Feliks Kurp

# ALGORYTMY

## Struktury danych i złożoność obliczeniowa

Helion

Redaktor prowadzący: Szymon Szwajger

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
*https://helion.pl/user/opinie/alstda*
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-9530-5


Printed in Poland.

# Spis treści

# ROZDZIAŁ 4. Algorytmy sortowania tablic

**Sortowanie tablic** to proces, którego wynikiem końcowym jest ustawienie elementów tablicy rosnąco lub w porządku odwrotnym — malejąco.

Opracowano wiele algorytmów fizycznego sortowania tablic, opartych na bardzo różnych pomysłach. Mówiąc o fizycznym sortowaniu, mamy na myśli algorytmy, które przemieszczają fizycznie elementy tablicy w pamięci komputera z jednego położenia w drugie w celu uzyskania uporządkowania. Jest to operacja bardzo kosztowna. Czas realizacji operacji przemieszczania elementów może być nawet kilka rzędów dłuższy niż innych operacji podstawowych, takich jak przykładowo przypisanie wartości do zmiennej.

Większość algorytmów sortowania fizycznego charakteryzuje się kosztem $O(N^2)$, gdzie $N$ jest liczbą elementów tablicy. Algorytmy te wymagają bowiem dwóch pętli iteracyjnych, przy czym jedna z nich jest zanurzona w drugiej.

Przy ocenie jakości algorytmów sortowania (podobnie jak innych bardziej złożonych algorytmów) należy również brać pod uwagę ich prostotę i złożoność pamięciową.

**Prostota algorytmu** jest cechą dość istotną. Algorytmy o prostej strukturze, oparte na prostym pomyśle, można bowiem łatwo modyfikować i dostosowywać do aktualnych potrzeb.

Kolejnym parametrem, który należy rozważyć, analizując własności algorytmów, jest zapotrzebowanie na dodatkową pamięć w trakcie pracy algorytmu. Tu algorytmy fizycznego sortowania nie są uciążliwe, na ogół bowiem sięgamy do metod tak zwanego **sortowania w miejscu**, nazywanych inaczej **metodami *in situ*** (łac.). Ich zapotrzebowanie na dodatkową pamięć ogranicza się zwykle do wielkości zajmowanej przez wartość pojedynczego elementu tablicy, przez który przesyłane są elementy tablicy celem ich zamiany.

W odniesieniu do algorytmów fizycznego sortowania, wybierając algorytm, należy zwrócić uwagę na wrażliwość na uporządkowanie sortowanej tablicy. Z tego punktu widzenia algorytmy sortowania dzielimy na:

- całkowicie niewrażliwe na uporządkowanie,
- częściowo wrażliwe na uporządkowanie,
- całkowicie wrażliwe na uporządkowanie.

W pierwszej grupie znajdą się algorytmy, dla których uporządkowanie tablicy (pierwotne bądź powstałe w trakcie sortowania) nie wpływa w sposób zasadniczy na czas realizacji algorytmu.

Za algorytmy częściowo wrażliwe na uporządkowanie uznamy te algorytmy sortowania, które w sposób znamienny ograniczają liczbę operacji wykonywanych w trakcie procesu sortowania (np. poprzez zawieszenie wykonywania pewnych pętli wewnętrznych).

Algorytmy sortowania całkowicie wrażliwe na uporządkowanie potrafią w trakcie realizacji algorytmu niejako przy okazji stwierdzić uporządkowanie tablicy (pierwotne bądź powstałe w dowolnym momencie procesu sortowania), przerywając natychmiast proces sortowania. Te algorytmy są oczywiście najefektywniejsze.

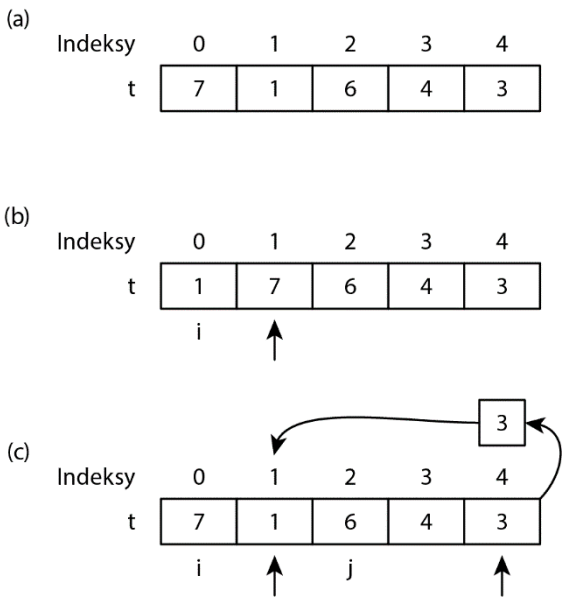
Algorytmy fizycznego sortowania tablic, jakkolwiek ciekawe z teoretycznego punktu widzenia, w chwili obecnej mają (głównie ze względu na swoją złożoność obliczeniową) znaczenie tylko historyczne, zostały bowiem zastąpione innymi metodami sortowania. Dwie z nich zostaną omówione w dalszej części.

