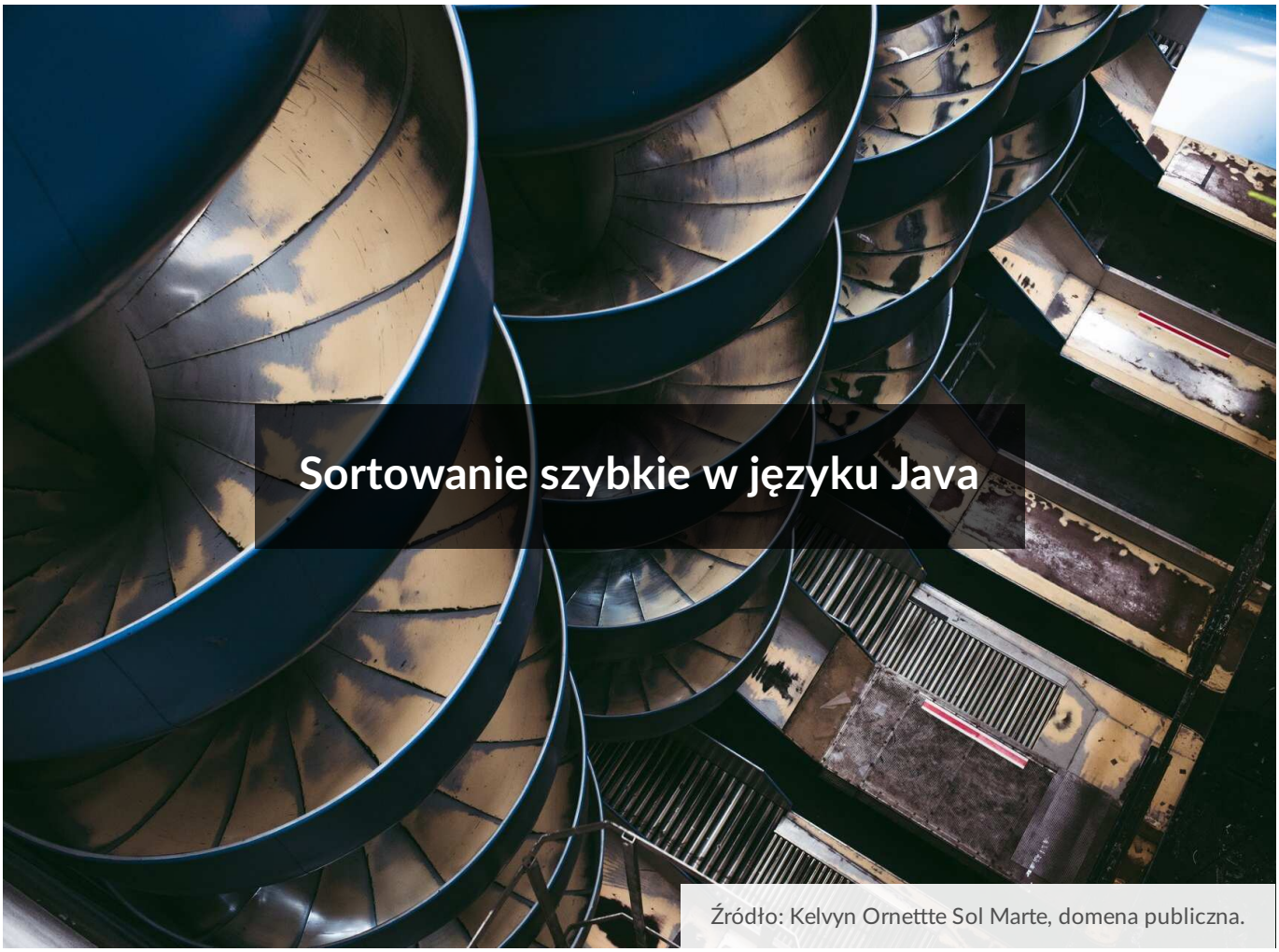
Sortowanie szybkie w języku Java



Źródło: Kelvyn Ornettte Sol Marte, domena publiczna.

Algorytmy różnią się od siebie złożonością czasową. Algorytm sortowania szybkiego, zaprezentowany w e-materiale Sortowanie szybkie, osiąga zadowalającą średnią złożoność wynoszącą $O(n \cdot log(n))$, w związku z czym wciąż dość powszechnie się go używa. Jak zaimplementować go w programie napisanym w języku Java?

