Sortowanie przez scalanie – zadania maturalne

Bibliografia:

- Źródło: oprac. własne.

Źródło: Xavi Cabrera, domena publiczna.

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

W tym e-materiale zapoznamy się z przykładowym zadaniem maturalnym, dotyczącym tego zagadnienia.

Implementację algorytmu sortowania przez scalanie w poszczególnych językach programowania przedstawiamy w e-materiałach:

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

Twoje cele

- Przeanalizujesz implementację algorytmu sortowania przez scalanie w pseudokodzie.
- Rozwiążesz zadania typu maturalnego dotyczące sortowania przez scalanie.
- Scharakteryzujesz algorytm sortowania przez scalanie i prześledzisz jego działanie.

Przeczytaj

Algorytm sortowania przez scalanie

Sortowanie przez scalanie polega na wielokrotnym dzieleniu na połowy tablicy zawierającej dane przeznaczone do uporządkowania.

Kiedy wszystkie tablice będą zawierać jeden element, następuje ich łączenie.

Dane pochodzące z dwóch tablic są zapisywane w ustalonym porządku (niemalejąco lub nierosnąco).

Proces łączenia tablic trwa do momentu, w którym pozostanie tylko jedna tablica.

Przykładowe zadanie typu maturalnego

Zadanie 1. Hodowla owiec

W miejscowości Rekursja działa farma zajmująca się hodowlą owiec. W gospodarstwie znajduje się 10 000 zwierząt. Każda owca jest oznaczona unikalnym numerem z zakresu (0, 100 000).

Zadanie 1.1

Co roku odbywa się strzyżenie owiec. Przeprowadzane jest przez dziesięć zespołów. Owce są prowadzone z pastwiska na miejsce strzyżenia w kolejności losowej: pierwsza przyprowadzona owca trafia do pierwszego zespołu, druga do drugiego; jedenasta owca znów do pierwszego zespołu i tak dalej. W ramach poszczególnych zespołów owce są strzyżone w kolejności zgodnej z ich oznaczeniami, zaczynając od najmniejszego numeru.

Napisz program, który przydzieli owce do poszczególnych zespołów oraz określi kolejność strzyżenia owiec w ramach zespołów.

W pliku `owce.txt` zapisano 10 000 liczb całkowitych z przedziału (0, 100 000), każdą w nowej linii – są to identyfikatory owiec; kolejność w pliku odzwierciedla kolejność, w której owce są sprowadzane z pastwiska na miejsce strzyżenia.

Plik o rozmiarze 67.28 KB w języku polskim

> Przykładowe dane:
>
> ```
> 50270
> ```

```
51360
59263
```

Do oceny oddajesz:

- plik `wyniki.txt` zawierający odpowiedź (identyfikatory owiec podzielone na 10 grup zgodnie z treścią zadania, posortowane niemalejąco w obrębie grup; każda grupa powinna być oddzielona pojedynczą pustą linią)
- plik(i) z komputerową realizacją zadania (kodem programu)

Rozwiązanie przedstawimy w postaci pseudokodu, ponieważ na egzaminie maturalnym można korzystać z samodzielnie wybranego języka programowania: C++, Java lub Python.

> **Praca domowa**
>
> Przedstaw rozwiązanie zadania w postaci programu w języku C++, Java lub Python. Zadbaj o prawidłowe wczytanie danych z pliku tekstowego do programu. Odpowiedź do zadania znajdziesz w osobnym pliku umieszczonym po omówieniu pseudokodu.

Rozwiązanie

Na początek zauważmy, że opisany sposób podziału owiec pomiędzy zespoły zajmujące się strzyżeniem to nic innego jak przyporządkowanie na podstawie reszty z dzielenia przez 10. Nadajmy zespołom numery od 0 do 9. Uznajmy również, że będziemy numerować linie w pliku wejściowym od 0. Dzięki temu możemy powiedzieć, że wszystkie owce o identyfikatorach znajdujących się w liniach, których reszta z dzielenia przez 10 wynosi 0, powinny być strzyżone przez zespół o numerze 0. Wszystkie owce o identyfikatorach znajdujących się w liniach, których reszta z dzielenia przez 10 wynosi 1, powinny być strzyżone przez zespół o numerze 1 itd.