Niżej omówiono działanie dwóch wybranych metod fizycznego sortowania tablic.

## 4.1. Sortowanie przez proste wstawianie

Pierwszy z dwóch wybranych algorytmów fizycznego sortowania tablic jest oparty na prostym pomyśle zamiany dwóch elementów tablicy przez „przepompowanie" ich przez pojedynczą komórkę pamięci, jest to więc algorytm typu *in situ*. Algorytm jest całkowicie niewrażliwy

na uporządkowanie. Rysunek 4.1 pozwala zrozumieć ideę jego działania.




**RYSUNEK 4.1.** Algorytm sortowania przez proste wstawianie: a) stan wyjściowy; b) stan po zakończeniu pierwszej iteracji; c) ilustracja procesu przepisywania elementów

Działanie algorytmu przebiega w dwóch pętlach: zewnętrznej i zanurzonej w niej pętli wewnętrznej.

Pętla zewnętrzna przesuwa indeks `i` wzdłuż tablicy. Dla każdego położenia indeksu `i` (rysunek 4.1b) pętla wewnętrzna przesuwa w pozostałej części tablicy indeks `j`, wyszukując element najmniejszy w tej części tablicy. Jeśli go znajdzie, wymienia ten element z elementem `t[i]` (rysunek 4.1c).

Złożoność obliczeniowa algorytmu sortowania przez proste wstawianie jest kwadratowa i wynosi $O(N^2)$, gdzie $N$ jest rozmiarem zadania (liczbą elementów tablicy).

Spróbujmy udowodnić tę tezę.

W algorytmie sortowania przez proste wstawianie, którego ideę zaprezentowano na rysunku 4.1, w sytuacji najgorszego przypadku (gdy tablica jest odwrotnie posortowana) pętla zewnętrzna wykona się

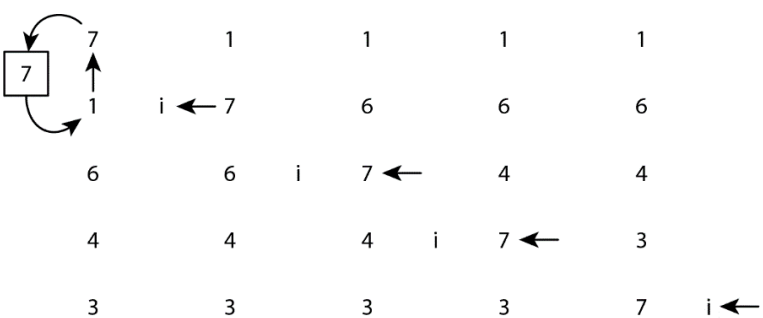
4 razy, a zanurzona w niej pętla wewnętrzna odpowiednio 3, 2 i 1 raz dla kolejnych położeń indeksu `i`. W sumie 6 razy.

Z powyższego wynika, że zanurzona w pętli wewnętrznej instrukcja zamiany elementów może wykonać się maksymalnie 4·6 = 24 razy, to jest nieco mniej niż podniesiona do kwadratu liczba elementów tablicy ($5^2 = 25$). Jak widać, jest to zgodne z definicją złożoności obliczeniowej (patrz podrozdział 2.4), która mówi o szacowaniu od góry (w tym przypadku przez $N^2$) zapotrzebowania na czas obliczeń w sytuacji najgorszego przypadku.

Algorytm uznamy za całkowicie niewrażliwy na uporządkowanie, ponieważ obie pętle wykonywać się będą niezależnie od początkowego uporządkowania elementów tablicy.

## 4.2. Sortowanie przez prostą zamianę (sortowanie bąbelkowe)

Drugim, bardzo popularnym algorytmem fizycznego sortowania jest algorytm sortowania przez prostą zamianę, znany jako sortowanie bąbelkowe. Rysunek 4.2 ułatwi zrozumienie działania tego algorytmu i jego własności. Jest to również algorytm typu *in situ*, całkowicie niewrażliwy na uporządkowanie, o złożoności obliczeniowej $O(N^2)$.