Omówienie implementacji sortowania szybkiego w innych językach programowania znajdziesz w e-materiałach:

- Sortowanie szybkie w języku C++,
- Sortowanie szybkie w języku Python.

Więcej zadań? Znajdziesz je w e-materiale: Sortowanie szybkie – zadania maturalne.

Twoje cele

- Przeanalizujesz działanie algorytmu sortowania szybkiego.
- Przeprowadzisz analizę złożoności czasowej i pamięciowej algorytmu.
- Zaimplementujesz algorytm sortowania szybkiego, który uporządkuje zadany zbiór.
- Wykonasz kilka ćwiczeń z programowania w języku Java, związanych z algorytmem sortowania szybkiego.

Przeczytaj

Implementacja algorytmu w języku Java

Algorytm sortowania szybkiego oparty jest na dwóch funkcjach. Są to:

1. Funkcja wywoływana rekurencyjnie – nazwiemy ją `sortowanieSzybkie`. Nie będzie ona zwracała żadnej wartości, jej nazwa zostanie poprzedzona słowem kluczowym `void`. Funkcja ta będzie przyjmowała trzy parametry:
    - `int tab[]` – tablica zawierająca liczby całkowite do posortowania,
    - `int indeksPoczatkowy` – indeks elementu początkowego fragmentu tablicy, od którego chcemy rozpocząć sortowanie tablicy,
    - `int indeksKoncowy` – indeks końcowego elementu fragmentu tablicy, na którym chcemy zakończyć sortowanie tablicy.
2. Funkcja odpowiadająca za podział tablicy – wartością zwracaną przez funkcję będzie indeks miejsca podziału tablicy (indeks pivota). W naszym programie funkcja przyjmie nazwę `podzielTablice`. Parametry tej funkcji będą identyczne jak w przypadku funkcji `sortowanieSzybkie`.

Oto obie zadeklarowane funkcje:

```
1 void sortowanieSzybkie(int tab[], int indeksPoczatkowy, int indek
2
3 }
4
5 int podzielTablice(int tab[], int indeksPoczatkowy, int indeksKon
6
7 }
```

Zacznijmy od wypełnienia bloku funkcji `sortowanieSzybkie`. W instrukcji warunkowej sprawdzamy, czy podana jako argument tablica składa się z więcej niż jednego elementu. Jeżeli tak, następuje wywołanie funkcji podziału tablicy – `podzielTablice`. Do zmiennej `indeksPodzialu` zostanie zapisany punkt podziału tablicy (określony przez funkcję `podzielTablice`).

Po wywołaniu funkcji podziału następują dwa wywołania rekurencyjne funkcji `sortowanieSzybkie()`:

- `sortowanieSzybkie(tab, indeksPoczatkowy, indeksPodzialu - 1);` – dla pierwszej (lewej) części tablicy,

- `sortowanieSzybkie(tab, indeksPodzialu + 1, indeksKoncowy);` – dla drugiej (prawej) części tablicy.

Oto cały blok funkcji `sortowanieSzybkie()`:

```
static void sortowanieSzybkie(int tab[], int indeksPoczatkowy, in
{
    if(indeksPoczatkowy < indeksKoncowy) {
        int indeksPodzialu = podzielTablice(tab, indeksPoczatkowy
        sortowanieSzybkie(tab, indeksPoczatkowy, indeksPodzialu -
        sortowanieSzybkie(tab, indeksPodzialu + 1, indeksKoncowy)
    }
}
```

Kolejnym krokiem będzie przygotowanie funkcji `podzielTablice`.

Tworzymy zmienną `pivot`, do której zapisujemy wartość ostatniego elementu z sortowanego zakresu tablicy.