Zaczniemy pseudokod od wczytania pliku z danymi do dwuwymiarowej tablicy `zespolyOwce`. Tablica miała tyle tyle wierszy, ile jest zespołów (10) oraz tyle kolumn, ile wynosi liczba owiec *n* przez liczbę zespołów (czyli 1000). Zakładamy, że operator `%` to operator modulo (reszty z dzielenia), a operator `//` to dzielenie całkowitoliczbowe.

```
n = 10000
liczbaZespolow = 10
zespolyOwce[liczbaZespolow][n//liczbaZespolow]

dla i = 0, 1, ..., n - 1 wykonuj:
    zespol = i % liczbaZespolow
    miejsceWTablicy = i // liczbaZespolow
    zespolyOwce[zespol][miejsceWTablicy] = i-ty wiersz pliku "ow
```

Będziemy sortować tak przygotowane dane. Każdy wiersz tablicy `zespolyOwce` będziemy sortować oddzielnie z użyciem algorytmu sortowania przez scalanie.

```
1 dla i = 0, 1, ..., liczbaZespolow - 1 wykonuj:
2     sortowanieScalanie(zespolyOwce[i], 1, n//liczbaZespolow)
```

Wewnątrz pętli wywołujemy funkcję realizującą algorytm sortowania przez scalanie. Jako dane wejściowe podajemy: tablicę z numerami owiec – czyli wiersz dwuwymiarowej tablicy, indeks komórki, od której rozpoczniemy sortowanie oraz indeks komórki, na której zakończymy sortowanie.

Gdy już wszystkie wiersze zostaną posortowane, pozostaje zapisać dane do pliku wyjściowego zgodnie z treścią zadania: kolejnymi grupami, a między każdą grupą powinien znaleźć się jeden pusty wiersz.

```
1 dla i = 0, 1, ..., liczbaZespolow - 1 wykonuj:
2     dla j = 0, 1, ..., n//liczbaZespolow - 1 wykonuj:
3         wypisz zespolyOwce[i][j] do pliku "wyniki.txt"
4     wypisz pustą linię do pliku "wyniki.txt"
```

Przeanalizujmy algorytm sortujący.

```
1 funkcja sortowanieScalanie(owce[], lewy, prawy)
2     jeżeli lewy >= prawy wykonaj:
3         zakończ
4     środek = (lewy + prawy) // 2
5     sortowanieScalanie(owce, lewy, środek)
6     sortowanieScalanie(owce, środek + 1, prawy)
7     scal(owce, lewy, środek, prawy)
```

1. Definujemy funkcję, która będzie dzielić tablicę na części. **[linia 1]**
2. Sprawdzamy, czy tablica, która ma zostać podzielona, nie jest tablicą jednoelementową (jeżeli tak jest, kończymy bieżące wywołanie rekurencyjne funkcji). **[linie 2, 3]**
3. Wyznaczamy środek podziału tablicy. **[linia 4]**
4. Wywołujemy funkcję sortowania przez scalanie dla utworzonych podtablic. **[linie 5, 6]**
5. Scalamy jednoelementowe tablice. **[linia 7]**

```
1 funkcja scal(owce[], lewy, środek, prawy)
```

```
2     długośćL = środek - lewy
3     długośćP = prawy - środek
4     dla i = 1, 2, ... długośćL wykonuj:
5         lewaPom[i] = owce[lewy + i]
6     dla i = 1, 2, ... długośćP wykonuj:
7         prawaPom[i] = owce[środek + i + 1]
8     x = 1
9     y = 1
10    z = lewy
11    dopóki (x < długośćL && y < długośćP) wykonuj:
12        jeżeli lewaPom[x] <= prawaPom[y] wykonaj:
13            owce[z] = lewaPom[x]
14            x = x + 1
15        jeżeli lewaPom[x] > prawaPom[y] wykonaj:
16            owce[z] = prawaPom[y]
17            y = y + 1
18        z = z + 1
19    dopóki x < lewy wykonuj:
20        owce[z] = lewaPom[x]
21        x = x + 1
22        z = z + 1
23    dopóki y < prawy wykonuj:
24        owce[z] = prawaPom[y]
25        y = y + 1
26        z = z + 1
27    zakończ
```

