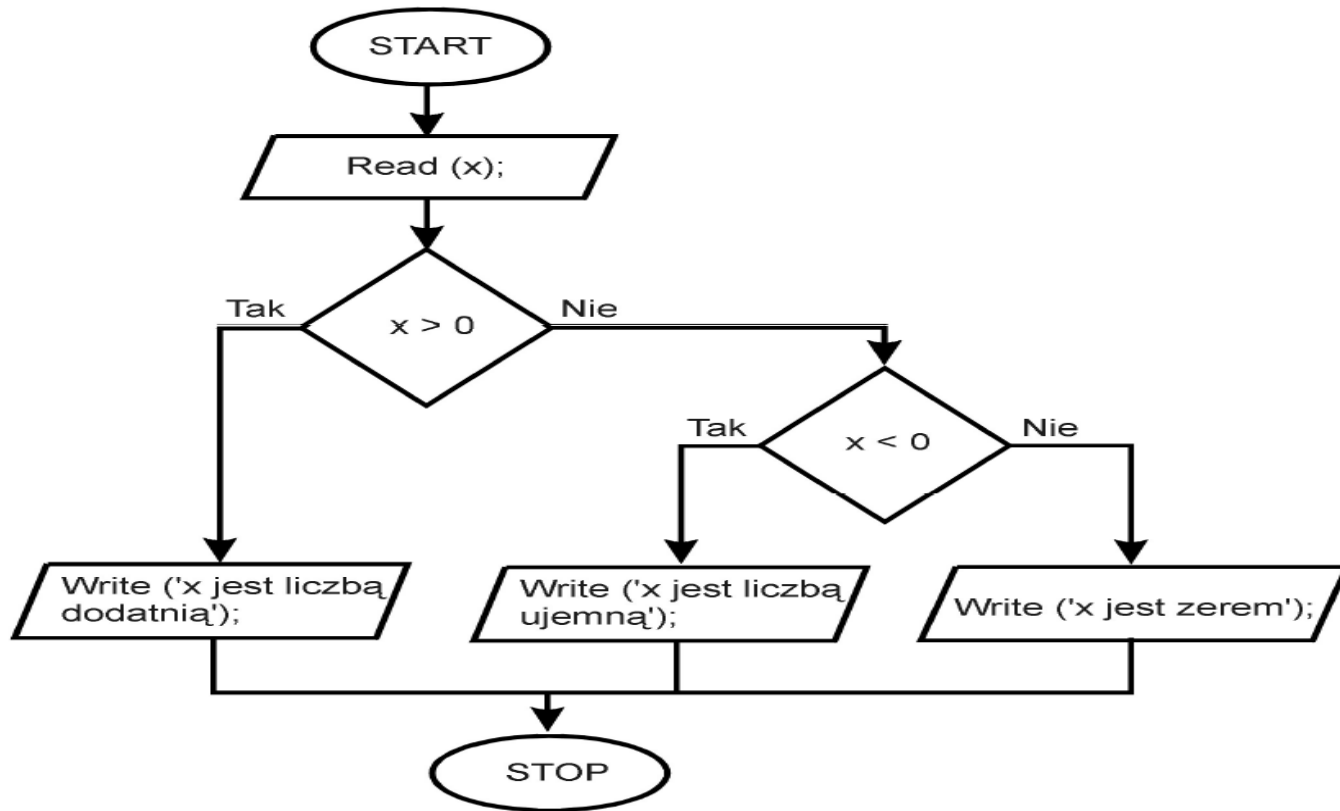
Operatory arytmetyczne		Operatory relacji	
Symbol	Znaczenie	Symbol	Znaczenie
+	dodawanie	=	równy
-	odejmowanie	>	większy
*	mnożenie	>=	większy lub równy
/	dzielenie	<	mniejszy
div	dzielenie całkowite (3div2 = 1)	<=	mniejszy lub równy
mod	reszta z dzielenia liczb całkowitych	<>	różny

Operatory logiczne

and	- koniunkcja (iloczyn zdań)	∧
or	- alternatywa (suma zdań)	∨
not	- negacja (zaprzeczenie zdania)	~

Algorytmy z rozgałęzieniem

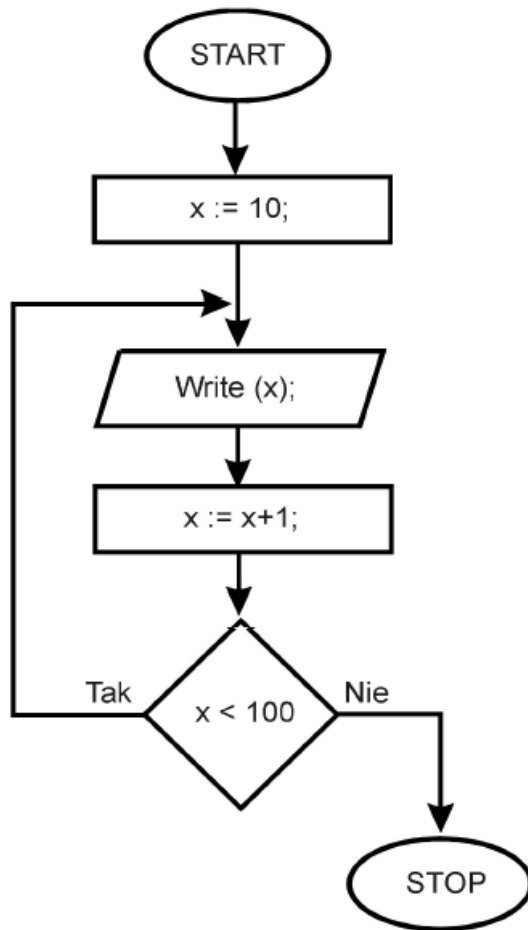
8



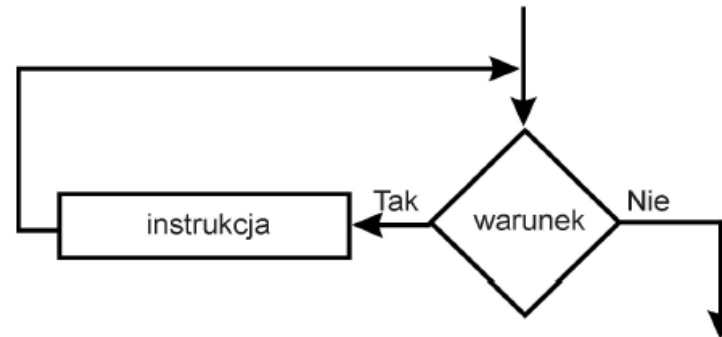
Warunek musi być tak określony, aby jego ocena prawdziwości była jednoznaczna.

Instrukcja iteracji

9



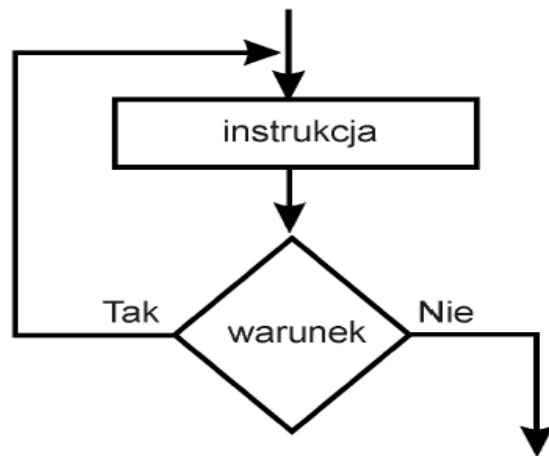
Poniżej przedstawiamy trzy podstawowe przypadki iteracji stosowanych w algorytmach.



Najpierw sprawdzany jest warunek, potem wykonywana jest instrukcja. (dopóki spełniony jest warunek wykonuj instrukcję - w Pascalu : While warunek do instrukcja)

Instrukcja iteracji

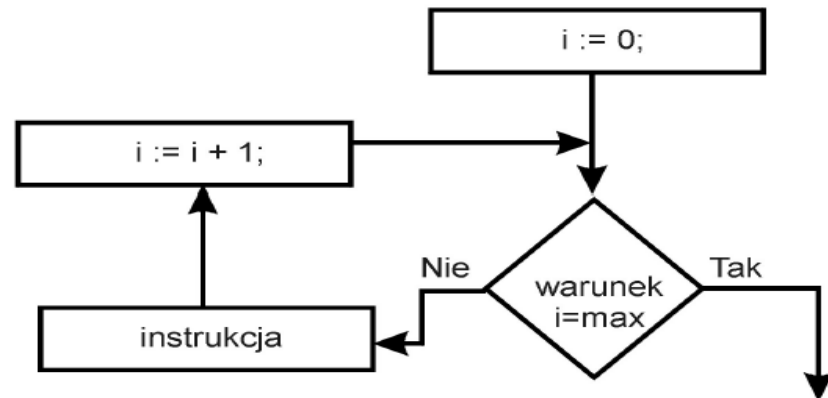
10



Najpierw wykonywana jest instrukcja , a potem sprawdzany jest warunek (wykonuj instrukcję dopóki spełniony jest warunek – w Pascalu : repeat instrukcja until wyrażenie).

