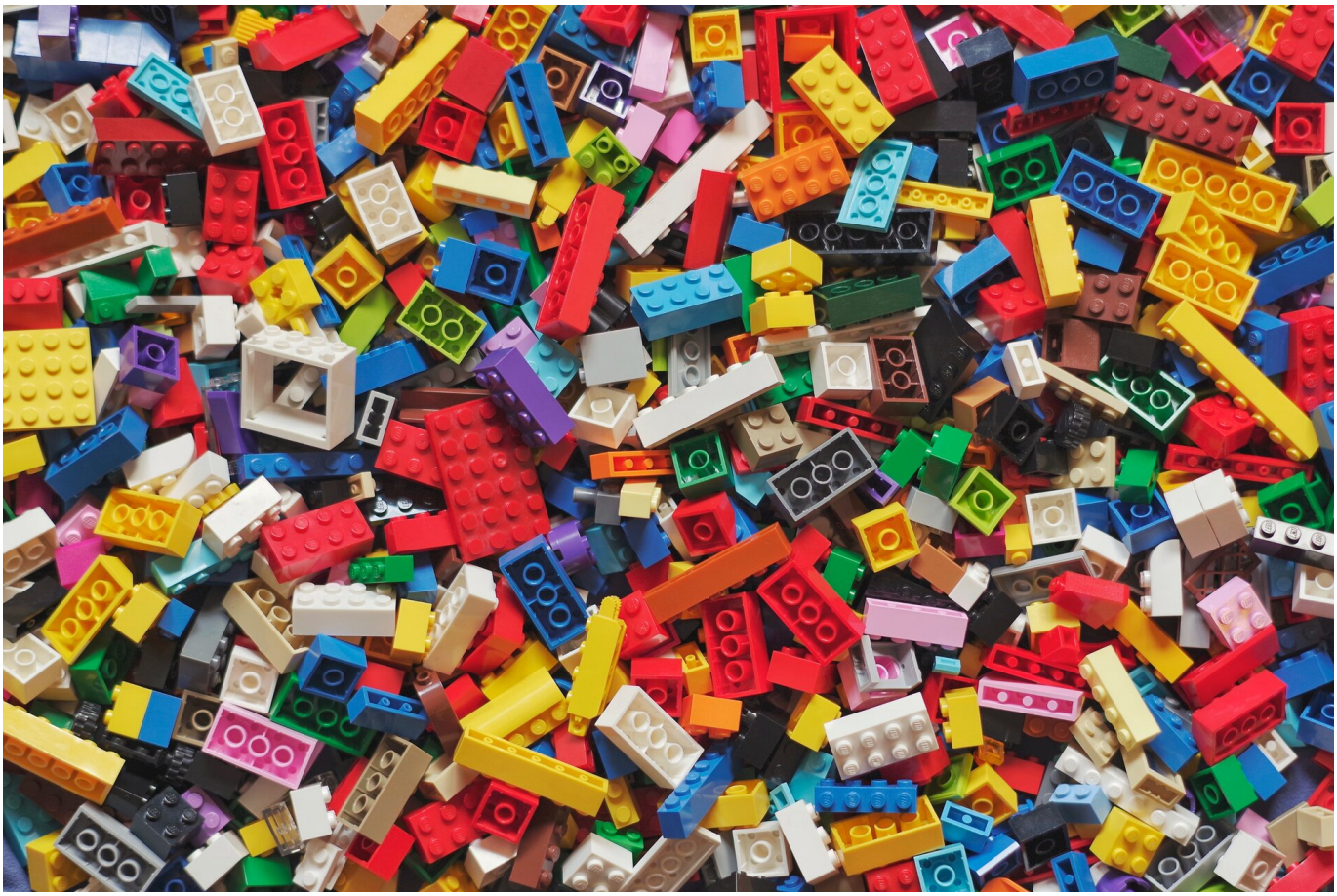
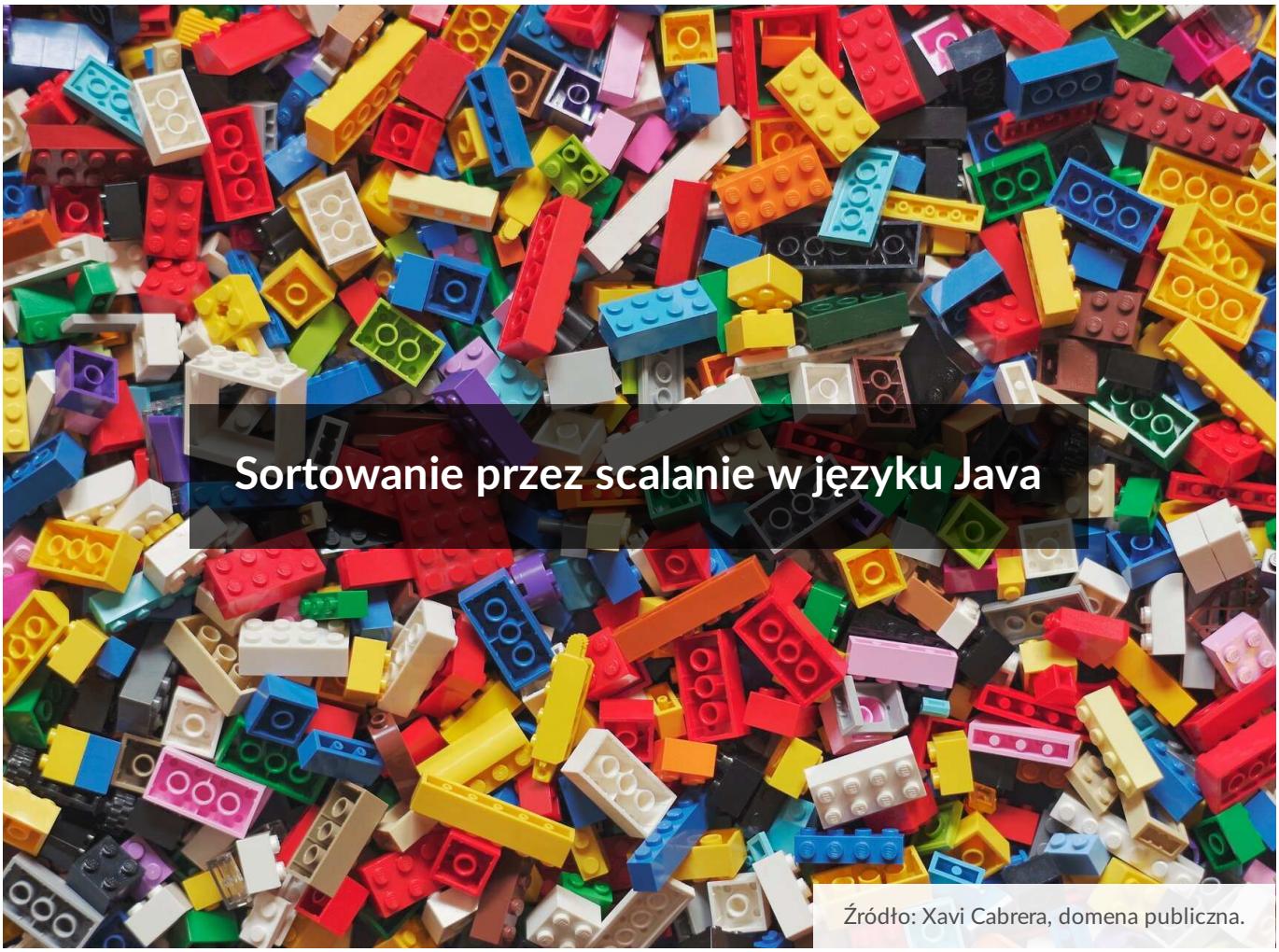