1. Definiujemy funkcję scalania, która będzie zapisywać elementy pochodzące z dwóch podtablic w jednej tablicy z zachowaniem odpowiedniej kolejności. **[linia 1]**
2. Ustalamy długości podtablic, a następnie zapisujemy je w zmiennych pomocniczych. Musimy pamiętać, że podział jest umowny – nie tworzyliśmy nowych tablic danych, a zatem cały czas przetwarzamy tablicę `Owce`. Aby nie utracić danych z oryginalnej tablicy, musimy umieścić sortowaną część w tablicach pomocniczych. **[linie od 2 do 7]**
3. Tworzymy zmienne pomocnicze i ustawiamy ich wartość na 1 (w przedstawionym kodzie indeksy komórek tablicy zaczynają się od 1). **[linie od 8 do 10]**
4. Dopóki podtablice zawierają elementy, które nie znajdują się w scalonej tablicy, wykonywane są następne kroki. **[linia 11]**
5. Porównujemy kolejne wartości z podtablic i ustawiamy je w kolejności od najmniejszej do największej w pierwotnej tablicy. Wartość `x` inkrementujemy po wpisaniu danych z tablicy `lewaPom` do pierwotnej tablicy. Tak samo postępujemy w przypadku wartości `y` (dla tablicy `prawaPom`). **[linie od 11 do 17]**

6. Inkrementujemy wartość zmiennej `z`, która przechowuje indeks komórki pierwotnej tablicy, do której powinniśmy wpisać dane. **[linia kodu 18]**
7. Jeżeli zdarzy się, że wyczerpiemy dane z jednej podtablicy, wpisujemy resztę danych z drugiej podtablicy do końca jej długości, a następnie kończymy działanie funkcji. **[linie od 19 do 27]**

> Odpowiedź dla danych zapisanych w pliku tekstowym:
>
> Plik o rozmiarze 57.52 KB w języku polskim

Ćwiczenie 1

Zapisz program, wykorzystując wybrany język programowania.

```
1
```

```
1
```

Zadanie 1.2

Starsze owce przechodzą rutynowe badania kontrolne co 2 lata. Każdego roku jedna połowa owiec jest kierowana na badania. Druga połowa jest badana w kolejnym roku. W latach nieparzystych badania przechodzą tylko owce mające nieparzystą liczbę lat. Podobnie jest w przypadku lat parzystych i owiec mających parzystą liczbę lat. Owce badane są w kolejności od najstarszej do najmłodszej.

W 2012 roku przebadanych zostanie 2000 owiec.

Napisz program, który wyznaczy, w jakiej kolejności zwierzęta zostaną przebadane.

W pliku `owce do badania.txt` zapisano dane na temat wieku 2000 owiec. W każdej linii znajduje się liczba naturalna – wiek owcy w miesiącach.

Plik o rozmiarze 9.67 KB w języku polskim

Do oceny oddajesz:

- plik `weterynarz.txt` zawierający odpowiedź (oznaczenia owiec, zakwalifikowanych do badań w 2012 r.; kolejne wyniki zapisane w pliku powinny zgadzać się z kolejnością opisaną w zadaniu)
- plik(i) z komputerową realizacją zadania (kodem programu)

Rozwiązanie przedstawimy w postaci pseudokodu, ponieważ na egzaminie maturalnym można korzystać z samodzielnie wybranego języka programowania: C++, Java lub Python.