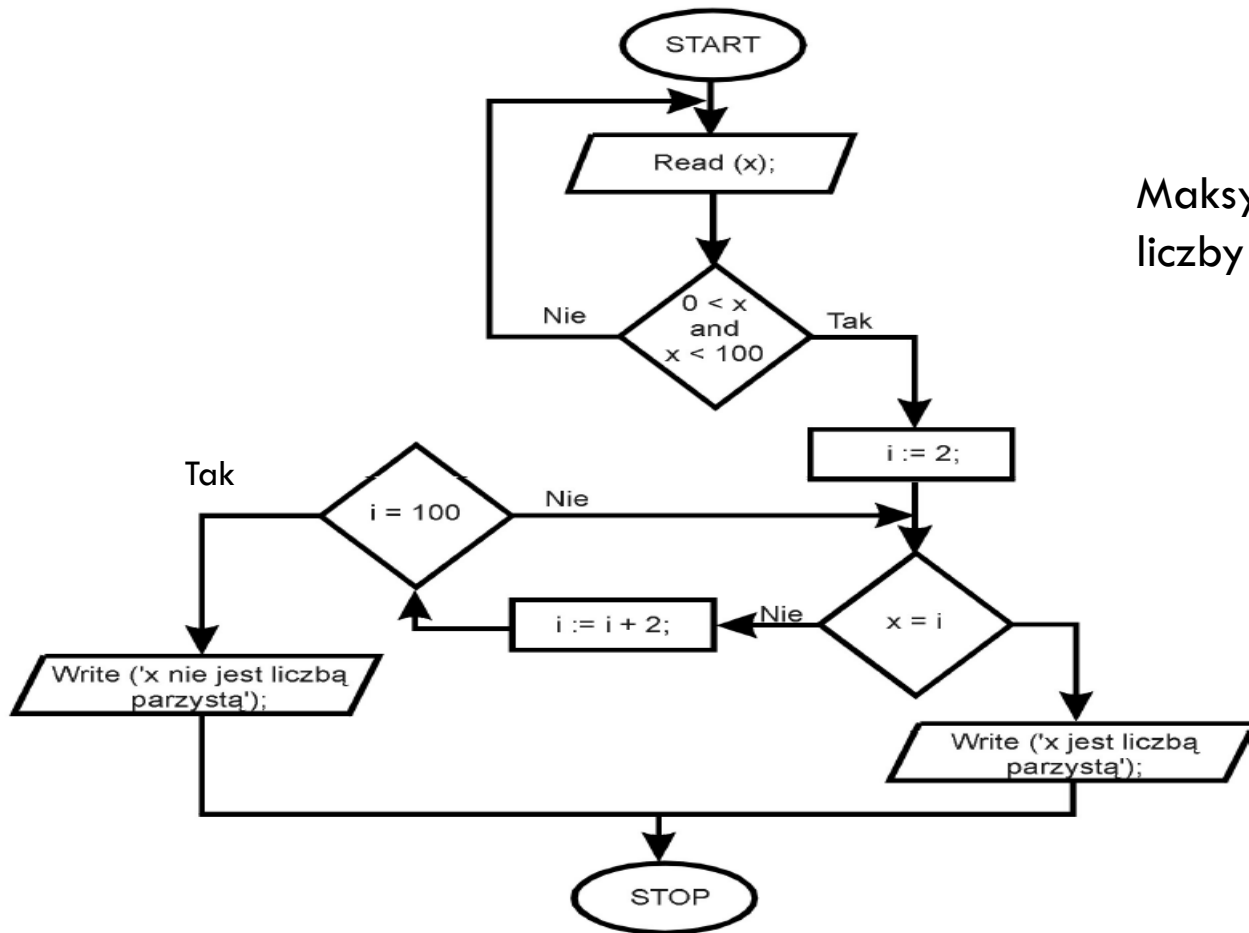
Instrukcja wykonywana jest z góry ustaloną (max) ilością razy (w Pascalu : For zmienna := wyrażenie_1 to max do (downto) instrukcja)

W pętli tej wartość licznika „i” zwiększana jest o 1 po każdej wykonywanej instrukcji czyli jest **inkrementowana**



Badanie parzystości: algorytm 1

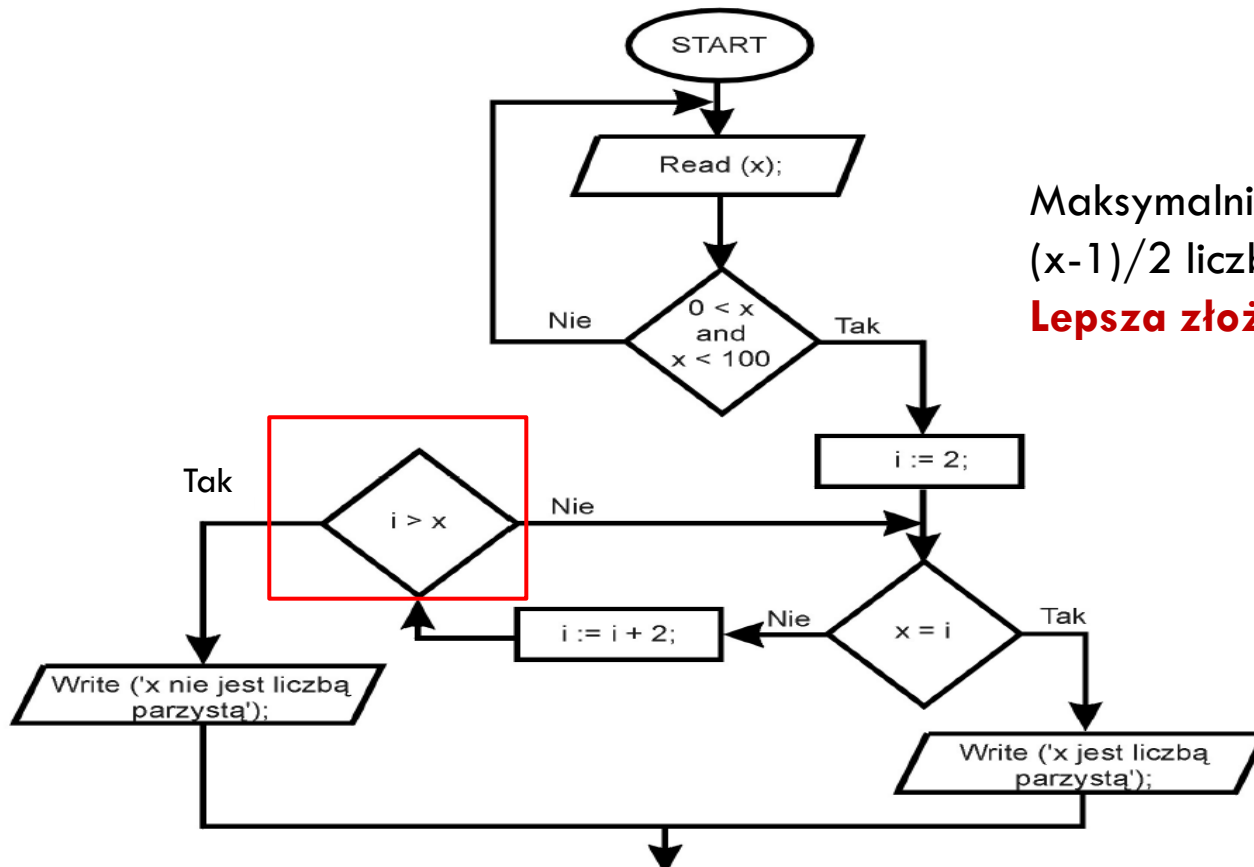
11



Maksymalnie 49 porównań
liczby i z liczbą 100

Badanie parzystości: algorytm 2

12



Maksymalnie porównań
 $(x-1)/2$ liczby i z liczbą x .
Lepsza złożoność obliczeniowa

Algorytm Euklidesa

13

□ Największy wspólny dzielnik dwóch liczb.

Pobieramy dwie liczby naturalne, od większej z nich odejmujemy mniejszą, a następnie większą liczbę zastępujemy różnicą. Postępujemy tak do momentu, gdy dwie liczby będą równe. Otrzymana liczba będzie NWD.

Przykład : $a = 12$, $b = 20$, ponieważ $b > a$ to $b = 20 - 12 = 8$, teraz $a > b$ czyli $a = 12 - 8 = 4$, dalej $b > a$ czyli $b = 8 - 4 = 4$ i $a = 4$ stąd $\text{NWD} = 4$