Sortowanie przez scalanie w języku Java



Źródło: Xavi Cabrera, domena publiczna.

Poznaliśmy już rekurencyjny algorytm sortowania przez scalanie wykorzystujący metodę „dziel i zwyciężaj". Jest on jednym z najszybszych sposobów sortowania.

W tym e-materiale zajmiemy się implementacją algorytmu *merge sort* w języku Java.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych lekcjach z tej serii:

- Sortowanie przez scalanie w języku C++,
- Sortowanie przez scalanie w języku Python.

Więcej zadań? Sięgnij do: Sortowanie przez scalanie – zadania maturalne.

Twoje cele

- Wyjaśnisz, czym jest rekurencja oraz metoda „dziel i zwyciężaj".
- Przeanalizujesz, jak działa algorytm sortowania przez scalanie.
- Napiszesz program wykorzystujący algorytm sortowania przez scalanie oraz rozwiążesz kilka zadań z zastosowaniem tego algorytmu.

Film samouczek

Polecenie 1

Napisz program niemalejąco sortujący wyniki ankiety dotyczącej równości w życiu społecznym zrealizowanej przez CBOS w 2000 roku. Wykorzystaj w tym celu sortowanie przez scalanie (*merge sort*).

Procentowe wyniki ankiety „Czy pana/pani zdaniem równość w społeczeństwie powinna czy też nie powinna oznaczać, że wszyscy obywatele mają równe szanse zdobycia wykształcenia i osiągnięcia wysokiej pozycji społecznej, o ile mają odpowiednie zdolności i chęci?", zrealizowanej w lutym 2000 roku przez CBOS:

- 67% odpowiedziało: „Zdecydowanie tak".
- 21% odpowiedziało: „Raczej tak".
- 2% odpowiedziało: „Trudno powiedzieć".
- 3% odpowiedziało: „Zdecydowanie nie".
- 7% odpowiedziało: „Raczej nie".