**RYSUNEK 4.2.** Algorytm sortowania bąbelkowego

W pierwszej kolumnie widzimy nieuporządkowaną tablicę ustawioną „pionowo". Wzdłuż tablicy przesuwa się indeks `i`, który sprawdza,

czy element poprzedzający element wskazywany przez indeks `i` jest większy od elementu wskazywanego przez `i`. Jeśli tak, dokonuje zamiany obu elementów, jak to widać na rysunku. W kolumnie drugiej widzimy efekt tej zamiany z indeksem `i` przesuniętym na kolejny element tablicy. Ostatnia kolumna przedstawia sytuację, w której indeks `i` doszedł w pętli do ostatniego elementu tablicy. Efektem tej operacji będzie „opadnięcie najcięższego bąbelka" (elementu o największej wartości) na dno, czyli do ostatniego elementu tablicy. Będzie się to wykonywać niezależnie od początkowego uporządkowania elementów.

Teraz całą opisaną wyżej operację trzeba powtórzyć, ale już dla nieco krótszego obszaru tablicy (nad pogrubioną linią). Musimy więc „opakować" pętlę, która przesuwała indeks `i`, inną pętlą zewnętrzną, która będzie nadzorować wykonanie pętli wewnętrznej, dopóki długość tablicy dla pętli wewnętrznej nie skróci się do jednego elementu.

**Ćwiczenie nr 9 do samodzielnego wykonania**

Zapisz iteracyjny algorytm w postaci funkcji, która sortuje rosnąco przez prostą zamianę elementów tablicy przekazanej do funkcji poprzez parametr.

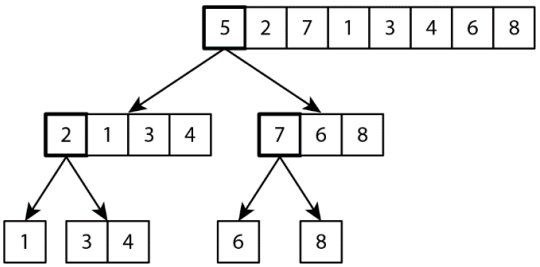
**Ćwiczenie nr 10 do samodzielnego wykonania**

Oceń i uzasadnij złożoność obliczeniową algorytmu sortowania bąbelkowego.

## 4.3. Sortowanie szybkie (QuickSort). Metoda „dziel i zwyciężaj"

Jest to algorytm oparty na zupełnie innym pomyśle w porównaniu z algorytmami omawianymi powyżej. Zastosowano tu (prezentowaną już przy okazji wyszukiwania binarnego) metodę dekompozycji problemu w wersji „dziel i zwyciężaj". Algorytm jest algorytmem rekurencyjnym.

Wersja rekurencyjna tego algorytmu charakteryzuje się (co jest typowe dla algorytmów rekurencyjnych) niesłychaną prostotą. Składa się z dwóch kroków.




**RYSUNEK 4.3.** Ilustracja idei algorytmu rekurencyjnego sortowania szybkiego QuickSort

W pierwszym kroku na każdym poziomie rekurencyjnego wywołania algorytm rozdziela tylko elementy coraz krótszych tablic na dwie tablice: lewą, zawierającą elementy mniejsze od tzw. elementu osiowego, i prawą — zawierającą elementy większe od elementu osiowego. Elementem osiowym może być wybrany dowolny element. Tutaj rolę tę pełni element pierwszy każdej nowo powstałej tablicy.

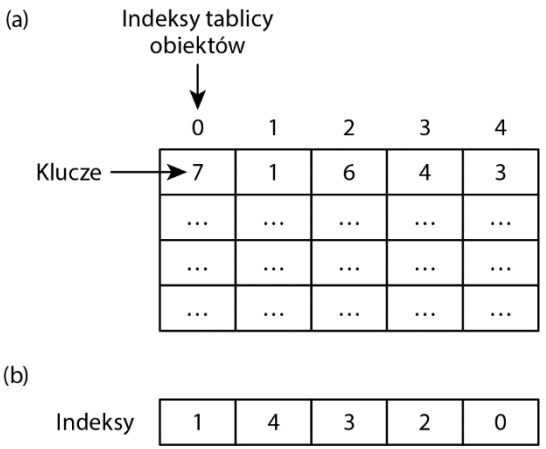
Drugim krokiem algorytmu jest rekurencyjne wywołanie samego siebie dla każdej nowo powstałej tablicy. Po rozdzieleniu tablicy na pojedyncze elementy bądź dwuelementowe uporządkowane już tablice algorytm wycofuje się, dokonując na każdym poziomie łączenia dwóch swoich podtablic w jedną. Rekurencja bezogonowa zapewnia automatyczne połączenie tablic cząstkowych z odpowiednimi włączeniami elementów osiowych.