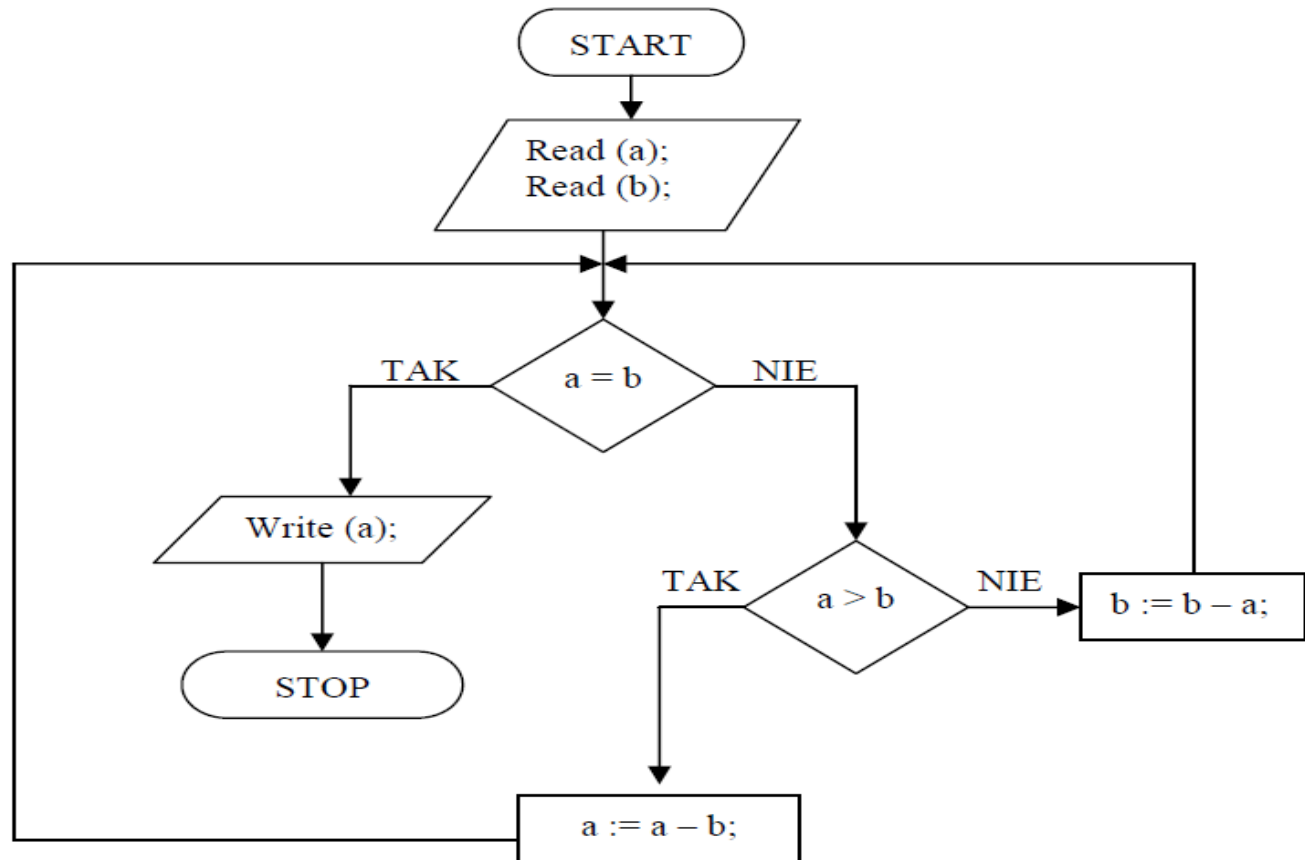
Równie często stosuje się modyfikację algorytmu $\text{NWD}(a,b) = \text{NWD}(a \bmod b, b - (a \bmod b))$

Będziemy iteracyjnie zmieniać wartości a i b aż do momentu, gdy a osiągnie wartość 0.

Przykład : $a = 12$, $b = 20$, $a = 12 \bmod 20 = 12$, $b = 20 - 12 = 8$ następnie $a = 12 \bmod 8 = 4$, $b = 8 - 4 = 4$, i następny krok $a = 4 \bmod 4 = 0$ czyli $b = \text{NWD} = 4$

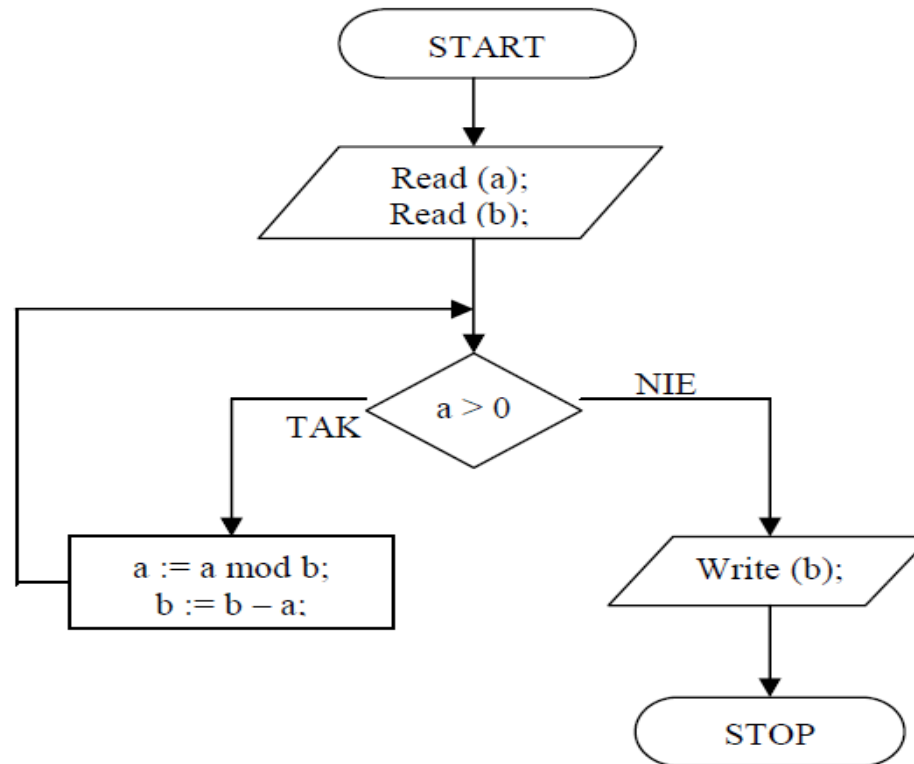
Algorytm Euklidesa: wersja 1

14



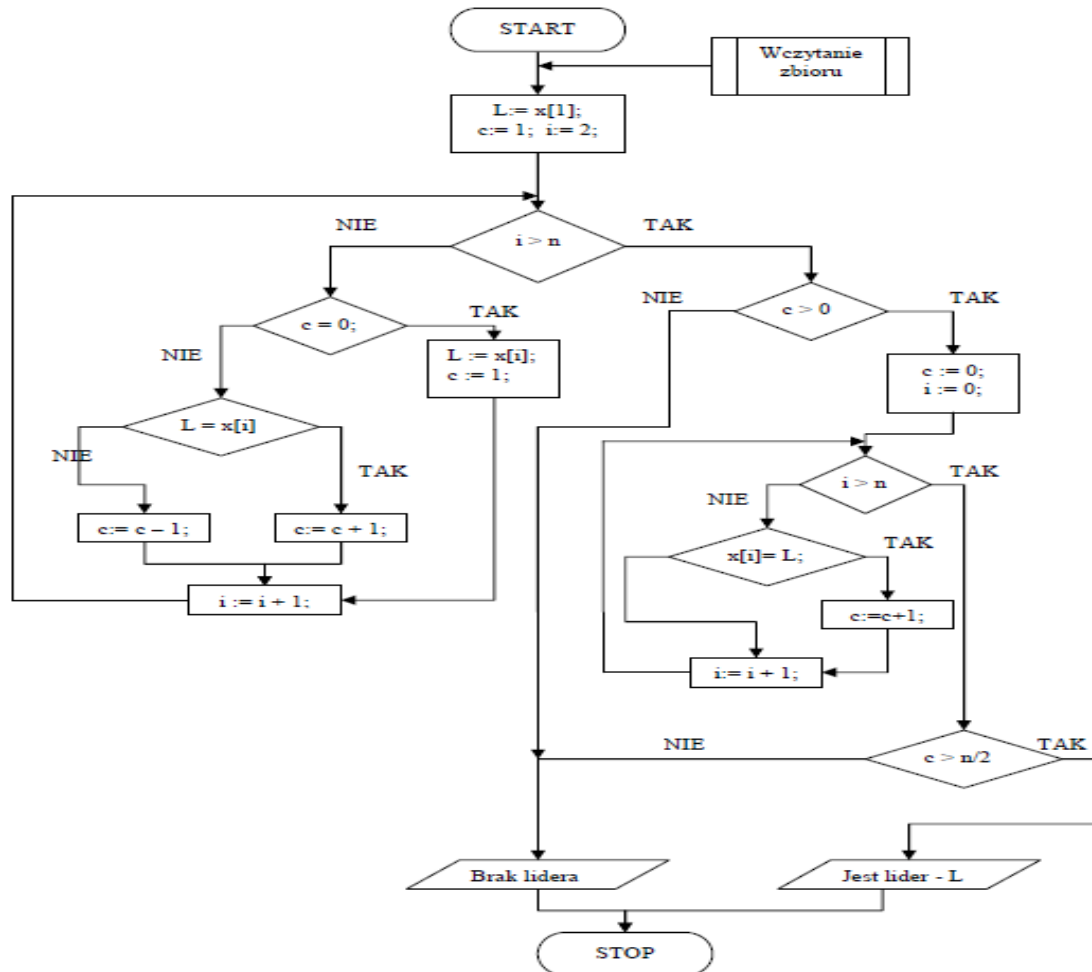
Algorytm Euklidesa: wersja 2

15



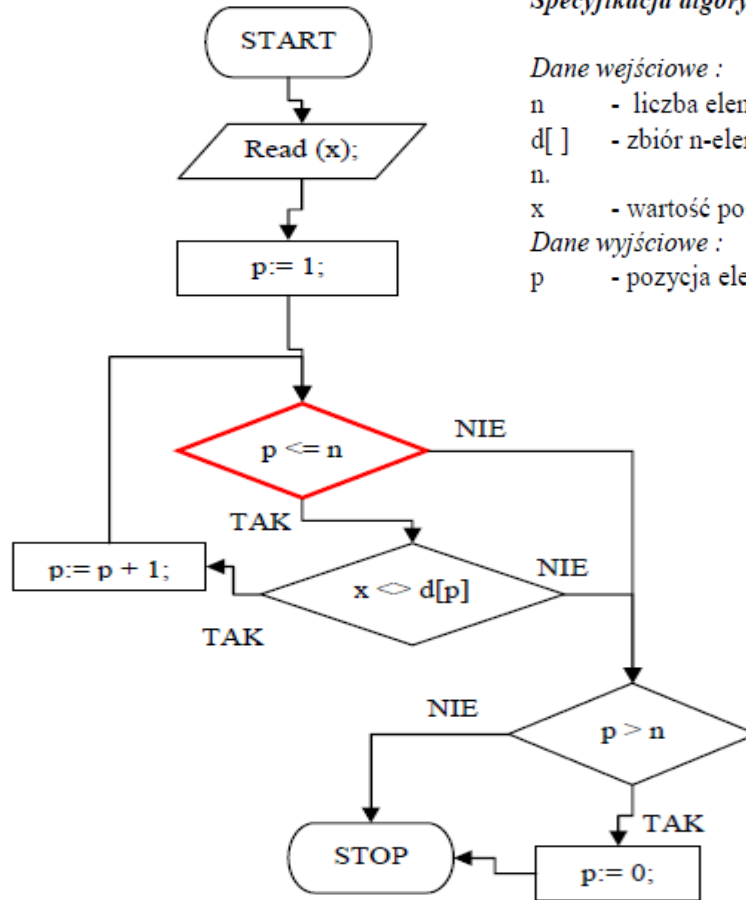
Poszukiwanie lidera zbioru

16



Przeszukiwanie sekwencyjne

17



Specyfikacja algorytmu :

Dane wejściowe :

n - liczba elementów w sortowanym zbiorze, $n \in \mathbb{N}$

$d[]$ - zbiór n -elementowy, który będzie przeszukiwany. Elementy zbioru mają indeksy od 1 do n .

x - wartość poszukiwana

Dane wyjściowe :

p - pozycja elementu x w zbiorze $d[]$. Jeśli $p = 0$, to element x w zbiorze nie występuje.