Specyfikacja problemu:

Dane:

- `dane[ ]` – tablica liczb naturalnych

Wynik:

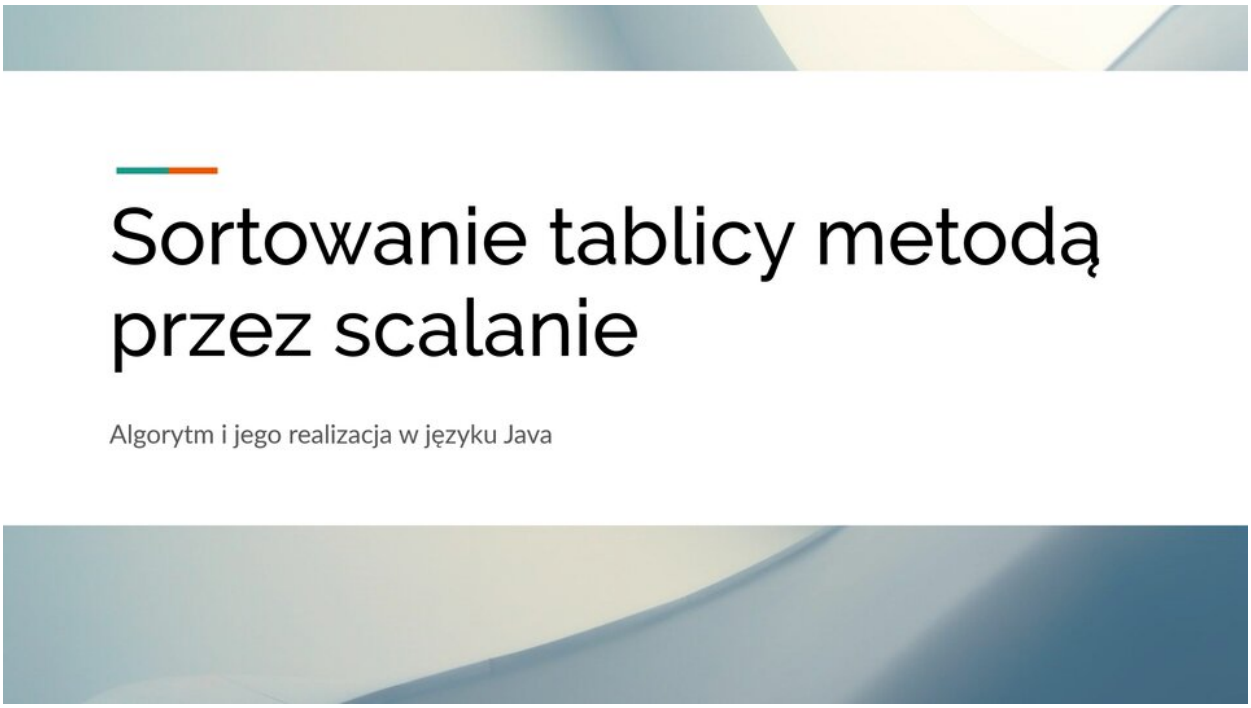
- Na standardowym wyjściu wyświetlane są wszystkie elementy tablicy `dane[ ]` posortowanej w kolejności niemalejącej, oddzielone pojedynczym znakiem spacji.

Twoje zadania

1. Wypisz elementy tablicy posortowane niemalejąco za pomocą metody sortowania przez scalanie.

Polecenie 2

Porównaj swoje rozwiązanie z przedstawionym w filmie.



Film dostępny pod adresem /preview/resource/RN7AQ0YePKo2J

Film nawiązujący do treści materiału: Sortowanie tablicy metodą przez scalanie.

Kod programu zaprezentowanego w filmie:

Plik o rozmiarze 1.44 KB w języku polskim

Przeczytaj

Implementacja algorytmu sortowania przez scalanie w języku Java

Specyfikacja problemu

Dane:

- `tablica` – tablica liczb naturalnych do posortowania
- `indeksPoczatku` – liczba naturalna
- `indeksKonca` – liczba naturalna

Wynik:

- `tablica` – posortowana nierosnąco tablica liczb naturalnych

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

Implementację zaczniemy od stworzenia funkcji rekurencyjnej `sortujPrzezScalenie`, która jako argumenty przyjmuje tablicę do posortowania, indeks początku oraz indeks końca części, która właśnie jest analizowana.

```
1 public static void
  sortujPrzezScalenie(int
  tablica[], int
  indeksPoczatku, int
  indeksKonca) {
2
3 }
```

Warunkiem kończącym wywoływanie rekurencyjne kolejnych instancji funkcji jest to, że w badanej części tablicy jest tylko jeden element. W takiej sytuacji kończymy działanie funkcji.