> **Ważne!**
>
> Pamiętajmy, że ten element możemy wybrać na inne sposoby, np. poprzez wskazanie elementu środkowego. Wybór pivota wraz ze sposobem uporządkowania tablicy mają znaczący wpływ na czas działania algorytmu. Jeżeli za każdym razem wybierany będzie element o wartości największej lub najmniejszej w tablicy, podział tablicy będzie nierównomierny. Jedna część tablicy będzie zawierała jeden element, natomiast druga część – wszystkie pozostałe. Ten pesymistyczny przypadek możemy otrzymać, gdy np. wybierzemy (pivot) o wartości ostatniego lub pierwszego elementu już posortowanej tablicy. Uzyskujemy wtedy kwadratową złożoność czasową algorytmu $O\left(n^2\right)$ i liniową złożoność pamięciową $O\left(n\right)$, spowodowaną dużą głębokością drzewa wywołań rekurencyjnych. W przypadku optymistycznym, tzn. jeżeli tablica będzie dzielona za każdym razem na dwie równe części (element osiowy będzie medianą z sortowanego fragmentu tablicy), złożoność czasowa algorytmu wyniesie $O\left(n \cdot \log n\right)$, natomiast pamięciowa – $O\left(\log n\right)$.

Następnym krokiem jest utworzenie zmiennej `indeksElementuMniejszegoOdPivota`, której ustawiamy wartość o 1 mniejszą od tej zapisanej w zmiennej `indeksPoczatkowy`. Będzie to iterator, spełniający dwa ważne zadania. Po pierwsze będzie wskazywał, na którą pozycję mamy przenieść element, gdy okaże się on mniejszy od pivota. Po drugie pomoże ustalić, w którym miejscu znajdują się elementy mniejsze od pivota. Tę zmienną wykorzystamy przy kolejnych podziałach tablicy.

Tworzymy pętlę `for`. Dla czytelności iterator sterujący pętlą określimy jako `indeksPoszukujacy`. Jego zadaniem będzie przejście po wszystkich elementach tablicy `tab`, które znajdują się w zadanym zakresie.

Wewnątrz pętli tworzymy instrukcję warunkową. Jeżeli wartość elementu `tab[indeksPoszukujacy]` jest mniejsza od wartości `pivot`, przenosimy ją na lewą stronę. Zwiększamy `indeksElementuMniejszegoOdPivota` i zamieniamy miejscami element znajdujący się w tablicy `tab[indeksElementuMniejszegoOdPivota]` z elementem `tab[indeksPoszukujacy]`. Pamiętajmy, że operacja ta jest powtarzana dla każdej komórki w tablicy, której indeks znajduje się w przedziale `<indeksPoczatkowy, indeksKoncowy>`.

Po przejściu pętli porządkujemy pivota – zamieniamy go miejscami z elementem wskazywanym przez `indeksElementuMniejszegoOdPivota + 1`.

Funkcję kończymy, zwracając indeks wskazujący na miejsce podziału: `indeksElementuMniejszegoOdPivota + 1`.

```
static int podzielTablice(int tab[], int indeksPoczatkowy, int in
{
    int pivot = tab[indeksKoncowy];
    int indeksElementuMniejszegoOdPivota = indeksPoczatkowy - 1;

    for(int indeksPoszukujacy = indeksPoczatkowy; indeksPoszukuja
        if(tab[indeksPoszukujacy] < pivot){
            indeksElementuMniejszegoOdPivota++;

            int temp = tab[indeksElementuMniejszegoOdPivota];
            tab[indeksElementuMniejszegoOdPivota] = tab[indeksPos
            tab[indeksPoszukujacy] = temp;
        }
    }

    int temp = tab[indeksElementuMniejszegoOdPivota + 1];
    tab[indeksElementuMniejszegoOdPivota + 1] = tab[indeksKoncowy
    tab[indeksKoncowy] = temp;

    return indeksElementuMniejszegoOdPivota + 1;
}
```

Przejdźmy teraz do głównej funkcji programu – `main`, w której utworzymy tablicę z liczbami całkowitymi, a następnie wywołamy funkcję `sortowanieSzybkie` w celu posortowania

elementów. Wypiszemy też elementy tablicy przed sortowaniem i po sortowaniu, żeby zobaczyć, czy zostało ono wykonane poprawnie.

```
public static void main(String[] args)
{
    int tab[] = {4, 5, 65, 7, 4, 0, 23, 34, 12, 1, 3, 4, 5};
    int ostatniElement = tab.length;

    for(int i = 0; i < ostatniElement; i++) {
        System.out.print(tab[i] + "  ");
    }

    System.out.println();

    sortowanieSzybkie(tab, 0, tab.length - 1);

    for(int i = 0; i < ostatniElement; i++) {
        System.out.print(tab[i] + "  ");
    }
}
```