Warunek gwarantuje zakończenie pętli, możemy też wprowadzić wartownika

Złożoność obliczeniowa $O(n)$

Poszukiwanie najczęstszego elementu występującego w zbiorze

18

Specyfikacja problemu

Dane wejściowe :

- n - liczba elementów w zbiorze wejściowym
- d[] - zbiór wejściowy, w którym dokonujemy poszukiwań. Indeksy elementów rozpoczynają się od 1. Zbiór musi posiadać miejsce na dodatkowy element, który zostanie dopisany na końcu.

Dane wyjściowe :

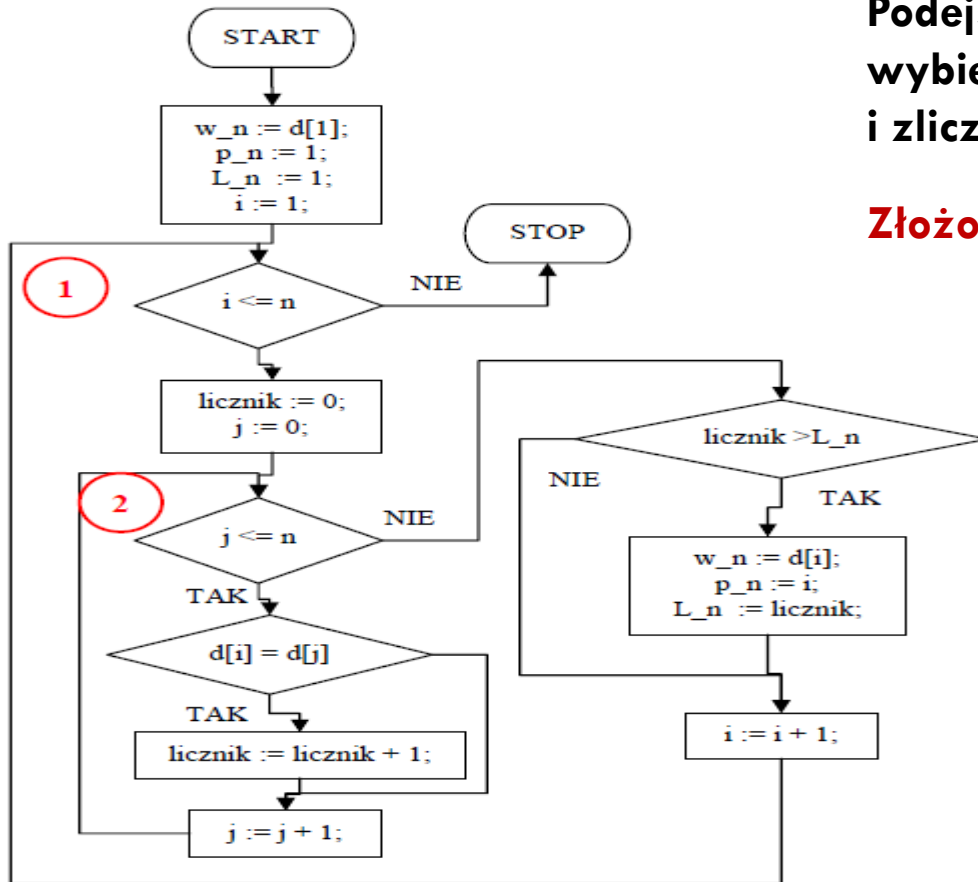
- w_n - wartość elementu powtarzającego się najczęściej
- p_n - pierwsza pozycja elementu najczęstszego
- L_n - liczba wystąpień najczęstszego elementu

Zmienne pomocnicze :

- i,j - zmienne licznikowe pętli
- licznik - licznik wystąpień elementu

Poszukiwanie najczęstszego elementu występującego w zbiorze

19

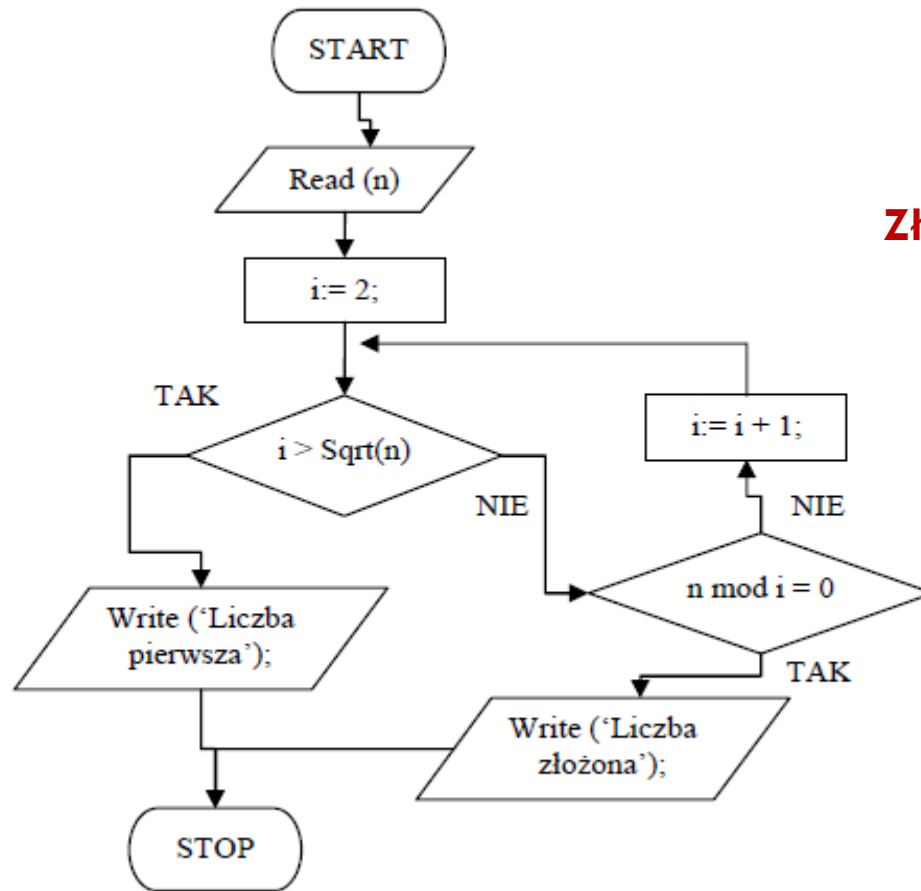


Podejście bezpośrednie:
wybieramy kolejne elementy zbioru
i zliczamy częstość ich występowania.

Złożoność obliczeniowa $O(n^2)$

Algorytm sprawdzający czy liczba jest liczbą pierwszą.

20



Złożoność obliczeniowa $O(n^{1/2})$

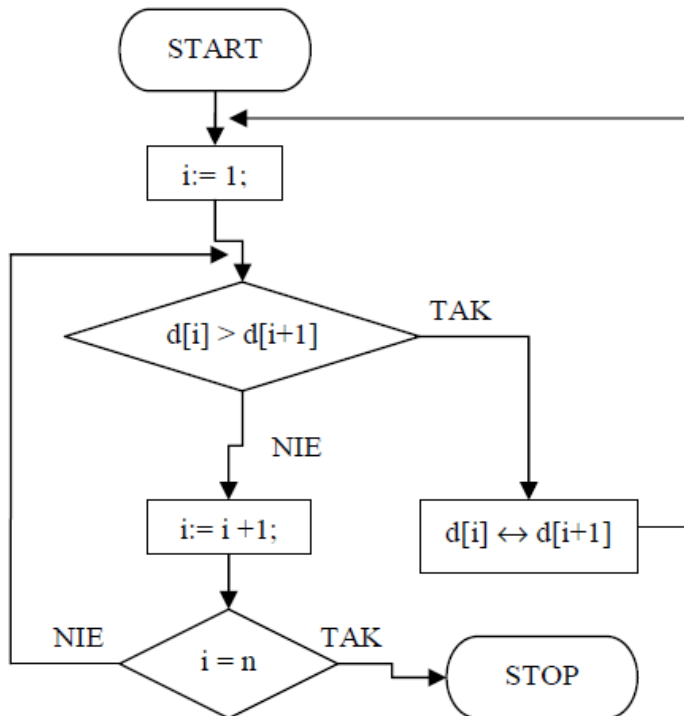
Złożoność obliczeniowa

21

Porównanie klas złożoności obliczeniowych		
Klasa złożoności obliczeniowej	Nazwa klasy złożoności obliczeniowej	Cechy algorytmu
$\Theta(1)$	stała	działa prawie natychmiast
$\Theta(\log n)$	logarytmiczna	bardzo szybki
$\Theta(n)$	liniowa	szybki
$\Theta(n \log n)$	liniowo-logarytmiczna	dosyć szybki
$\Theta(n^2)$	kwadratowa	wolny dla dużych n
$\Theta(n^3)$	sześcienne	wolny dla większych n
$\Theta(2^n), \Theta(n!)$	wykładnicza	nierealizowalny dla większych n