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

Jeżeli w badanej części będzie tylko jeden element, `indeksPoczatku` będzie równy `indeksKonca`.

```
1 public static void
  sortujPrzezScalenie(int
  tablica[], int
  indeksPoczatku, int
  indeksKonca) {
2   if (indeksPoczatku <
  indeksKonca) {
3     // Wykonuj kroki
  algorytmu
4   } else {
5     return;
6   }
7 }
```

Jeżeli w części tablicy znajduje się więcej niż jeden element, wykonujemy wcześniej wspomniane cztery kroki algorytmu.

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

Te kroki to: podział na dwie części, podział lewej części, podział prawej części oraz ich złączenie.

```
1 public static void
  sortujPrzezScalenie(int
  tablica[], int
  indeksPoczatku, int
  indeksKonca) {
2
```

```
  if (indeksPoczatku < indeksKonca) {
    // Krok 1 - podziel tablice na dwie części
    int srodkowyIndeks = (indeksKonca + indeksPoczatku) / 2;

    // Krok 2 - podziel lewą część
 sortujPrzezScalenie(tablica, indeksPoczatku, srodkowyIndeks);

    // Krok 3 - podziel prawa część
 sortujPrzezScalenie(tablica, srodkowyIndeks + 1, indeksKonca);

    // Krok 4 - scal dwie części
 scalDwieCzesci(tablica, indeksPoczatku, srodkowyIndeks, indeksKonca);
  } else {
    return;
  }
}
```

Żeby ułatwić łączenie części, stworzymy dla niego osobną funkcję `scalDwieCzesci`.

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

Zadeklarujemy w związku z tym funkcję `scalDwieCzesci` w następujący sposób:

```
1 public static void
  scalDwieCzesci(int
  tablica[], int
  indeksPoczatku, int
  srodkowyIndeks, int
  indeksKonca) {
2
3 }
```

Zmienna `indeksPoczatku` odpowiada pierwszemu indeksowi lewej części, natomiast `srodkowyIndeks` odpowiada ostatniemu indeksowi lewej części. `srodkowyIndeks + 1` to pierwszy indeks prawej części, a `indeksKonca` to ostatni indeks prawej części.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

5

Aby swobodnie zamieniać elementy miejscami, stworzymy kopie lewej oraz prawej części tablicy. Dzięki kopiowaniu nie będziemy musieli martwić się o nadpisanie danych w tablicy. Stworzenie kopii wymaga określenia rozmiaru lewej oraz prawej części.

```
1 public static void
  scalDwieCzesci(int
  tablica[], int
  indeksPoczatku, int
  srodkowyIndeks, int
  indeksKonca) {
2   int rozmiarLewejCzesci =
  srodkowyIndeks -
  indeksPoczatku  + 1;
3   int rozmiarPrawejCzesci
  = indeksKonca -
```

```
    srodkowyIndeks;
4 }
```

Ponieważ element o indeksie `srodkowyIndeks` należy do lewej części, dodajemy jeden do różnicy indeksów.

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

Znając już rozmiar lewej oraz prawej części możemy zainicjować tablice, które będą przechowywać kopie elementów znajdujących się w tych właśnie częściach tablicy.

```
1 public static void
  scalDwieCzesci(int
  tablica[], int
  indeksPoczatku, int
  srodkowyIndeks, int
  indeksKonca) {
2   int rozmiarLewejCzesci =
  srodkowyIndeks -
  indeksPoczatku  + 1;
3   int rozmiarPrawejCzesci
  = indeksKonca -
  srodkowyIndeks;
4
5   int kopiaLewejCzesci[] =
  new
  int[rozmiarLewejCzesci];
6   int kopiaPrawejCzesci[]
  = new
  int[rozmiarPrawejCzesci];
7 }
```

Kolejnym krokiem będzie wypełnienie tablic wartościami z lewej oraz prawej części.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

Wypełniamy więc nowe tablice wartościami z odpowiednich indeksów z tablicy.

```
1  public static void
   scalDwieCzesci(int
   tablica[], int
   indeksPoczatku, int
   srodkowyIndeks, int
   indeksKonca) {
2    int rozmiarLewejCzesci =
   srodkowyIndeks -
   indeksPoczatku  + 1;
3    int rozmiarPrawejCzesci
   = indeksKonca -
   srodkowyIndeks;
4
5    int kopiaLewejCzesci[] =
   new
   int[rozmiarLewejCzesci];
6    int kopiaPrawejCzesci[]
   = new
   int[rozmiarPrawejCzesci];
7
8    for (int i = 0; i <
   rozmiarLewejCzesci; i++) {
9      kopiaLewejCzesci[i] =
   tablica[indeksPoczatku +
   i];
10   }
11   for (int i = 0; i <
   rozmiarPrawejCzesci; i++)
   {
12     kopiaPrawejCzesci[i] =
   tablica[srodkowyIndeks + i
   + 1];
13   }
14 }
```