Napisano wiele odmian tego algorytmu. Ich złożoność obliczeniowa wynosi $O(N \cdot \log_2 N)$. Algorytm QuickSort jest więc algorytmem znacznie efektywniejszym czasowo niż omówione poprzednio dwa algorytmy fizycznego sortowania (patrz rysunek 2.10).

Niwelowana jest też zajętość pamięciowa, ponieważ algorytm ten nie musi dokonywać fizycznych przemieszczeń elementów tablicy, tworząc nowe podtablice. W zamian zmieniane są tylko referencje do elementów i podtablic.

## 4.4. Sortowanie z użyciem dodatkowej tablicy

Elementami tablicy są często rekordy bądź obiekty klas (na rysunku 4.4 w owalu), z których każdy zawiera szereg pól. Jedno z nich jest zwykle wybierane jako klucz sortowania.




**RYSUNEK 4.4.** Tablica indeksowa b) powstała z posortowania wejściowej tablicy obiektów a) wg wybranego klucza

Zaprezentowana metoda sortowania sprowadza się do wytworzenia dodatkowej tablicy (zwanej **tablicą indeksową**), która przechowuje numery indeksów elementów tablicy oryginalnej, ale ustawione tak, że wybierając je kolejno, możemy według ich wartości wybierać obiekty z tablicy oryginalnej w kolejności rosnącego klucza.

W przytoczonym przykładzie pierwszy indeks tablicy indeksowej o wartości 1 pozwala wybrać obiekt tablicy oryginalnej o najmniejszej wartości klucza równej 1. Kolejny indeks z tablicy indeksowej o wartości 4 odpowiada obiektowi w tablicy oryginalnej o wartości klucza równej 3. Kolejne wartości indeksów tablicy indeksowej pozwolą wybierać kolejne obiekty z tablicy oryginalnej w porządku rosnącego klucza.

Metoda ta ma **minimalną złożoność pamięciową**, wymaga bowiem tylko użycia dodatkowej jednowymiarowej tablicy o rozmiarze tablicy oryginalnej, pozwalając zachować sortowaną tablicę w stanie pierwotnym. Wielką zaletą tej metody jest możliwość jednoczesnego generowania wielu tablic indeksowych wg różnych kluczy tablicy oryginalnej. Jednak jakakolwiek zmiana wartości kluczy tablicy oryginalnej pociąga za sobą konieczność generowania nowych tablic indeksowych.

### Ćwiczenie nr 11 do samodzielnego wykonania

Zapisz w pseudokodzie iteracyjny algorytm tworzenia tablicy indeksowej dla klucza obiektów. Oceń czasową złożoność obliczeniową algorytmu.

# Skorowidz

## Ł



## M



## N



## O



## P



## R

## S



## Ś



## T



## U



## V



## W



## Z


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

## To, co najważniejsze w programowaniu

Algorytmy to skończone ciągi jasno zdefiniowanych czynności, prowadzących do zrealizowania określonych zadań. Ten podręcznik skupia się na algorytmach imperatywnych (od łac. *impero* — rozkazywać) i wprowadza podstawowe pojęcia algorytmiki niezbędne do nauki programowania. Uczy projektowania, zapisywania i analizy poprawności, jak również podstaw szacowania złożoności czasowej i pamięciowej algorytmów. Co więcej, zamieszczono tu wiele zadań, których rozwiązanie zaowocuje lepszym zrozumieniem teorii i pogłębieniem umiejętności praktycznych. By jeszcze bardziej ułatwić pracę z podręcznikiem, zawarta w nim treść została bogato zilustrowana rysunkami poglądowymi i fragmentami kodów.

## Dzięki tej książce między innymi:

- **Poznasz szereg ważnych algorytmów, jak wyszukiwanie binarne, sortowanie szybkie, algorytmy klasy *dziel i zwyciężaj*, algorytmy zachłanne itd.**
- **Nauczysz się korzystać z używanych powszechnie w programowaniu struktur danych: tablic, słowników, list wiązanych, stosów, kolejek, drzew binarnych i grafów**
- **Dowiesz się, jak stosować iterację i rekurencję w programowaniu**
- **Opanujesz podstawy języka Java**

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
*Sięgnij po więcej!* ▶

ISBN 978-83-283-9530-5

9 788328 395305

Cena: 39,90 zł