Sortowanie naiwne

22



Cechy Algorytmu Sortowania Naiwnego

klasa złożoności obliczeniowej optymistyczna	$\Theta(n) - \Theta(n^2)$
klasa złożoności obliczeniowej typowa	$\Theta(n^3)$
klasa złożoności obliczeniowej pesymistyczna	$\Theta(n^3)$
Sortowanie w miejscu	TAK
Stabilność	TAK

Pesymistyczna:

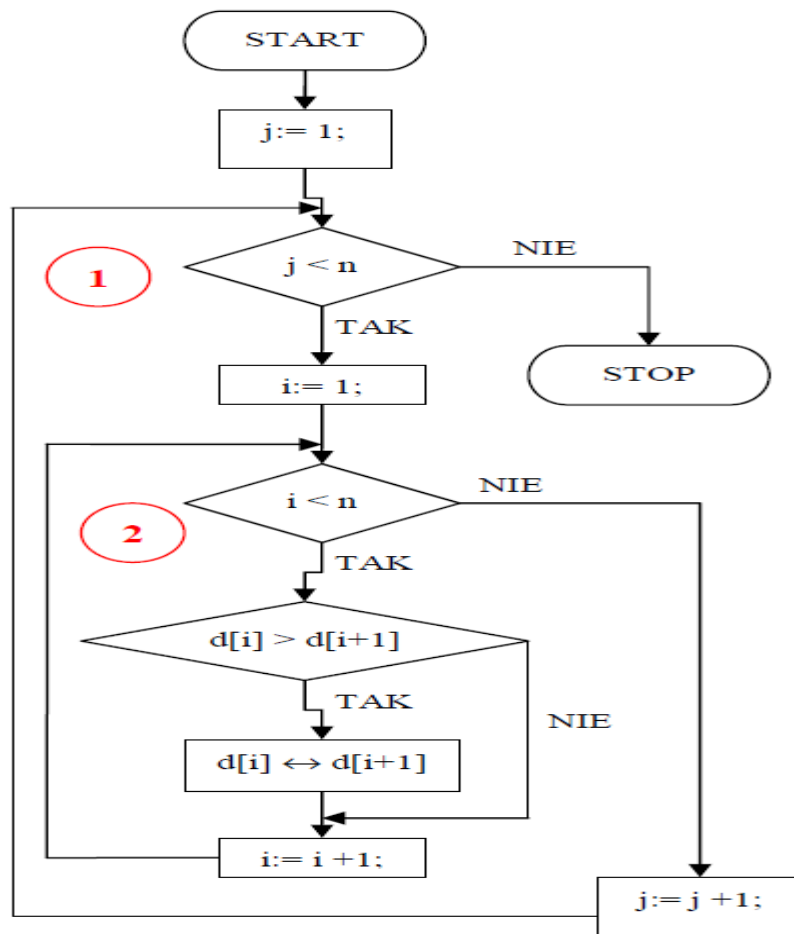
dla zbiorów posortowanych odwrotnie

Optymistyczna:

dla zbiorów uporządkowanych z niewielką ilością elementów nie na swoich miejscach

Sortowanie bąbelkowe

23



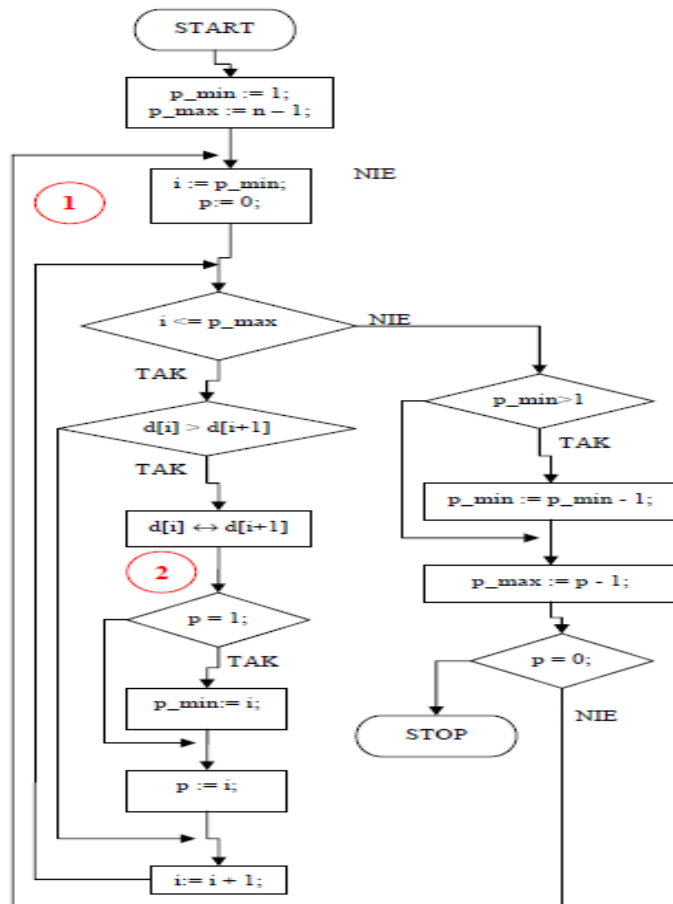
Sortowanie wykonywane jest w dwóch zagnieżdżonych pętlach. Pętla zewnętrzna nr 1 kontrolowana jest przez zmienną j . Wykonuje się ona $n - 1$ razy. Wewnątrz pętli nr 1 umieszczona jest pętla nr 2 sterowana przez zmienną i . Wykonuje się ona również $n - 1$ razy. W efekcie algorytm wykonuje w sumie: $T_1(n) = (n - 1)^2 = n^2 - 2n + 1$ obiegów pętli wewnętrznej, po których zakończeniu zbiór zostanie posortowany.

Cechy Algorytmu Sortowania Bąbelkowego wersja nr 1

klasa złożoności obliczeniowej optymistyczna	$\Theta(n^2)$
klasa złożoności obliczeniowej typowa	$\Theta(n^2)$
klasa złożoności obliczeniowej pesymistyczna	$\Theta(n^2)$
Sortowanie w miejscu	TAK
Stabilność	TAK

Sortowanie bąbelkowe: modyfikacje

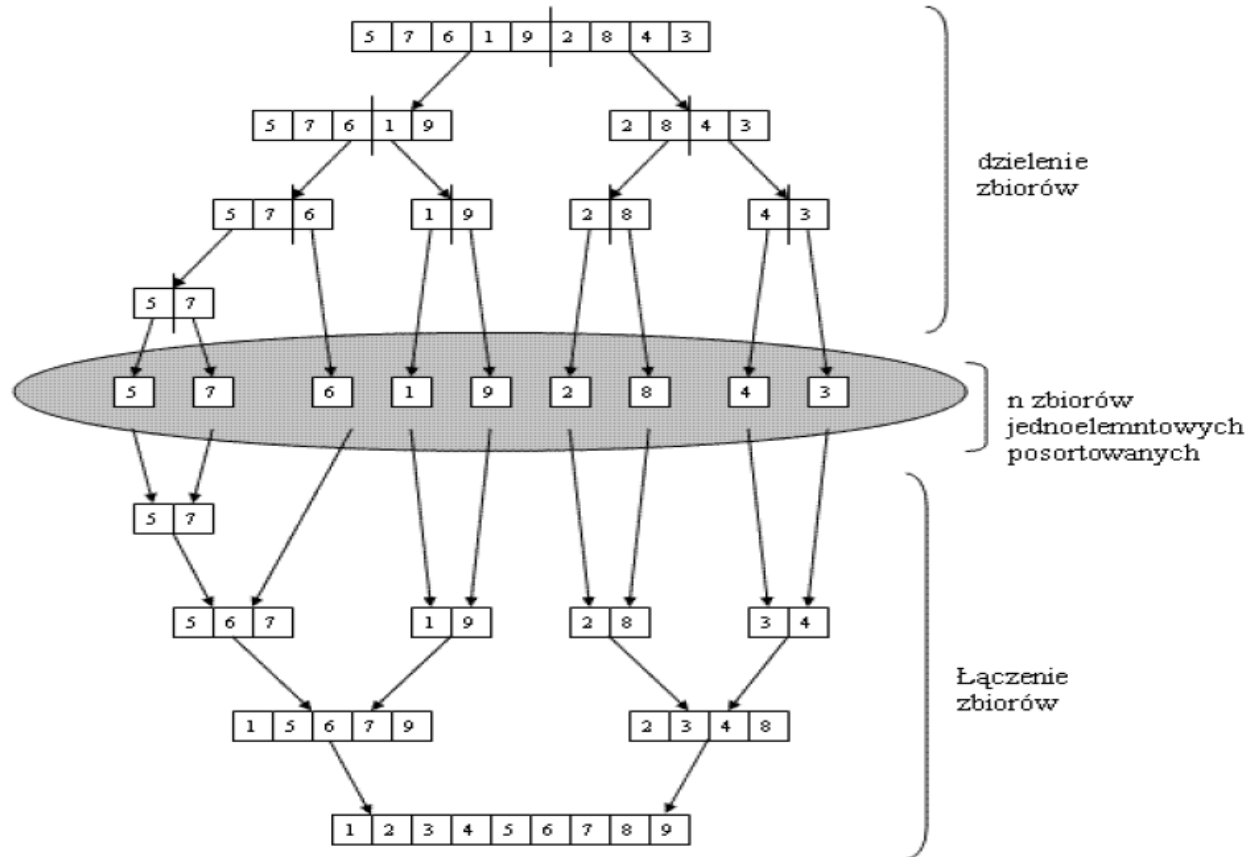
24



Cechy Algorytmu Sortowania Bąbelkowego wersja nr 2	
klasa złożoności obliczeniowej optymistyczna	$\Theta(n)$
klasa złożoności obliczeniowej typowa	$\Theta(n^2)$
klasa złożoności obliczeniowej pesymistyczna	$\Theta(n^3)$
Sortowanie w miejscu	TAK
Stabilność	TAK