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

Zanim przejdziemy do scalania, musimy zainicjować kilka zmiennych pomocniczych.

```
1  public static void scalDwieCzesci(int tablica[], int indeksPoczatku, int srodkowyIndeks, int indeksKonca) {
2    int rozmiarLewejCzesci = srodkowyIndeks - indeksPoczatku  + 1;
3    int rozmiarPrawejCzesci = indeksKonca - srodkowyIndeks;
4
5    int kopiaLewejCzesci[] = new int[rozmiarLewejCzesci];
6    int kopiaPrawejCzesci[] = new int[rozmiarPrawejCzesci];
7
8    for (int i = 0; i < rozmiarLewejCzesci; i++) {
9      kopiaLewejCzesci[i] = tablica[indeksPoczatku + i];
10   }
11   for (int i = 0; i < rozmiarPrawejCzesci; i++) {
12     kopiaPrawejCzesci[i] = tablica[srodkowyIndeks + i + 1];
13   }
14
15   int biezacyIndeksLewejCzesci = 0,
```

```
    biezacyIndeksPrawejCzesci
    = 0;
16    int
    biezacyIndeksWOryginalnejT
    ablicy = indeksPoczatku;
17  }
```

`biezacyIndeksLewejCzesci` oraz `biezacyIndeksPrawejCzesci` będzie odpowiedzialny za przechowywanie bieżącego indeksu każdej z części. Jeżeli przeniesiemy element z lewej części, zmienna `biezacyIndeksLewejCzesci` zostanie powiększona o jeden i będzie wskazywać na pierwszy nieprzeniesiony jeszcze element lewej części. Natomiast `biezacyIndeksWOryginalnejTablicy` wskazuje miejsce w oryginalnej tablicy, gdzie zostanie umieszczony kolejny element.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

9

Mając już kopie tablic oraz zmienne pomocnicze, możemy przejść do scalania części. Zaczniemy od porównywania elementów dwóch części. Dopóki obie części mają przynajmniej po jednym elemencie, porównujemy je i przenosimy większy.

```
1  public static void
   scalDwieCzesci(int
   tablica[], int
   indeksPoczatku, int
   srodkowyIndeks, int
   indeksKonca) {
2    int rozmiarLewejCzesci =
   srodkowyIndeks -
   indeksPoczatku  + 1;
3    int rozmiarPrawejCzesci
   = indeksKonca -
```

```
srodkowyIndeks;
4
5   int kopiaLewejCzesci[] =
new
int[rozmiarLewejCzesci];
6   int kopiaPrawejCzesci[]
= new
int[rozmiarPrawejCzesci];
7
8   for (int i = 0; i <
rozmiarLewejCzesci; i++) {
9     kopiaLewejCzesci[i] =
tablica[indeksPoczatku +
i];
10  }
11  for (int i = 0; i <
rozmiarPrawejCzesci; i++)
{
12    kopiaPrawejCzesci[i] =
tablica[srodkowyIndeks + i
+ 1];
13  }
14
15  int
biezacyIndeksLewejCzesci =
0,
biezacyIndeksPrawejCzesci
= 0;
16  int
biezacyIndeksWOryginalnejT
ablicy = indeksPoczatku;
17
18  while
(biezacyIndeksLewejCzesci
< rozmiarLewejCzesci &&
biezacyIndeksPrawejCzesci
< rozmiarPrawejCzesci) {
19    // Wybieraj wiekszy
element
20  }
21 }
```

Teraz musimy sprawdzić, czy pierwszy element lewej części jest większy lub równy prawemu. Jeżeli jest, to pierwszy element z lewej części

zostaje przeniesiony na miejsce `biezacyIndeksWOryginalnejTablicy`, w przeciwnym wypadku trafia tam pierwszy element prawej części.

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

Scalanie jest już gotowe:

```
1  public static void
   scalDwieCzesci(int
   tablica[], int
   indeksPoczatku, int
   srodkowyIndeks, int
   indeksKonca) {
2    int rozmiarLewejCzesci =
   srodkowyIndeks -
   indeksPoczatku  + 1;
3    int rozmiarPrawejCzesci
   = indeksKonca -
   srodkowyIndeks;
4
5    int kopiaLewejCzesci[] =
   new
   int[rozmiarLewejCzesci];
6    int kopiaPrawejCzesci[]
   = new
   int[rozmiarPrawejCzesci];
7
8    for (int i = 0; i <
   rozmiarLewejCzesci; i++) {
9      kopiaLewejCzesci[i] =
   tablica[indeksPoczatku +
   i];
10   }
11   for (int i = 0; i <
   rozmiarPrawejCzesci; i++)
   {
12     kopiaPrawejCzesci[i] =
   tablica[srodkowyIndeks + i
```

```
+ 1];
  }

  int
biezacyIndeksLewejCzesci =
0,
biezacyIndeksPrawejCzesci
= 0;
  int
biezacyIndeksWOryginalnejT
ablicy = indeksPoczatku;

  while
(biezacyIndeksLewejCzesci
< rozmiarLewejCzesci &&
biezacyIndeksPrawejCzesci
< rozmiarPrawejCzesci) {
    if
(kopiaLewejCzesci[biezacyI
ndeksLewejCzesci] >=
kopiaPrawejCzesci[biezacyI
ndeksPrawejCzesci]) {

 tablica[biezacyIndeksWOry
ginalnejTablicy] =
kopiaLewejCzesci[biezacyIn
deksLewejCzesci];

 biezacyIndeksLewejCzesci+
+;
    } else {

 tablica[biezacyIndeksWOry
ginalnejTablicy] =
kopiaPrawejCzesci[biezacyI
ndeksPrawejCzesci];

 biezacyIndeksPrawejCzesci
++;
    }

 biezacyIndeksWOryginalnej
Tablicy++;
  }
}
```