> **Praca domowa**
>
> Przedstaw rozwiązanie zadania w postaci programu w języku C++, Java lub Python. Zadbaj o prawidłowe wczytanie danych z pliku tekstowego do programu. Odpowiedź do zadania znajdziesz w osobnym pliku umieszczonym za omówieniem pseudokodu.
> **Ważne!**
>
> Przy sprawdzaniu parzystości wieku, uwzględnij tylko skończone lata.

Rozwiązanie

> Podczas rozwiązywania zadania skorzystamy ze zmodyfikowanej wersji kodu z zadania 1.1.
>
> W tym przypadku nie sortujemy wszystkich danych. Musimy zatem zapisać interesujące nas informacje do osobnej tabeli, a następnie odpowiednio posortować jej zawartość.

```
owce[] = "owce do badania.txt"
długośćTablicy = 2000
owceDoBadania = doBadania(owce, długośćTablicy)
długośćTablicyOwceDoBadania = długość tablicy owceDoBadania
sortowanieScalanie(owceDoBadania, 1, długośćTablicyOwceDoBadania)
```

```
6 dla i = 0, 1, ... długośćTablicy - 1 wykonuj:
7     wypisz owceDoBadania[i]
```

- Wywołujemy funkcję `doBadania()`, która zapisze interesujące nas dane w nowej tablicy `owceDoBadania`. **[linia 3]**
- Pozostałe operacje zapisane w głównej części kodu są takie same jak te z zadania 1.1.

```
1 funkcja doBadania(owce[], długośćTablicy)
2     x = 1
3     dla i = 0, 1, ... długośćTablicy - 1 wykonuj:
4         jeżeli ((owce[i] - (owce[i] % 12)) // 12) % 2 == 0 wykona
5             owceDoBadania[x] = owce[i]
6             x = x + 1
7     zwróć owceDoBadania
```

1. Definiujemy funkcję, która zapisze dane spełniające warunki zadania w tablicy `owceDoBadania`. **[linia 1]**
2. Zapisujemy indeks komórki tablicy `owceDoBadania`, w której będą przechowywane kolejne dane **[linia 2]**
3. Odczytujemy kolejne komórki tablicy `owce` i wpisujemy do tablicy `owceDoBadania` te spośród nich, które odpowiadają parzystemu wiekowi zwierząt (liczonemu w latach). Aby ustalić wiek owcy z uwzględnieniem tylko skończonych lat (ignorując pozostałe miesiące), zaczynamy od wyznaczenia reszty z dzielenia wieku owcy podanego w miesiącach przez 12 (liczbę miesięcy, które nie składają się na pełne lata). Wynik odejmujemy od wieku liczonego w miesiącach i dzielimy przez 12, aby wyznaczyć wiek w latach. Następnie sprawdzamy, czy jest on parzysty. **[linia 4]**
4. Jeżeli wiek jest parzysty, wpisujemy go do tablicy `owceDoBadania` i inkrementujemy wartość `x` (wskazujemy indeks następnej komórki). **[linie 5, 6]**

```
1 sortowanieScalanie(owceDoBadania[], lewy, prawy)
2     jeżeli lewy >= prawy wykonaj:
3         zakończ
4     środek = (lewy + prawy) // 2
5     sortowanieScalanie(owceDoBadania, lewy, środek)
6     sortowanieScalanie(owceDoBadania, środek + 1, prawy)
7     scal(owceDoBadania, lewy, środek, prawy)
```