Rekurencja: sortowanie

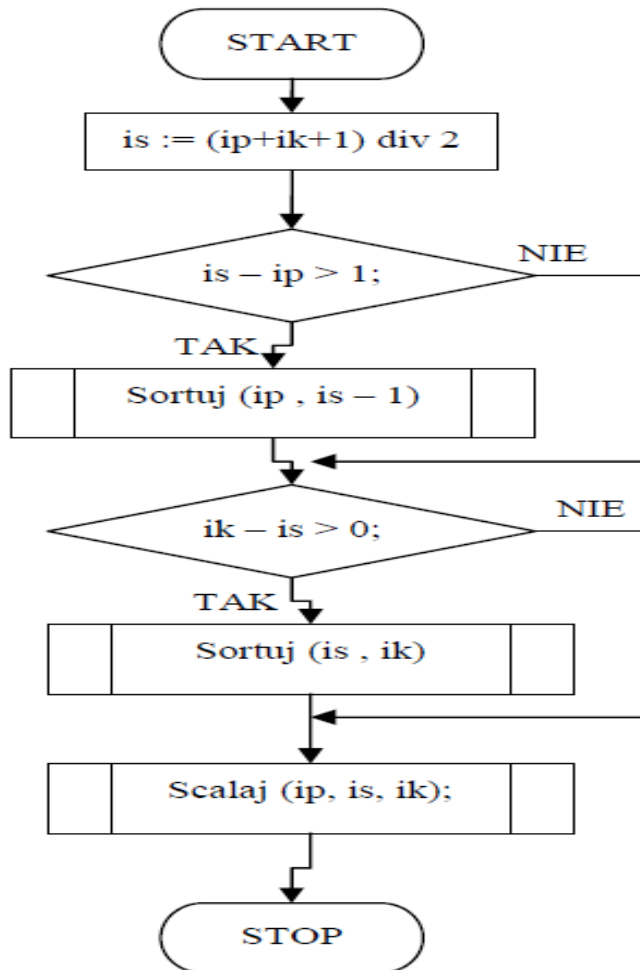
25



Przykład : sortujemy zbiór o postaci: $\{ 6\ 5\ 4\ 1\ 3\ 7\ 9\ 2\}$

Rekurencja: program sortuj

26



Złożoność obliczeniowa
 $O(n \log(n))$

Dane wejściowe :

d [] - zbiór scalony
ip - indeks pierwszego elementu w młodszym podzbiorze, $ip \in \mathbb{N}$
ik - indeks ostatniego elementu w starszym podzbiorze, $ik \in \mathbb{N}$

Dane wyjściowe :

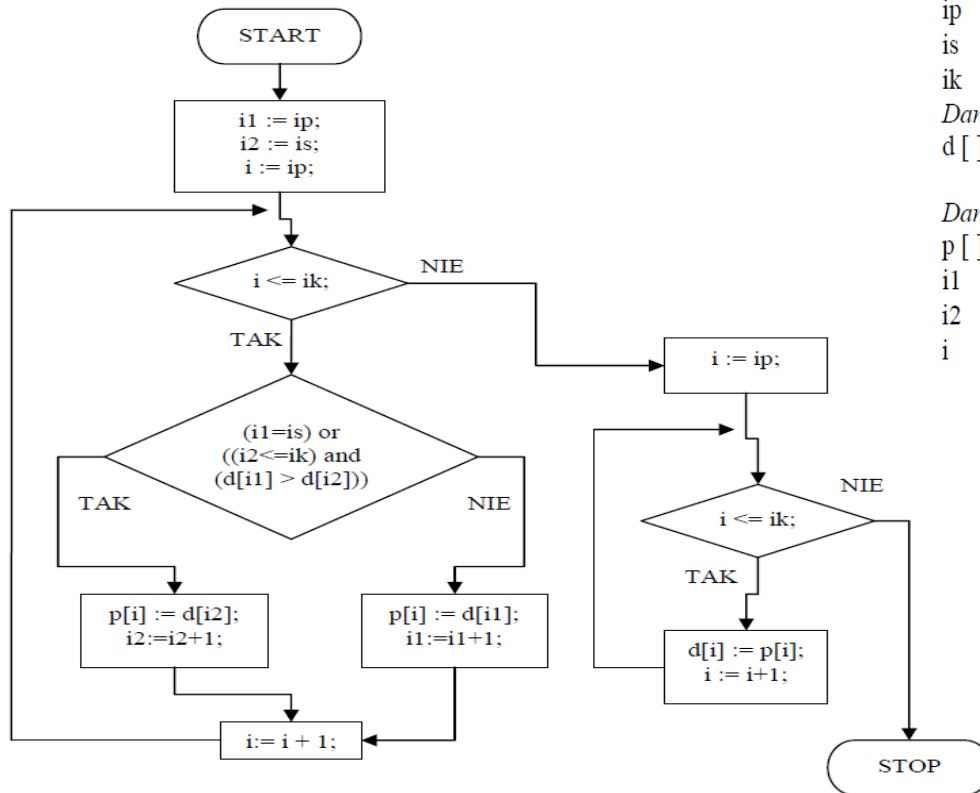
d [] - zbiór scalony

Dane pomocnicze :

is - indeks pierwszego elementu w starszym podzbiorze, $is \in \mathbb{N}$

Rekurencja: blok scalaj

27



Dane wejściowe :

- $d[]$ - zbiór scalony
- ip - indeks pierwszego elementu w młodszym podzbiore, $ip \in \mathbb{N}$
- is - indeks pierwszego elementu w starszym podzbiore, $is \in \mathbb{N}$
- ik - indeks ostatniego elementu w starszym podzbiore, $ik \in \mathbb{N}$

Dane wyjściowe :

- $d[]$ - zbiór scalony

Dane pomocnicze :

- $p[]$ - zbiór pomocniczy zawierający tyle samo elementów ile zbiór d
- $i1$ - indeks elementów w młodszej połowie zbioru $d[]$, $i1 \in \mathbb{N}$
- $i2$ - indeks elementów w starszej połowie zbioru $d[]$, $i2 \in \mathbb{N}$
- i - indeks elementów w zbiorze $p[]$, $i \in \mathbb{N}$

Złożoność obliczeniowa $O(n)$

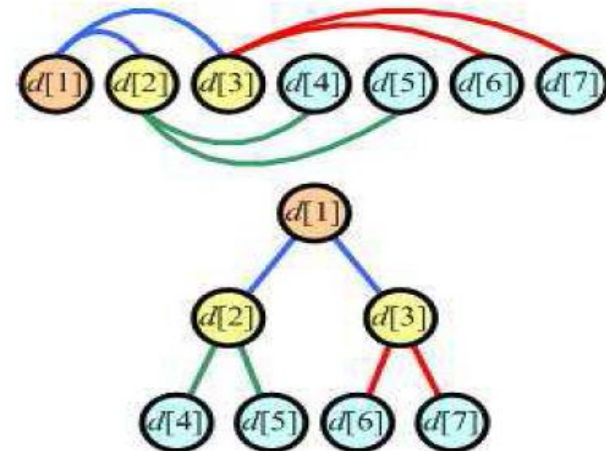
Sortowanie stogowe

28

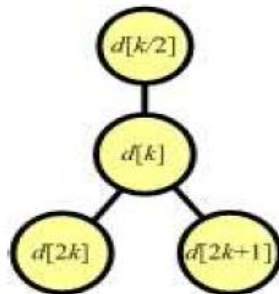
Zastosujmy następujące odwzorowanie:

- Element $d[1]$ będzie zawsze korzeniem drzewa.
- i -ty poziom drzewa binarnego wymaga 2^{i-1} węzłów. Będziemy je kolejno pobierać z tablicy.

Otrzymamy w ten sposób następujące odwzorowanie elementów tablicy w drzewo binarne:



Dla węzła k -tego wprowadzamy następujące wzory:



węzły potomne mają indeksy równe:

$2k$ - lewy potomek

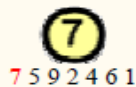
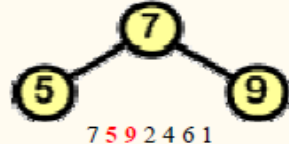
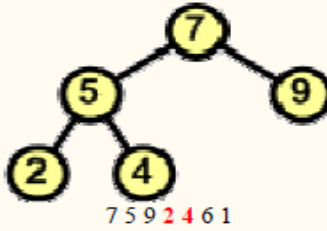
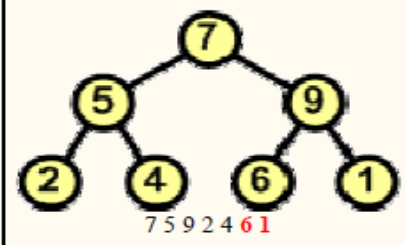
$2k+1$ - prawy potomek

węzeł nadrzędny ma indeks równy $[k/2]$ (dzielenie całkowitoliczbowe)

Drzewo binarne

29

Przykład : Skonstruować drzewo binarne z elementów zbioru {7 5 9 2 4 6 1}


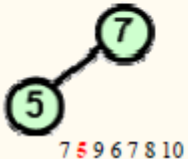
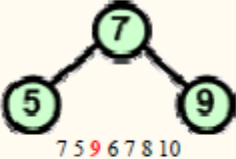

Operacja	Opis
	Konstrukcję drzewa binarnego rozpoczynamy od korzenia, który jest pierwszym elementem zbioru, czyli liczbą 7.
	Do korzenia dołączamy dwa węzły potomne, które leżą obok w zbiorze. Są to dwa kolejne elementy, 5 i 9.
	Do lewego węzła potomnego (5) dołączamy jego węzły potomne. Są to kolejne liczby w zbiorze, czyli 2 i 4.
	Pozostaje nam dołączyć do prawego węzła ostatnie dwa elementy zbioru, czyli liczby 6 i 1. Drzewo jest kompletne.