Jeżeli w którejś części skończą się elementy, wszystkie elementy z drugiej części zostają przeniesione w takiej kolejności, w jakiej są zapisane.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

11

```
1  public static void
   scalDwieCzesci(int
   tablica[], int
   indeksPoczatku, int
   srodkowyIndeks, int
   indeksKonca) {
2    int rozmiarLewejCzesci =
   srodkowyIndeks -
   indeksPoczatku  + 1;
3    int rozmiarPrawejCzesci
   = indeksKonca -
   srodkowyIndeks;
4
5    int kopiaLewejCzesci[] =
   new
   int[rozmiarLewejCzesci];
6    int kopiaPrawejCzesci[]
   = new
   int[rozmiarPrawejCzesci];
7
8    for (int i = 0; i <
   rozmiarLewejCzesci; i++) {
9      kopiaLewejCzesci[i] =
   tablica[indeksPoczatku +
   i];
10   }
11   for (int i = 0; i <
   rozmiarPrawejCzesci; i++)
   {
12     kopiaPrawejCzesci[i] =
   tablica[srodkowyIndeks + i
   + 1];
```

```
13    }
14
15    int
biezacyIndeksLewejCzesci =
0,
biezacyIndeksPrawejCzesci
= 0;
16    int
biezacyIndeksWOryginalnejT
ablicy = indeksPoczatku;
17
18    while
(biezacyIndeksLewejCzesci
< rozmiarLewejCzesci &&
biezacyIndeksPrawejCzesci
< rozmiarPrawejCzesci) {
19      if
(kopiaLewejCzesci[biezacyI
ndeksLewejCzesci] >=
kopiaPrawejCzesci[biezacyI
ndeksPrawejCzesci]) {
20
 tablica[biezacyIndeksWOry
ginalnejTablicy] =
kopiaLewejCzesci[biezacyIn
deksLewejCzesci];
21
 biezacyIndeksLewejCzesci+
+;
22      } else {
23
 tablica[biezacyIndeksWOry
ginalnejTablicy] =
kopiaPrawejCzesci[biezacyI
ndeksPrawejCzesci];
24
 biezacyIndeksPrawejCzesci
++;
25      }
26
 biezacyIndeksWOryginalnej
Tablicy++;
27    }
28
```

```
29   while
(biezacyIndeksLewejCzesci
< rozmiarLewejCzesci) {
30
 tablica[biezacyIndeksWOry
ginalnejTablicy] =
kopiaLewejCzesci[biezacyIn
deksLewejCzesci];
31
 biezacyIndeksLewejCzesci+
+;
32
 biezacyIndeksWOryginalnej
Tablicy++;
33   }
34
35   while
(biezacyIndeksPrawejCzesci
< rozmiarPrawejCzesci) {
36
 tablica[biezacyIndeksWOry
ginalnejTablicy] =
kopiaPrawejCzesci[biezacyI
ndeksPrawejCzesci];
37
 biezacyIndeksPrawejCzesci
++;
38
 biezacyIndeksWOryginalnej
Tablicy++;
39   }
40 }
```

Wywołajmy więc naszą funkcję i sprawdźmy jej działanie.