- Definicja funkcji `sortowanieScalanie()` jest taka sama jak zadaniu 1.1.

```
1  scal(owceDoBadania[], lewy, środek, prawy)
2      długośćL = środek - lewy
3      długośćP = prawy - środek
4      dla i = 1, 2, ... długośćL wykonuj:
5          lewaPom[i] = owceDoBadania[lewy + i]
6      dla i = 1, 2, ... długośćP wykonuj:
7          prawaPom[i] = owceDoBadania[środek + i + 1]
8      x = 1
9      y = 1
10     z = 1
11     dopóki (x < długośćL && y < długośćP) wykonuj:
12         jeżeli lewaPom[x] >= prawaPom[y] wykonaj:
13             owceDoBadania[z] = lewaPom[x]
14             x = x + 1
15         jeżeli lewaPom[x] < prawaPom[y] wykonaj:
16             owceDoBadania[z] = prawaPom[y]
17             y = y + 1
18         z = z + 1
19     dopóki x < lewy wykonuj:
20         owceDoBadania[z] = lewaPom[x]
21         x = x + 1
22         z = z + 1
23     dopóki y < prawy wykonuj:
24         owceDoBadania[z] = prawaPom[y]
25         y = y + 1
26         z = z + 1
27     zakończ
```

- Modyfikujemy funkcję `scal()` tak, aby zapisywała ona wartości od największej do najmniejszej. Zamieniamy w tym celu znaki mniejszości (<) na znaki większości (>) i odwrotnie. **[linie 12, 15]**

> Odpowiedź dla danych zawartych w pliku tekstowym:
>
> Plik o rozmiarze 2.82 KB w języku polskim

Ćwiczenie 2

Zapisz program, wykorzystując wybrany język programowania.

```
1
```

```
1
```

Słownik

indeks komórki

oznaczenie (numer) pozycji elementu w tablicy
inkrementacja

zwiększenie wartości o 1
rekurencja

metoda programowania w dowolnym języku, polegająca na wywołaniu przez funkcję samej siebie; rekurencję można porównać do pętli, ponieważ jest ona wywoływana określoną liczbę razy (do momentu, gdy zajdzie warunek stopu)

Prezentacja multimedialna

Zadanie 2. Pracownia chemiczna

W pracowni chemicznej przechowuje się wiele odczynników o różnych wartościach pH.

Przed wykonaniem pewnego eksperymentu postanowiono posortować odczynniki o pH zasadowym niemalejąco.

Substancje należy posortować tak, aby sortowanie odbywało się tylko na pozycjach, na których znajdują się odczynniki o odpowiednim pH.

Przyjmij, że wartość pH zasadowego należy do przedziału [8, 14].

Napisz program, który wypisze posortowane niemalejąco odczynniki o zasadowym pH.

W pliku `odczynniki.txt` zapisano 100 wierszy z wartościami pH różnych substancji. Wartość pH każdego odczynnika jest liczbą z przedziału [1, 14].

Plik o rozmiarze 618.00 B w języku polskim

> Przykładowe dane:
>
> ```
> 10,74
> 10,8
> 10,9
> ```

Do oceny oddajesz:

- plik `eksperyment.txt` zawierający odpowiedź (odczynniki o pH zasadowym posortowane niemalejąco)
- plik(i) z komputerową realizacją zadania (kodem programu)

> **Ważne!**
>
> Pamiętaj, że substancje o pH innym niż zasadowe muszą pozostać na pierwotnych pozycjach. Natomiast odczynniki o pH zasadowym, po posortowaniu, powinny znajdować się na pozycjach, na których pierwotnie znajdowały się inne (lub te same) odczynniki o pH zasadowym.
> **Polecenie 1**
>
> Przedstaw rozwiązanie zadania w postaci programu w języku C++, Java lub Python. Zadbaj o prawidłowe wczytanie danych z pliku tekstowego do programu. Odpowiedź do zadania znajdziesz w osobnym pliku umieszczonym pod omówieniem pseudokodu.

Rozwiązanie

> **Polecenie 2**
>
> Zapoznaj się z rozwiązaniem przedstawionym w postaci pseudokodu. Jest ono zapisane w tej formie, ponieważ na egzaminie maturalnym można korzystać z samodzielnie wybranego języka programowania: C++, Java lub Python.

W celu rozwiązania zadania musimy oddzielić najpierw substancje spełniające podane kryteria (odpowiednia wartość pH), od tych które nie powinny być sortowane.

Następnie posortujemy substancje o pH zasadowym i zapiszemy je we właściwej kolejności na pozycjach, na których pierwotnie znajdowały się odczynniki o pH zasadowym.