Kopiec : tworzenie

30

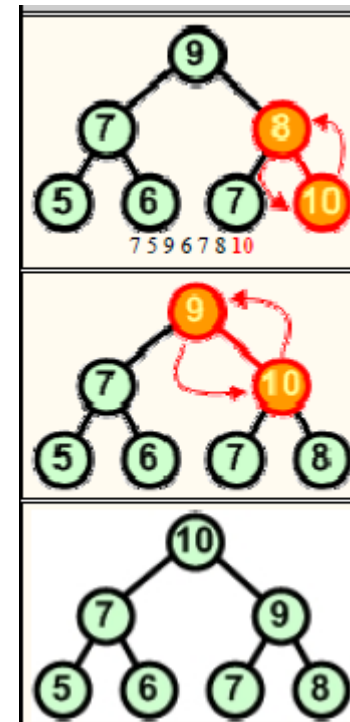
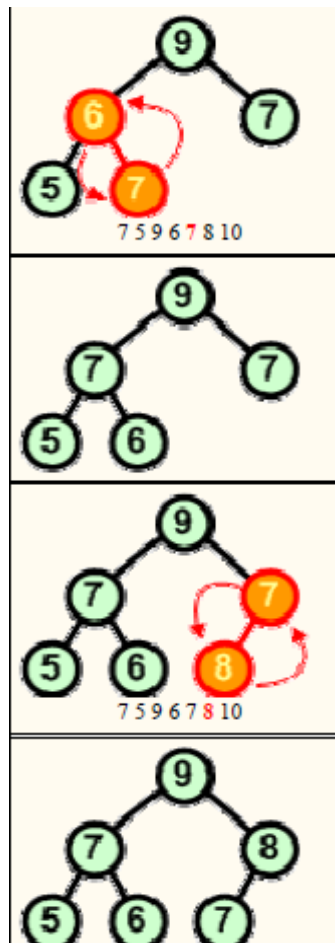
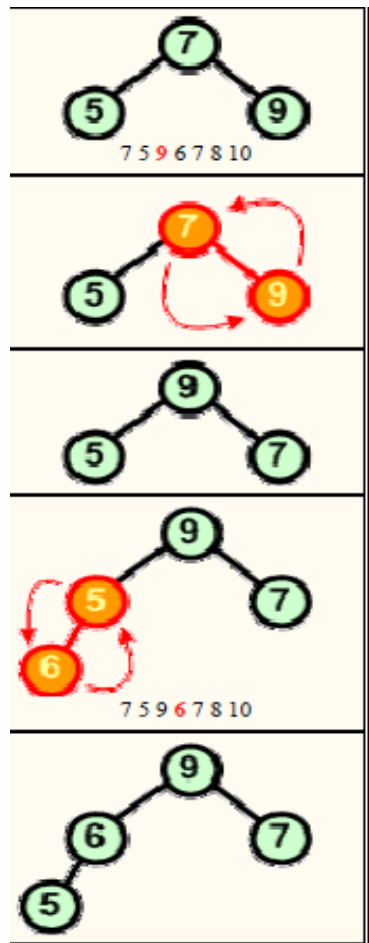
Kopiec jest drzewem binarnym, w którym wszystkie węzły spełniają następujący warunek (zwany warunkiem kopca) : węzeł nadrzędny jest większy lub równy węzłom potomnym (w porządku malejącym relacja jest odwrotna - mniejszy lub równy).

Przykład : Skonstruować kopiec z elementów zbioru {7 5 9 6 7 8 10}

Operacja	Opis
	Budowę kopca rozpoczynamy od pierwszego elementu zbioru, który staje się korzeniem.
	Do korzenia dołączamy następny element. Warunek kopca jest zachowany.
	Dodajemy kolejny element ze zbioru.
	Po dodaniu elementu 9 warunek kopca przestaje być spełniony. Musimy go przywrócić. W tym celu za nowy węzeł nadrzędny wybieramy nowo dodany węzeł. Poprzedni węzeł nadrzędny wędruje w miejsce węzła dodanego - zamieniamy węzły 7 i 9 miejscami.

Kopiec

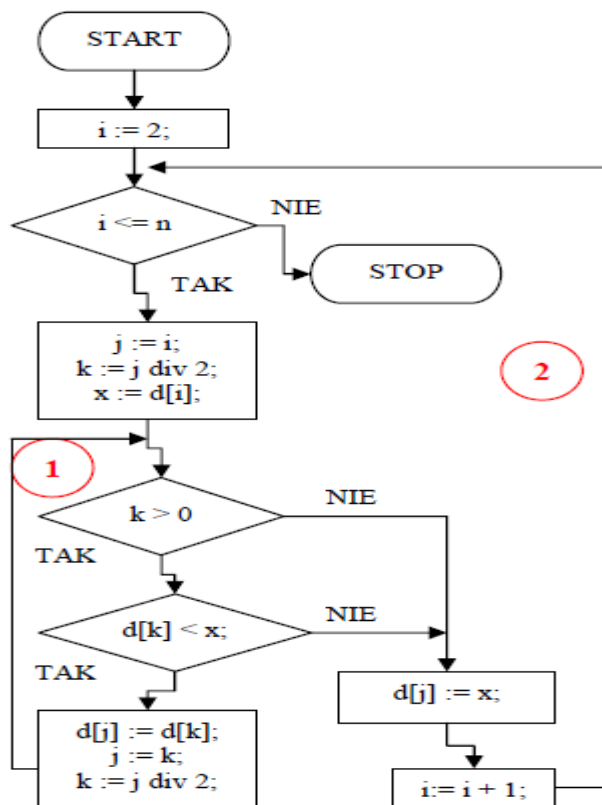
31



Kopiec

32

Schemat blokowy



Złożoność obliczeniowa $O(n \log(n))$

Algorytm tworzy kopiec w tym samym zbiorze wejściowym $d[]$. Nie wymaga zatem dodatkowych struktur danych i ma złożoność pamięciową klasy $\Theta(n)$.

Pętla nr 1 wyznacza kolejne elementy wstawiane do kopca. Pierwszy element pomijamy, ponieważ zostałby i tak na swoim miejscu. Dlatego pętla rozpoczyna wstawianie od elementu nr 2.

Wewnątrz pętli nr 1 inicjujemy kilka zmiennych:
 j - pozycja wstawianego elementu (liścia)
 k - pozycja elementu nadrzędnego (przodka)
 x - zapamiętuje wstawiany element

Następnie rozpoczynamy pętlę warunkową nr 2, której zadaniem jest znalezienie w kopcu miejsca do wstawienia zapamiętanego elementu w zmiennej x . Pętla ta wykonuje się do momentu osiągnięcia korzenia kopca ($k = 0$) lub znalezienia przodka większego od zapamiętanego elementu. Wewnątrz pętli przesuwamy przodka na miejsce potomka, aby zachować warunek kopca, a następnie przesuwamy pozycję j na pozycję zajmowaną wcześniej przez przodka. Pozycja k staje się pozycją nowego przodka i pętla się kontynuuje. Po jej zakończeniu w zmiennej j znajduje się numer pozycji w zbiorze $d[]$, na której należy umieścić element w x . Po zakończeniu pętli nr 1 w zbiorze zostaje utworzona struktura kopca.

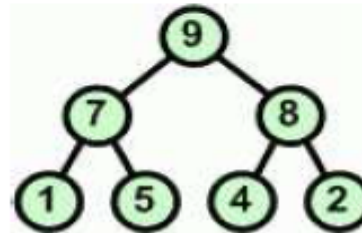
Kopiec: rozbieranie

33

Zasady rozbioru kopca :

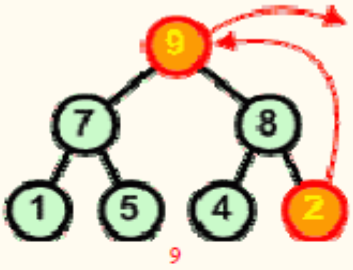
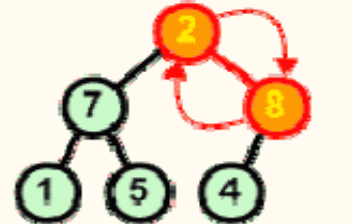
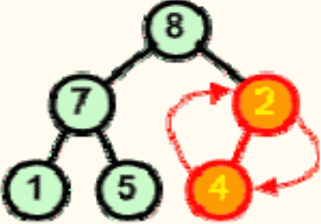
1. Zamień miejscami korzeń z ostatnim liściem, który wyłącz ze struktury kopca. Elementem pobieranym z kopca jest zawsze jego korzeń, czyli element największy.
2. Jeśli jest to konieczne, przywróć warunek kopca idąc od korzenia w dół.
3. Kontynuuj od kroku 1, aż kopiec będzie pusty.