12

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PYL73e33E

Kompletny program znajduje się poniżej:

```java
public class Main {
  public static void main(String args[]) {
    int tablica[] = {8, 5, 2, 1, 6};


 sortujPrzezScalenie(tablica, 0, tablica.length - 1);

    for (int i = 0; i < tablica.length; i++) {

 System.out.print(tablica[i] + " ");
    }
  }

  public static void sortujPrzezScalenie(int tablica[], int indeksPoczatku, int indeksKonca) {

    if (indeksPoczatku < indeksKonca) {
      // Krok 1 - podziel tablice na dwie części
      int srodkowyIndeks = (indeksKonca + indeksPoczatku) / 2;

      // Krok 2 - podziel lewą część

 sortujPrzezScalenie(tablica, indeksPoczatku, srodkowyIndeks);

      // Krok 3 - podziel prawa część

 sortujPrzezScalenie(tabli
```

```
ca, srodkowyIndeks + 1,
indeksKonca);
23
24        // Krok 4 - scal
dwie części
25
 scalDwieCzesci(tablica,
indeksPoczatku,
srodkowyIndeks,
indeksKonca);
26      } else {
27        return;
28      }
29    }
30
31    public static void
scalDwieCzesci(int
tablica[], int
indeksPoczatku, int
srodkowyIndeks, int
indeksKonca) {
32      int rozmiarLewejCzesci
= srodkowyIndeks -
indeksPoczatku  + 1;
33      int
rozmiarPrawejCzesci =
indeksKonca -
srodkowyIndeks;
34
35      int kopiaLewejCzesci[]
= new
int[rozmiarLewejCzesci];
36      int
kopiaPrawejCzesci[] = new
int[rozmiarPrawejCzesci];
37
38      for (int i = 0; i <
rozmiarLewejCzesci; i++) {
39        kopiaLewejCzesci[i]
= tablica[indeksPoczatku +
i];
40      }
41      for (int i = 0; i <
rozmiarPrawejCzesci; i++)
{
```

```
42      kopiaPrawejCzesci[i] = tablica[srodkowyIndeks + i + 1];
43    }
44
45    int biezacyIndeksLewejCzesci = 0, biezacyIndeksPrawejCzesci = 0;
46    int biezacyIndeksWOryginalnejTablicy = indeksPoczatku;
47
48
49    while (biezacyIndeksLewejCzesci < rozmiarLewejCzesci && biezacyIndeksPrawejCzesci < rozmiarPrawejCzesci) {
50      if (kopiaLewejCzesci[biezacyIndeksLewejCzesci] >= kopiaPrawejCzesci[biezacyIndeksPrawejCzesci]) {
51
 tablica[biezacyIndeksWOryginalnejTablicy] = kopiaLewejCzesci[biezacyIndeksLewejCzesci];
52
 biezacyIndeksLewejCzesci++;
53      } else {
54
 tablica[biezacyIndeksWOryginalnejTablicy] = kopiaPrawejCzesci[biezacyIndeksPrawejCzesci];
55
 biezacyIndeksPrawejCzesci++;
56      }
57
 biezacyIndeksWOryginalnej
```

```
Tablicy++;
58      }
59
60      while
(biezacyIndeksLewejCzesci
< rozmiarLewejCzesci) {
61
 tablica[biezacyIndeksWOry
ginalnejTablicy] =
kopiaLewejCzesci[biezacyIn
deksLewejCzesci];
62
 biezacyIndeksLewejCzesci+
+;
63
 biezacyIndeksWOryginalnej
Tablicy++;
64      }
65
66      while
(biezacyIndeksPrawejCzesci
< rozmiarPrawejCzesci) {
67
 tablica[biezacyIndeksWOry
ginalnejTablicy] =
kopiaPrawejCzesci[biezacyI
ndeksPrawejCzesci];
68
 biezacyIndeksPrawejCzesci
++;
69
 biezacyIndeksWOryginalnej
Tablicy++;
70      }
71    }
72  }
```


Słownik

dzielenie całkowite

wynik dzielenia, który jest zawsze zaokrąglany w dół do najbliższej liczby całkowitej

sortowanie przez scalanie

podzielenie sortowanej struktury danych na dwie części, następnie na rekurencyjnym sortowaniu każdej z tych części i ponownym scaleniu już posortowanych części w jedną całość

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program, który łączy dwie tablice (`tablica1` oraz `tablica2`) w jedną, a następnie wypisuje jej zawartość. Połączenie tablic następuje w taki sposób, że zawartość `tablica2` zostaje przyklejona na końcu `tablica1`. Przetestuj jego działanie dla następujących tablic:

```
int tablica1 = {4, 65, 7654, 543, 654, 432}
int tablica2 = {2, 99, 75, 45, 65, 76, 411}
```

Specyfikacja:

Dane:

- n – liczba naturalna
- `m` – liczba naturalna
- `tablica1[]` – n-elementowa posortowana niemalejąco tablica liczb naturalnych
- `tablica2[]` – `m`-elementowa posortowana niemalejąco tablica liczb naturalnych

Wynik:

- `tablica3[]` – tablica zawierająca elementy z `tablica1[]` oraz `tablica2[]`
- Na standardowym wyjściu wyświetlane są wszystkie elementy tablicy `tablica3[]`, każdy w nowej linii.

Twoje zadania

1. Program łączy dwie tablice w jedną.

```
public class Main {
    public static void main (String [] args) {
        int tablica1 = {4, 65, 7654, 543, 654, 432};
        int tablica2 = {2, 99, 75, 45, 65, 76, 411};
```

```
5
6           // Tutaj dodaj kod.
7           // Do wypisywania, użyj funkcji: System.out.println();
8
9       }
10  }
```

```
1
```

Ćwiczenie 2

Napisz program, który na podstawie tablicy `zbior` stworzy nową tablicę i nazwie ją `kopiaZbioru`. Program powinien skopiować tablicę, zaczynając od elementu oznaczonego indeksem a i kończąc na elemencie o innym indeksie b. Przetestuj działanie programu dla indeksu początkowego 3 oraz indeksu końcowego, który jest o 3 mniejszy niż długość tablicy.

