Projekt

porównanie sortowań: przez scalanie oraz przez kopcowanie

Jakub Kuźniar grupa 3

2021

Spis treści

[1. Wstęp 1](#_Toc90487045)

[1.1. Sortowanie przez scalanie 1](#_Toc90487046)

[1.1.1. Złożoność obliczeniowa sortowania przez scalanie 2](#_Toc90487047)

[1.1.2. Szczegóły implementacji 2](#_Toc90487048)

[1.1.3. Wykres oraz test dla algorytmu sortowania poprzez scalanie 6](#_Toc90487049)

[1.2. Sortowanie przez kopcowanie 7](#_Toc90487050)

[1.2.1. Złożoność obliczeniowa algorytmu sortowania przez kopcowanie 8](#_Toc90487051)

[1.2.2. Przedstawienie kodu oraz schematów blokowych dla sortowania przez kopcowanie 8](#_Toc90487052)

[1.2.3. Testy i wykresy 14](#_Toc90487053)

[Rys. 1 Algorytm do dzielenia, sortowania oraz scalania\ 2](#_Toc90487020)

[Rys. 2 Algorytm służący do zapisu oraz odczytu plików 3](#_Toc90487021)

[Rys. 3 Schemat blokowy algorytmu sortowania przez scalanie 4](#_Toc90487022)

[Rys. 4 Test dla 20 elementów 6](#_Toc90487023)

[Rys. 5 Test dla 100 elementów 6](#_Toc90487024)

[Rys. 6 Test dla 1000 elementów 6](#_Toc90487025)

[Rys. 7 Test dla 10000 elementów 6](#_Toc90487026)

[Rys. 8 Test dla 100000 elementów 6](#_Toc90487027)

[Rys. 9 Przykładowe działanie algorytmu sortowania przez kopcowanie 8](#_Toc90487028)

[Rys. 10 Kod źródłowy budowy kopca 9](#_Toc90487029)

[Rys. 11 Schemat blokowy budowy kopca 10](#_Toc90487030)

[Rys. 12 Kod źródłowy rozbioru kopca 11](#_Toc90487031)

[Rys. 13 Schemat blokowy rozbioru kopca 12](#_Toc90487032)

[Rys. 14 Obsługa plików tekstowych 13](#_Toc90487033)

[Rys. 15 Test dla 20 elementów pesymistycznie 14](#_Toc90487034)

[Rys. 16 Test dla 100 elementów pesymistycznie 14](#_Toc90487035)

[Rys. 17 Test dla 1000 elementów pesymistycznie 14](#_Toc90487036)

[Rys. 18 Test dla 10000 elementów pesymistycznie 14](#_Toc90487037)

[Rys. 19 Test dla 20 elementów optymistycznie 15](#_Toc90487038)

[Rys. 20 Test dla 100 elementów optymistycznie 15](#_Toc90487039)

[Rys. 21 Test dla 1000 elementów optymistycznie 15](#_Toc90487040)

[Rys. 22 Test dla 10000 elementów optymistycznie 15](#_Toc90487041)

[Wykres 1 Złożoność czasowa dla algorytmu sortowania przez scalanie 7](#_Toc90487042)

[Wykres 2 Sortowanie przez kopcowanie pesymistyczne 14](#_Toc90487043)

[Wykres 3 Wykres sortowania przez kopcowanie optymistyczne 15](#_Toc90487044)

# Wstęp

Porównanie dwóch algorytmów sortowania przez scalanie oraz kopcowanie.

## Sortowanie przez scalanie

Jest to algorytm rekurencyjny, wykorzystujący zasadę dziel i zwyciężaj, która polega na podzieleniu problemu na mniejsze części przez co jest łatwiejszy do rozwiązania. Algorytm sortujący dzieli zbiór na kolejne połowy dopóki podział jest możliwy. Następnie algorytm sortuje rekurencyjnie otrzymane zbiory, później je łączy za pomocą scalania, przez co zbiór końcowy jest posortowany.

### Złożoność obliczeniowa sortowania przez scalanie

Złożoność obliczeniowa to (z indukcji matematycznej)

### Szczegóły implementacji

Algorytm dzieli dany zbiór na połówki, następnie je sortuje i scala.

Funkcja MergeSort(int i\_p, int i\_k, int tab[], int pomoc[]) przyjmuje 4 argumenty: i\_p to index pierszego elementu w młodszym podzbiorze, i\_k to index ostatniego elementu w starszym podzbiorze, tab[] to posortowany zbiór, pomoc[] to zbiór pomocniczy.