Przykład : Rozebrać kopiec



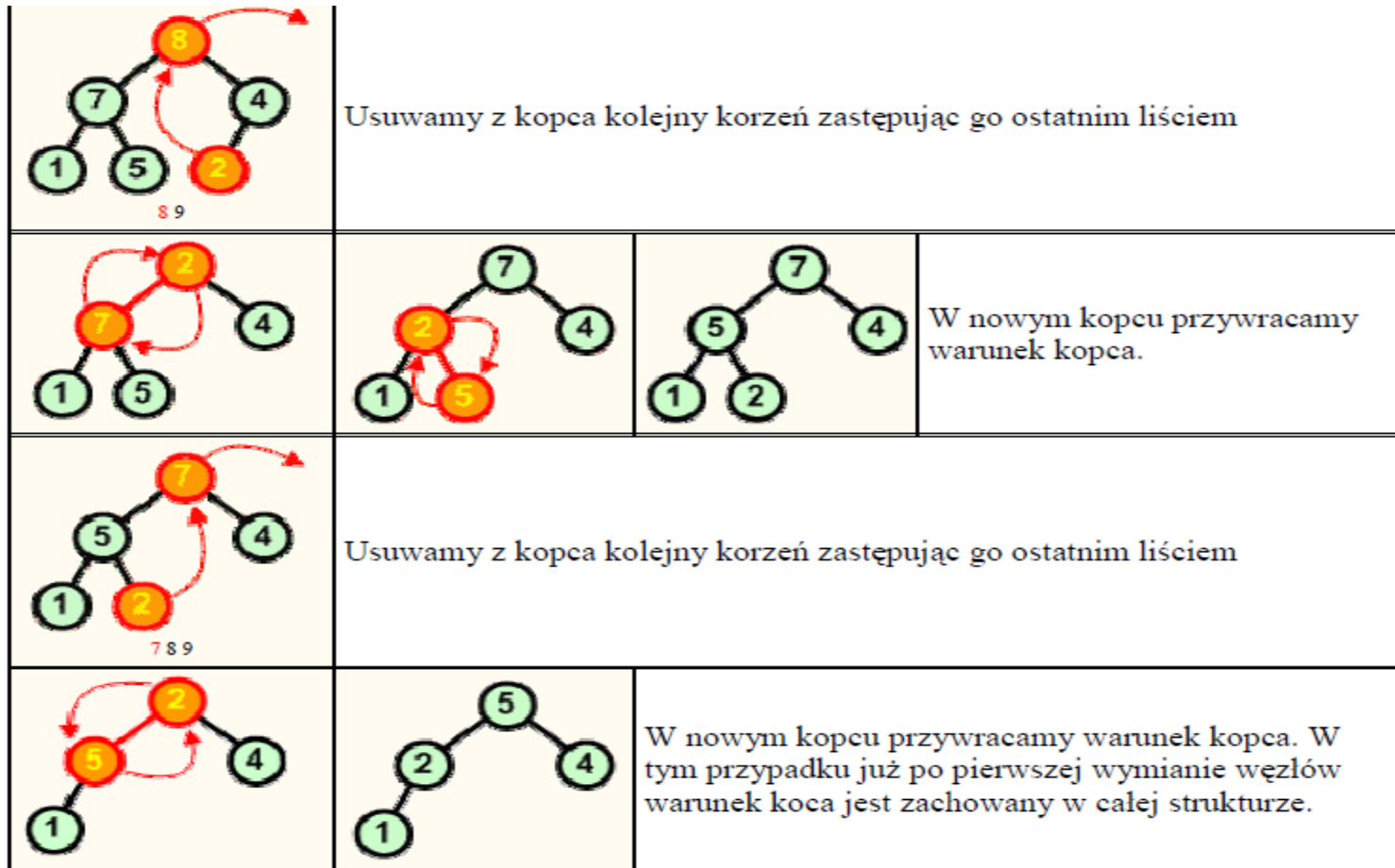
Kopiec: rozbieranie

34

Z	Operacja	Opis		
1		<p>Rozbiór kopca rozpoczynamy od korzenia, który usuwamy ze struktury kopca. W miejsce korzenia wstawiamy ostatni liść.</p>		
2				
3				
P				<p>Poprzednia operacja zaburzyła strukturę kopca. Idziemy zatem od korzenia w dół struktury przywracając warunek kopca - przodek większy lub równy od swoich potomków. Praktycznie polega to na zamianie przodka z największym potomkiem. Operację kontynuujemy dotąd, aż natrafimy na węzły spełniające warunek kopca.</p>

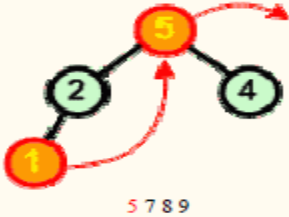







Kopiec: rozbieranie

35



Kopiec: rozbieranie

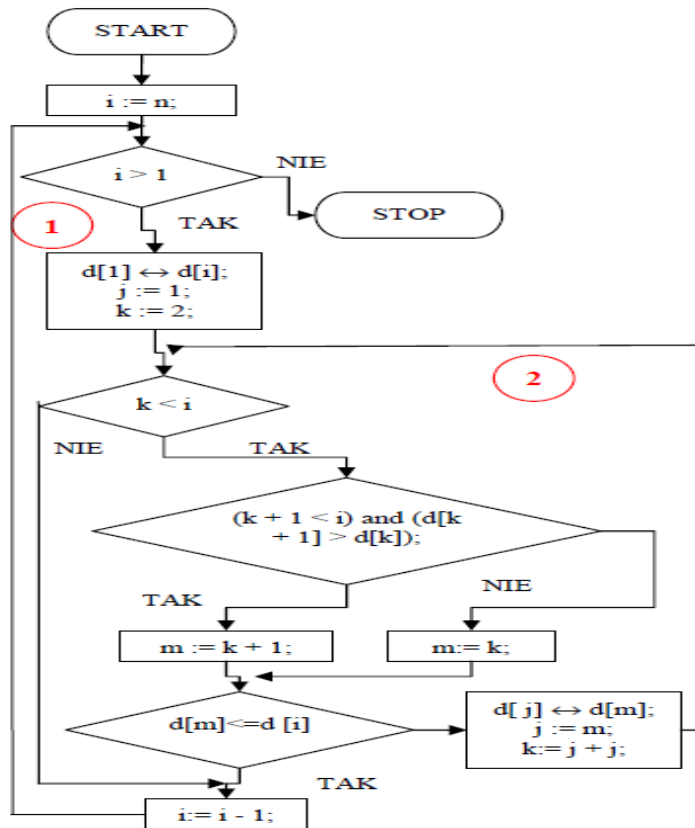
36

 <p>5 7 8 9</p>	Usuujemy z kopca kolejny korzeń zastępując go ostatnim liściem	
 <p>1 2 3 4 5 6 7 8 9</p>		Przywracamy warunek kopca w strukturze.
 <p>4 5 7 8 9</p>	Usuujemy z kopca kolejny korzeń zastępując go ostatnim liściem	
 <p>1 2 3 4 5 6 7 8 9</p>		Przywracamy warunek kopca w strukturze.
 <p>2 4 5 7 8 9</p>	Usuujemy z kopca kolejny korzeń zastępując go ostatnim liściem.	
 <p>1 2 4 5 7 8 9</p>	Po wykonaniu poprzedniej operacji usunięcia w kopcu pozostał tylko jeden element - usuwamy go. Zwróć uwagę, iż usunięte z kopca elementy tworzą ciąg uporządkowany.	

Kopiec: rozbieranie

37

Schemat blokowy



Rozbiór kopca wykonywany jest w dwóch zagnieżdżonych pętlach. Pętla nr 1 zamienia miejscami kolejne liście ze spodu drzewa z korzeniem. Zadaniem pętli nr 2 jest przywrócenie w strukturze warunku kopca.

Złożoność obliczeniowa
 $O(n \log(n))$

Sortowanie przez kopcowanie

38

- Krok 1 : Tworz_Kopiec
- Krok 2 : Rozbierz_Kopiec
- Krok 3 : Zakończ algorytm

Cechy Algorytmu Sortowania Przez Kopcowanie	
klasa złożoności obliczeniowej optymistyczna	$\Theta(n \log n)$
klasa złożoności obliczeniowej typowa	
klasa złożoności obliczeniowej pesymistyczna	
Sortowanie w miejscu	TAK
Stabilność	NIE

Ponieważ sortowanie przez kopcowanie składa się z dwóch następujących bezpośrednio po sobie operacji o klasie czasowej złożoności obliczeniowej $\Theta(n \log n)$, to dla całego algorytmu klasa złożoności również będzie wynosić $\Theta(n \log n)$.

Algorytmy sortujące

39

Nazwa algorytmu sortującego	Klasa złożoności			Stabilność	Sortowanie w miejscu	Zalecane?
	optymistyczna	typowa	pesymistyczna			
Zwariowane	$\Theta(1)$	$\Theta(n*n!)$	$\Theta(\infty)$	NIE	TAK	NIE!!!
Naiwne	$\Theta(n) \dots \Theta(n^2)$	$\Theta(n^3)$	$\Theta(n^3)$	TAK	TAK	NIE
Bąbelkowe wersja 1	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	TAK	TAK	NIE
Bąbelkowe wersja 2	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	TAK	TAK	TAK
Przez wybór	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	NIE	TAK	TAK/NIE
Przez wstawianie	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	TAK	TAK	TAK
Metodą Shella	$\Theta(n^{1,14})$	$\Theta(n^{1,15})$	$\Theta(n^{1,15})$	NIE	TAK	TAK
Przez łączenie	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	TAK	NIE	TAK
Przez kopcowanie	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	NIE	TAK	TAK
Szybkie	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	NIE	TAK	TAK
Kubelkowe wersja I	$\Theta(m + n)$	$\Theta(m + n)$	$\Theta(m + n)$	NIE	NIE	TAK/NIE
Radix Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	TAK	NIE	TAK