Przetestuj jego działanie dla tablicy:

```
1 int zbior[16] = { 43, 6, 76, 87, 54, 654, 432, 657, 765, 543, 6
```

Nowa tablica powinna składać się ze wszystkich elementów poza trzema pierwszymi oraz trzema ostatnimi.

Specyfikacja:

Dane:

- n – liczba naturalna
- a – liczba naturalna
- b – liczba naturalna
- `zbior[]` – n-elementowa tablica liczb całkowitych

Wynik:

Na standardowym wyjściu wyświetlane są wszystkie elementy tablicy `kopiaZbioru[]`, każdy w nowej linii.

Twoje zadania

1. Program tworzy kopię zbioru, zaczynając od indeksu a, a kończąc na indeksie b.

```java
public class Main {
    public static void main (String [] args) {
        int zbior[] = {43, 6, 76, 87, 54, 654, 432, 657, 765, 543, 654, 7657, 987, 4, 43, 65};

        // Tutaj dodaj kod.
        // Do wypisywania, użyj poniższej linijki
        // for (int i = 0; i < kopiaZbioru; i++) { System.out.println(kopiaZbioru[i]); }

    }
}
```

```
1
```

Ćwiczenie 3

Napisz program, który sortuje nierosnąco daną tablicę `tablica`, używając algorytmu sortowania przez scalanie. Przetestuj działanie programu dla następującej tablicy:

```
int tablica[] = {4, 656, 65, 43, 65, 876, 543, 756, 435, 654, 5
```

Specyfikacja:

Dane:

- n – liczba naturalna
- `tablica[]` – n-elementowa tablica liczb naturalnych

Wynik:

- Na standardowym wyjściu wyświetlane są posortowane nierosnąco elementy tablicy `tablica[]` oddzielone znakiem spacji.

Twoje zadania

1. Program sortuje tablicę, używając algorytmu sortowania przez scalanie.

```
public class Main {
    public static void main (String [] args) {
        int tablica[] = {4, 656, 65, 43, 65, 876, 543, 756, 435, 654, 534, 234, 756, 432, 765, 543, 987, 423, 123, 423, 654, 756, 534, 6755, 534, 4756, 65, 4, 654, 54, 75};

        // Tutaj dodaj kod.
        // Do wypisywania, użyj poniższej linijki
        // for (int i = 0; i < tablica.length; i++) { System.out.print(tablica[i] + " "); }
    }
}
```

```
1
```

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie przez scalanie 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.

III. Posługiwanie się komputerem, urządzeniami cyfrowymi i sieciami komputerowymi, w tym: znajomość zasad działania urządzeń cyfrowych i sieci komputerowych oraz wykonywania obliczeń i programów.

Treści nauczania – wymagania szczegółowe

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

1) zapisuje za pomocą listy kroków, schematu blokowego lub pseudokodu, i implementuje w wybranym języku programowania, algorytmy poznane na wcześniejszych etapach oraz algorytmy:

e) sortowania ciągu liczb przez scalanie,

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):

- Wyjaśnisz, czym jest rekurencja oraz metoda „dziel i zwyciężaj".
- Przeanalizujesz, jak działa algorytm sortowania przez scalanie.
- Napiszesz program wykorzystujący algorytm sortowania przez scalanie oraz rozwiążesz kilka zadań z zastosowaniem tego algorytmu.

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;
- oprogramowanie dla języka Java SE 8 (lub nowszej wersji), w tym Eclipse 4.4 (lub nowszej wersji).

Przebieg lekcji

Przed lekcją:

1. Chętne lub wybrane osoby przedstawiają najważniejsze informacje dotyczące algorytmu sortowania przez scalanie.

Faza wstępna:

1. Uczniowie zapoznają się z sekcją „Film samouczek”.

Faza realizacyjna:

1. Uczniowie w parach szukają rozwiązania dla postawionego w sekcji „Film samouczek” problemu z polecenia 1. Chętne lub wybrane osoby prezentują swój kod. Nauczyciel go omawia.
2. Uczniowie podają swoje propozycje ulepszenia proponowanych rozwiązań.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1 z sekcji „Sprawdź się”, a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.
4. W kolejnym etapie uczniowie dobierają się w pary i wykonują ćwiczenia nr 2 i 3. Następnie konsultują swoje rozwiązania z inną parą uczniów i ustalają jedną wersję odpowiedzi.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie”. W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.
2. Wybrany uczeń podsumowuje zajęcia z programowania w Javie, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Dodaj do napisanych programów odpowiedni komentarz. Porównaj swoje rozwiązanie z filmem.

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 na lekcji jako podsumowanie i utrwalenie wiedzy uczniów.