Oto pełen kod programu:

```
public class SortowanieSzybkie
{

    static void sortowanieSzybkie(int tab[], int indeksPoczatkowy
    {
        if(indeksPoczatkowy < indeksKoncowy) {
            int indeksPodzialu = podzielTablice(tab, indeksPoczat
            sortowanieSzybkie(tab, indeksPoczatkowy, indeksPodzia
            sortowanieSzybkie(tab, indeksPodzialu + 1, indeksKonc
        }
    }

    static int podzielTablice(int tab[], int indeksPoczatkowy, in
    {
        int pivot = tab[indeksKoncowy];
        int indeksElementuMniejszegoOdPivota = indeksPoczatkowy -

```

```
18          for(int indeksPoszukujacy = indeksPoczatkowy; indeksPoszu
19              if(tab[indeksPoszukujacy] < pivot){
20                  indeksElementuMniejszegoOdPivota++;
21
22                  int temp = tab[indeksElementuMniejszegoOdPivota];
23                  tab[indeksElementuMniejszegoOdPivota] = tab[indek
24                  tab[indeksPoszukujacy] = temp;
25              }
26          }
27
28          int temp = tab[indeksElementuMniejszegoOdPivota + 1];
29          tab[indeksElementuMniejszegoOdPivota + 1] = tab[indeksKon
30          tab[indeksKoncowy] = temp;
31
32          return indeksElementuMniejszegoOdPivota + 1;
33      }
34
35      public static void main(String[] args)
36      {
37          int tab[] = {4, 5, 65, 7, 4, 0, 23, 34, 12, 1, 3, 4, 5};
38          int ostatniElement = tab.length;
39
40          for(int i = 0; i < ostatniElement; i++) {
41              System.out.print(tab[i] + "  ");
42          }
43
44          System.out.println();
45
46          sortowanieSzybkie(tab, 0, tab.length - 1);
47
48          for(int i = 0; i < ostatniElement; i++) {
49              System.out.print(tab[i] + "  ");
50          }
51      }
52  }
```

Słownik

> **pivot**
>
> element osiowy – element tablicy wybrany wedle ustalonego schematu; jest to element odpowiadający za sposób dzielenia tablicy

rekurencja

proces polegający na wywoływaniu funkcji przez siebie samą do momentu rozwiązania określonego problemu

wywołanie rekurencyjne

sytuacja, w której funkcja wywołuje samą siebie

zasada „dziel i zwyciężaj”

zasada często stosowana przez programistów przy tworzeniu algorytmów opartych o rekurencję, polegająca na tym, że jeden trudny, złożony problem dzielony jest na kilka mniejszych, prostszych do rozwiązania

złożoność czasowa

ilość czasu potrzebnego do wykonania zadania, wyrażona jako funkcja ilości danych

złożoność obliczeniowa

złożoność czasowa i złożoność pamięciowa; ilość zasobów komputerowych potrzebnych do wykonania zadania

Gra edukacyjna

Polecenie 1

Sprawdź swoją wiedzę, biorąc udział w grze.

Test

Test wiedzy o sortowaniu szybkim w języku Java.

Poziom trudności:

łatwy

Limit czasu:

8 min

Twój ostatni wynik:

-

Uruchom

Polecenie 2

Zastanów się, które pytania sprawiły ci trudność. Wróć do fragmentów materiału, których dotyczą.

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program, który za pomocą algorytmu sortowania szybkiego posortuje n-elementowy zbiór znaków `zbior`. Swoje rozwiązanie sprawdź dla zadanego zbioru:
`zbior = ['a', 'f', 'z', 'b', 'd', 't']` oraz `n = 6`.

Specyfikacja:

Dane:

- `n` – liczba naturalna, liczba elementów w zbiorze `zbior`
- `zbior[]` – n-elementowy zbiór znaków, małe litery alfabetu łacińskiego

Wynik:

Program wyświetla najpierw nieposortowane elementy zbioru (oddzielone spacją), a następnie (w nowej linii) elementy zbioru posortowane alfabetycznie.

Twoje zadania

1. Program ma wypisać elementy nieposortowanego zbioru `zbior` (kolejne elementy oddzielone spacją), a następnie w nowej linii elementy zbioru `zbior` po sortowaniu (kolejne elementy oddzielone spacją).