Na koniec zapiszemy wyniki w pliku `eksperyment.txt`.

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 1

Rozpocznijmy od zapisania w tablicy danych z pliku `odczynniki.txt` oraz zdefiniowania rozmiaru zbioru danych w głównej części kodu.

```
odczynniki[] =
"odczynniki.txt"
długośćListy = 100
```

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 2

Następnie musimy oddzielić dane do sortowania od tych, których sortować nie powinniśmy.

W tym celu wywołujemy funkcję, która zapisze dane do sortowania w tablicy pomocniczej (definicja funkcji zostanie przedstawiona w kroku 6).

```
1 odczynniki[] =
  "odczynniki.txt"
2 długośćListy = 100
3 doSortowania(odczynniki,
  długośćListy
```

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 3

Tablicę pomocniczą należy posortować. Wywołujemy więc funkcję sortowania przez scalanie (definicja funkcji pojawi się w kroku 10).

```
1 odczynniki[] =
  "odczynniki.txt"
2 długośćListy = 100
3 pomocnicza =
  doSortowania(odczynniki,
  długośćListy)
4 długośćPom = długość
  tabeli pomocnicza
5 sortowanieScalanie(pomocni
  cza, 1, długośćPom)
```

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 4

Po posortowaniu substancji o pH zasadowym należy wpisać je do komórek, w których przed sortowaniem również znajdowały się odczynniki o pH zasadowym. Skorzystamy z pętli, zapisując posortowane dane na odpowiednich pozycjach (zajmowanych przed sortowaniem przez substancje o pH zasadowym).

```
odczynniki[] = "odczynniki.txt"
długośćListy = 100
pomocnicza = doSortowania(odczynniki, długośćListy)
długośćPom = długość tabeli pomocnicza
sortowanieScalanie(pomocnicza, 1, długośćPom)
dla i = 0, 1, ... długośćPom - 1 wykonuj:
    odczynniki[pozycje[i]] = pomocnicza[i]
```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

5

Krok 5

Na koniec zapisujemy zmodyfikowaną tablicę w pliku `eksperyment.txt`.

```
odczynniki[] = "odczynniki.txt"
długośćListy = 100
pomocnicza, pozycje = doSortowania(odczynniki, długośćListy)
długośćPom = długość tabeli pomocnicza
```

```
sortowanieScalanie(pomocnicza, 1, długośćPom)
dla i = 0, 1, ... długośćPom - 1 wykonuj:
    odczynniki[pozycje[i]] = pomocnicza[i]
"eksperyment.txt" = odczynniki
```

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 6

Zdefiniujmy funkcję, która oddzieli dane do sortowania od reszty oraz zapisze indeksy komórek, w których się one znajdowały.

```
funkcja doSortowania(odczynniki[], długośćListy)
```

7

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 7

Zapisujemy w zmiennej pomocniczej x wartość pierwszego indeksu tablicy, a następnie sprawdzamy w pętli, czy pH substancji jest zasadowe (czyli większe lub równe 8).

```
funkcja doSortowania(odczynniki[], długośćListy)
    x = 1
```

```
3     dla i = 0, 1, ...
długośćListy - 1 wykonuj:
4         jeżeli
odczynniki[i] >= 8
```

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 8

Jeżeli pH odczynnika jest zasadowe, to zapisujemy tę wartość w tablicy do posortowania. Zapamiętujemy też odpowiadający odczynnikowi indeks (z tablicy `odczynniki`) w tablicy `pozycje`. Będziemy dzięki temu wiedzieć, gdzie umieścić posortowane dane. Inkrementujemy wartość zmiennej pomocniczej x; przechowa ona indeks następnej komórki tablicy `pomocnicza`.

```
1 funkcja
doSortowania(odczynniki[],
długośćListy)
2     x = 1
3     dla i = 0, 1, ...
długośćListy - 1 wykonuj:
4         jeżeli
odczynniki[i] >= 8
wykonaj:
5             pomocnicza[x]
= odczynniki[i]
6             pozycje[x] = i
7             x = x + 1
```