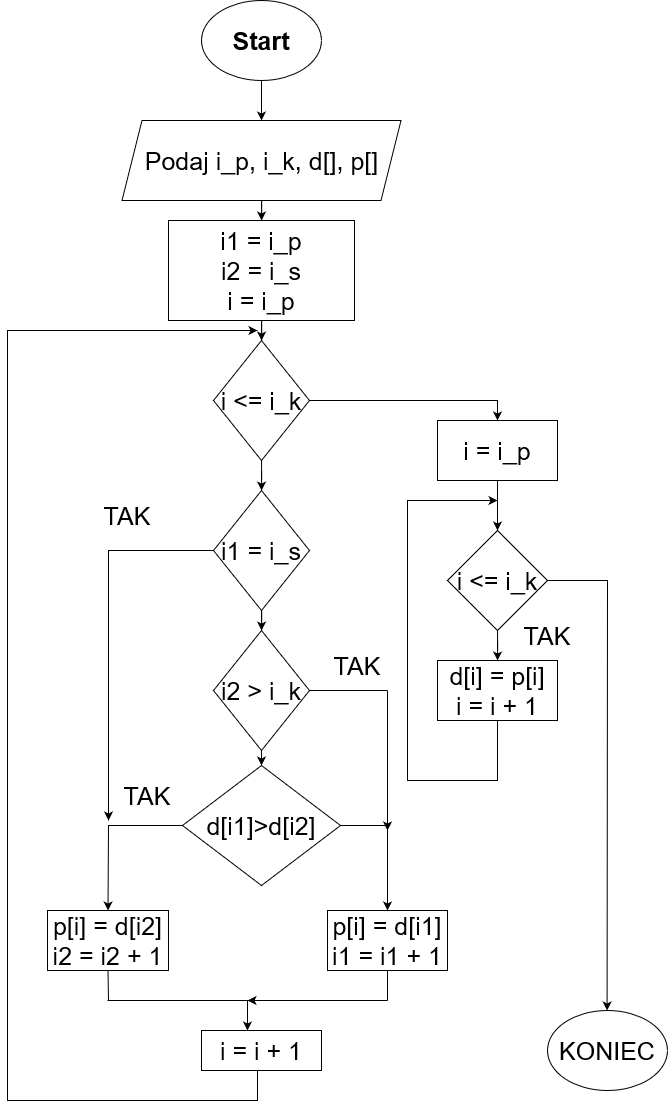
Rys. 1 Algorytm do dzielenia, sortowania oraz scalania\

Program zawiera także możliwość odczytywania z pliku, losowania do pliku oraz zapisywania wyników do innego pliku.



Rys. 2 Algorytm służący do zapisu oraz odczytu plików

Schemat blokowy:



Rys. 3 Schemat blokowy algorytmu sortowania przez scalanie

Pseudokod:

K01:

i 1= i\_p; i2 = i\_s, i = i\_p

K02:

Dla i = i\_p, i\_p + 1, ..., i\_k: wykonuj

jeśli (i1 = i\_s) ∨ (i2 ≤ i\_k i d[i1] > d[i2]), to

p[i] ← d[i2]; i2 ← i2 + 1

inaczej

p[i] ← d[i1]; i1 ← i1 + 1

K03:

Dla i = i\_p, i\_p + 1,...,i\_k: d[i] ← p[i]

K04:

Zakończ

### Wykres oraz test dla algorytmu sortowania poprzez scalanie

Algorytm posiada jedną złożoność obliczeniową dla każdego z 3 przypadków dlatego wykonam jeden wykres, który będzie dotyczył pesymistycznego, oczekiwanego oraz optymistycznego przypadku.

* Dla 20 elementów Obraz zawierający tekst, monitor, ekran, telewizja

  Opis wygenerowany automatycznie

Rys. 4 Test dla 20 elementów

* Dla 100 elementów

Obraz zawierający tekst, komputer, monitor, klawiatura

Opis wygenerowany automatycznie

Rys. 5 Test dla 100 elementów

* Dla 1000 elementów (dla zaoszczędzenia papieru oraz miejsca będę uwzględniał tylko czas wykonania algorytmu)

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rys. 6 Test dla 1000 elementów

* Dla 10000 elementów



Rys. 7 Test dla 10000 elementów

* Dla 100000 elementów



Rys. 8 Test dla 100000 elementów

Wykres 1 Złożoność czasowa dla algorytmu sortowania przez scalanie

## Sortowanie przez kopcowanie

Zadaniem algorytmu jest zbudowanie kopca, posortowanie danych z kopca i rozebranie go.