Ćwiczenie 2

Przedstawiony kod jest implementacją algorytmu sortowania szybkiego w języku Java. Dla trzech zadanych n-elementowych tablic `tab1`, `tab2`, `tab3` napisz kod odpowiadający za obliczenie głębokości drzewa wywołań rekurencyjnych funkcji `sortuj()`. Program na standardowe wyjście ma wypisać trzy głębokości drzew wywołań funkcji `sortuj()`, przy sortowaniu tablic `tab1`, `tab2`, `tab3`, każda wartość w nowej linii. Swój program przetestuj dla `n = 16` i trzech tablic:

- tablica z losowym rozkładem elementów
  `tab1 = {12, 1, 3, 15, 14, 2, 11, 4, 6, 5, 13, 10, 7, 9, 16, 8}`
- tablica posortowana rosnąco
  `tab2 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}`
- tablica posortowana malejąco
  `tab3 = {16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1}`

Specyfikacja:

Dane:

- n – liczba naturalna, liczba elementów w tablicach `tab1`, `tab2` i `tab3`
- `tab1[], tab2[], tab3[]` – tablice liczb naturalnych

Wynik:

Program na standardowym wyjściu wypisuje trzy głębokości drzew wywołań rekurencyjnych funkcji `sortuj()`, przy sortowaniu trzech n-elementowych tablic `tab1`, `tab2`, `tab3`.

Twoje zadania

1. Program wypisuje trzy głębokości drzew wywołań rekurencyjnych funkcji `sortuj()` uzyskane w trakcie sortowania trzech n-elementowych tablic `tab1`, `tab2`, `tab3`

Ćwiczenie 3

Zapytano kilka osób o miesięczne zarobki, a ich odpowiedzi zebrano w n-elementowej tabeli `zarobki`. Użyj algorytmu sortowania szybkiego, aby znaleźć medianę (wartość środkową) zarobków w tej grupie.

Swoje rozwiązanie przetestuj dla następujących odpowiedzi ankietowanych: `zarobki = {8500, 6400, 2800, 3500, 12870, 3300, 7020, 3000, 8100}` o raz dla `n = 9`.

Specyfikacja:

Dane:

- n – liczba naturalna, liczba ankietowanych
- `zarobki[]` – tablica liczb naturalnych, przechowująca zarobki ankietowanych

Wynik:

Program wyświetla medianę (wartość środkowa, rzeczywista) zarobków w grupie

Twoje zadania

1. Program powinien wyświetlić jedną liczbę – medianę wartości zbioru.

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie szybkie w języku Java

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

c) metodę dziel i zwyciężaj (jednoczesne znajdowanie minimum i maksimum, sortowanie przez scalanie i szybkie),

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Przeanalizujesz działanie algorytmu sortowania szybkiego.

- Przeprowadzisz analizę złożoności czasowej i pamięciowej algorytmu.
- Zaimplementujesz algorytm sortowania szybkiego, który uporządkuje zadany zbiór.
- Wykonasz kilka ćwiczeń z programowania w języku Java, związanych z algorytmem sortowania szybkiego.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Sortowanie szybkie w języku Java". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Prowadzący wyświetla na tablicy interaktywnej zawartość sekcji „Wprowadzenie" i omawia cele do osiągnięcia w trakcie lekcji.

Faza realizacyjna:

1. Uczniowie analizują przykład z sekcji „Przeczytaj" i powtarzają zaprezentowane rozwiązanie na swoim komputerze.

2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Gra edukacyjna". Uczniowie odpowiadają na 15 pytań i sprawdzają swoją wiedzę i umiejętności z zakresu sortowania szybkiego.
3. **Ćwiczenie umiejętności.** Prowadzący zapowiada uczniom, że będą rozwiązywać ćwiczenie nr 1 z sekcji „Sprawdź się". Każdy z uczniów robi to samodzielnie. Po ustalonym czasie następuje porównanie napisanych kodów podczas wspólnego omówienia rozwiązań.

Faza podsumowująca:

1. Nauczyciel zadaje pytania podsumowujące, np.
   – czym jest sortowanie szybkie i na jakiej metodzie się opiera?
   – jak nazywamy sytuację, w której funkcja wywołuje samą siebie?
   – co oznacza pojęcie „pętla zagnieżdżona"?
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń z programowania w języku Java.

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 2 z sekcji „Sprawdź się".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Java SE 8 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Eclipse 4.4 (lub nowszej wersji).

Wskazówki metodyczne:

- Treści w sekcji „Przeczytaj" można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.