Materiał audio dostępny pod adresem:

9

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 9

Po umieszczeniu danych w tablicy `pomocnicza` zapisujemy jej długość (pamiętaj, że zmienna `x` zawiera indeks następnej komórki, więc musimy odjąć od jej wartości 1).

```
1  funkcja
   doSortowania(odczynniki[],
   długośćListy)
2      x = 1
3      dla i = 0, 1, ...
   długośćListy - 1 wykonuj:
4          jeżeli
   odczynniki[i] >= 8
   wykonaj:
5              pomocnicza[x]
   = odczynniki[i]
6              pozycje[x] = i
7              x = x + 1
8      długośćPom = x - 1
9      zwróć pomocnicza,
   pozycje
10
```

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 10

Definiujemy funkcję sortowania przez scalanie.

```
1  funkcja
   sortowanieScalanie(pomocni
   cza[], lewy, prawy)
```

11

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 11

Sprawdzamy, czy tablica jest jednoelementowa. Jeżeli tak jest, kończymy działanie bieżącego wywołania funkcji (warunek konieczny do zakończenia rekurencyjnego podziału tablicy). Jeżeli tablica nie jest jednoelementowa, wyznaczamy jej środek.

```
1 funkcja sortowanieScalanie(pomocnicza[], lewy, prawy)
2     jeżeli lewy >= prawy wykonaj:
3         zakończ
4     środek = (lewy + prawy) / 2
```

12

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 12

Dokonujemy podziału tablicy na dane znajdujące się po lewej oraz prawej stronie środka tablicy. Robimy to przez rekurencyjne wywołanie funkcji dla lewej i prawej części tablicy.

```
1 funkcja sortowanieScalanie(pomocnicza[], lewy, prawy)
2     jeżeli lewy >= prawy wykonaj:
3         zakończ
```

```
    środek = (lewy + prawy) / 2

sortowanieScalanie(pomocnicza, lewy, środek)

sortowanieScalanie(pomocnicza, środek + 1, prawy)
```

13

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 13

Po zakończeniu podziału scalamy powstałe tablice funkcją `scal()`, która ustawia odpowiednio dane z podzielonych tablic (definicja funkcji znajduje się w kroku 14).

```
funkcja sortowanieScalanie(pomocnicza[], lewy, prawy)
    jeżeli lewy >= prawy wykonaj:
        zakończ
    środek = (lewy + prawy) / 2

sortowanieScalanie(pomocnicza, lewy, środek)

sortowanieScalanie(pomocnicza, środek + 1, prawy)
    scal(pomocnicza, lewy, środek, prawy)
```

14

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 14

Zdefiniujmy funkcję, która scali powstałe podtablice, ustawiając dane w odpowiedniej kolejności.

```
funkcja scal(pomocnicza, lewy, środek, prawy)
```

15

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 15

Zapisujemy długości podtablic w zmiennych pomocniczych.

```
funkcja scal(pomocnicza[], lewy, środek, prawy)
    długośćL = środek - lewy
    długośćP = prawy - środek
```

16

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 16

Umieszczamy scalaną (w tym wywołaniu rekurencyjnym funkcji `sortowanieScalanie()`) część danych w tablicach pomocniczych.

```
funkcja scal(pomocnicza[], lewy, środek, prawy)
    długośćL = środek - lewy
    długośćP = prawy - środek
    dla i = 0, 1, ... długośćL - 1 wykonuj:
        lewaPom[i] = pomocnicza[lewy + i]
    dla i = 0, 1, ... długośćP - 1 wykonuj:
        prawaPom[i] = pomocnicza[środek + i + 1]
```