Kopiec jest drzewem binarnym, w którym wszystkie węzły spełniają następujący warunek:

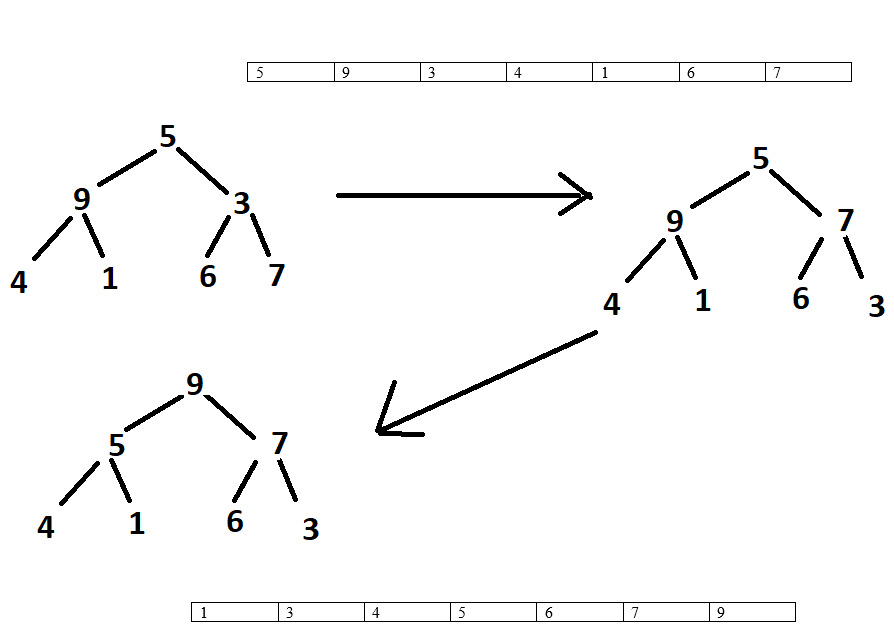
* Węzeł nadrzędny jest większy lub równy węzłom potomnym

Kopiec jest bardziej skomplikowany niż drzewo binarne ponieważ po dołączeniu nowego elementu musimy sprawdzać czy zachodzi warunek kopca.

Charakterystyczną cechą kopca jest to, iż korzeń zawsze jest największym (w porządku malejącym) elementem z całego drzewa.

Rozbiór kopca jest kolejna czynnością którą musimy wykonać. Zamieniamy miejscami korzeń z ostatnim liściem, który wyłączamy ze struktury kopca. Elementem pobieranym jest zawsze największy czyli jest korzeniem. Należy pamiętać o warunku i sprawdzać czy zachodzi. Powtarzamy aż kopiec będzie pusty.

Przykładowe działanie algorytmu:



Rys. 9 Przykładowe działanie algorytmu sortowania przez kopcowanie

### Złożoność obliczeniowa algorytmu sortowania przez kopcowanie

Złożoność obliczeniowa dla sortowania przez kopcowanie:

* Przypadek pesymistyczny -
* Przypadek optymistyczny - dla zbioru tych samych elementów

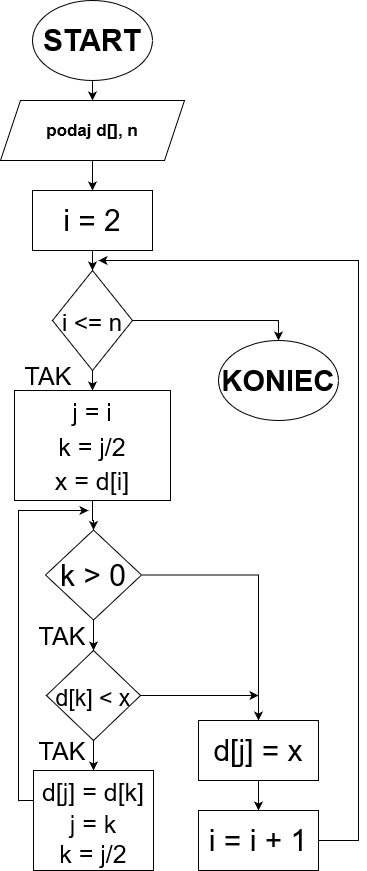
### Przedstawienie kodu oraz schematów blokowych dla sortowania przez kopcowanie

Ponieważ cały algorytm składa się z dwóch różnych funkcji (budowania oraz rozbioru kopca) przedstawię dwie funkcje, dwa schematy blokowe oraz dwa pseudokody.

1. Budowa kopca
   1. Kod źródłowy:

Rys. 10 Kod źródłowy budowy kopca

* 1. Schemat blokowy:



Rys. 11 Schemat blokowy budowy kopca

* 1. Pseudokod:

**K01: Dla i = 2, ..., n: wykonuj K02...K05**

**K02: j ← i; k ← j div 2**

**K03: x ← d[i]**

**K04: Dopóki (k > 0) ∧ (d[k] < x): wykonuj**

**d[j] ← d[k]**

**j ← k**

**k ← j div 2**

**K05: d[j] ← x**

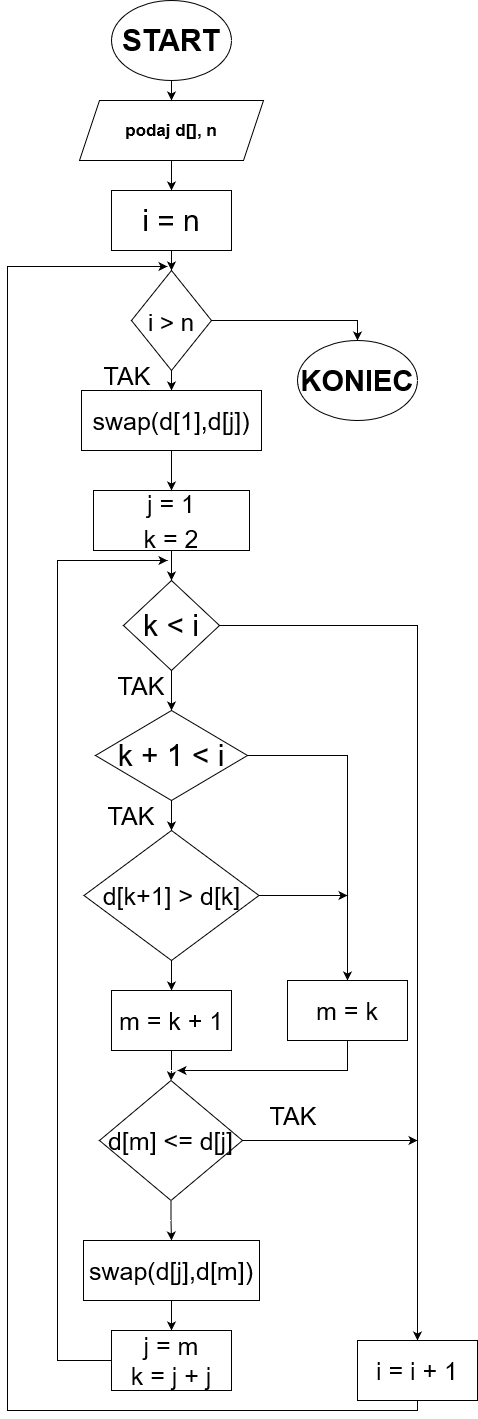
**K06: Zakończ**

1. Rozbiór kopca
   1. Kod źródłowy:



Rys. 12 Kod źródłowy rozbioru kopca

* 1. Schemat blokowy:



Rys. 13 Schemat blokowy rozbioru kopca

* 1. Pseudokod:

**K01: Dla i = n, n - 1, ..., 2: wykonuj K02...K08**

**K02: d[1] ↔ d[i]**

**K03: j ← 1; k ← 2**

**K04: Dopóki (k < i) wykonuj K05...K08**

**K05: Jeżeli (k + 1 < i) ∧ (d[k + 1] > d[k]),**

**to m ← k + 1**

**inaczej m ← k**

**K06: Jeżeli d[m] ≤ d[j], to wyjdź z pętli K04 i kontynuuj następny obieg K01**

**K07: d[j] ↔ d[m]**

**K08: j ← m; k ← j + j**

**K09: Zakończ**

1. Kod źródłowy dla dwóch funkcji obsługujących operacje na plikach



Rys. 14 Obsługa plików tekstowych

### Testy i wykresy

1. Przypadek pesymistyczny

* Dla 20 elementówObraz zawierający tekst, czarny, ekran, zamknąć

  Opis wygenerowany automatycznie

Rys. 15 Test dla 20 elementów pesymistycznie

* Dla 100 elementów

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rys. 16 Test dla 100 elementów pesymistycznie

* Dla 1000 elementów



Rys. 17 Test dla 1000 elementów pesymistycznie

* Dla 10000 elementów



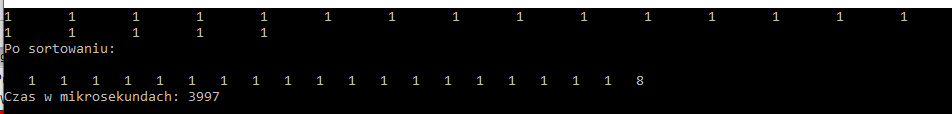
Rys. 18 Test dla 10000 elementów pesymistycznie

* Wykres

Wykres 2 Sortowanie przez kopcowanie pesymistyczne

1. Przypadek optymistyczny

* Dla 20 elementów



Rys. 19 Test dla 20 elementów optymistycznie

* Dla 100 elementów

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rys. 20 Test dla 100 elementów optymistycznie

* Dla 1000 elementów



Rys. 21 Test dla 1000 elementów optymistycznie

* Dla 10000 elementów



Rys. 22 Test dla 10000 elementów optymistycznie

* Wykres

Wykres 3 Wykres sortowania przez kopcowanie optymistyczne

Link do GitHub z kodem źródłowym: <https://github.com/Tyfytyfy/projektSortowanie>