17

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 17

Ustalamy wartości zmiennych pomoczniczych – w podanym kodzie indeksy komórek zaczynają się od 1.

```
funkcja scal(pomocnicza[], lewy, środek, prawy)
    długośćL = środek - lewy
    długośćP = prawy - środek
    dla i = 0, 1, ... długośćL - 1 wykonuj:
        lewaPom[i] = pomocnicza[lewy + i]
    dla i = 0, 1, ... długośćP - 1 wykonuj:
        prawaPom[i] = pomocnicza[środek + i + 1]
    x = 1
    y = 1
```

```
10      z = 1
```

18

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 18

Ustawiamy dane w tablicy pomocniczej od najmniejszej do największej. W tym celu porównujemy wartości na kolejnych pozycjach w podtablicach, a następnie mniejszy element zapisujemy w kolejnej komórce tablicy `pomocnicza`.

```
1  funkcja scal(pomocnicza[], lewy, środek, prawy)
2      długośćL = środek - lewy
3      długośćP = prawy - środek
4      dla i = 0, 1, ... długośćL - 1 wykonuj:
5          lewaPom[i] = pomocnicza[lewy + i]
6      dla i = 0, 1, ... długośćP - 1 wykonuj:
7          prawaPom[i] = pomocnicza[środek + i + 1]
8      x = 1
9      y = 1
10     z = 1
11     dopóki (x < długośćL && y < długośćP) wykonuj:
12         jeżeli lewaPom[x] <= prawaPom[y] wykonaj:
13             pomocnicza[z] = lewaPom[x]
14             x = x + 1
15         jeżeli lewaPom[x] > prawaPom[y] wykonaj:
```

```
            pomocnicza[z]
= prawaPom[y]
            y = y + 1
        z = z + 1
```

19

Więcej zadań znajdziesz w zbiorze dostępnym na stronie internetowej Okręgowej Komisji Egzaminacyjnej w Warszawie.
Źródło: OKE, oke.waw.pl, tylko do użytku edukacyjnego.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PCp5IZJ5y

Krok 19

Gdy zapiszemy wszystkie dane z jednej podtablicy, zapisujemy po kolei resztę danych na końcu tablicy `pomocnicza`. Po tej czynności kończymy działanie funkcji scalającej.

```
funkcja scal(pomocnicza[], lewy, środek, prawy)
    długośćL = środek - lewy
    długośćP = prawy - środek
    dla i = 0, 1, ... długośćL - 1 wykonuj:
        lewaPom[i] = pomocnicza[lewy + i]
    dla i = 0, 1, ... długośćP - 1 wykonuj:
        prawaPom[i] = pomocnicza[środek + i + 1]
    x = 1
    y = 1
    z = 1
```

```
11      dopóki (x < długośćL
&& y < długośćP) wykonuj:
12          jeżeli lewaPom[x]
<= prawaPom[y] wykonaj:
13              pomocnicza[z]
= lewaPom[x]
14              x = x + 1
15          jeżeli lewaPom[x]
> prawaPom[y] wykonaj:
16              pomocnicza[z]
= prawaPom[y]
17              y = y + 1
18          z = z + 1
19      dopóki x < lewy
wykonuj:
20          pomocnicza[z] =
lewaPom[x]
21          x = x + 1
22          z = z + 1
23      dopóki y < prawy
wykonuj:
24          pomocnicza[z] =
prawaPom[y]
25          y = y + 1
26          z = z + 1
27      zakończ
```


> Odpowiedzi dla danych z pliku tekstowego:
>
> `eksperyment.txt`
>
> Plik o rozmiarze 281.00 B w języku polskim

Ćwiczenie 1

Zapisz program, wykorzystując wybrany język programowania.

```
1
```

```
1
```

Sprawdź się

Zadanie 3. Kolekcja medali

Pan Operacyjny wywiesił na ścianie swoją kolekcję medali. Medale ustawione są w kolejności od najmniejszego do największego, w zależności od ich pola powierzchni. Występują one w dwóch kształtach – prostokąta oraz koła.

W związku z remontem, który miał miejsce w mieszkaniu, Pan Operacyjny musiał zdjąć wszystkie medale.

Po zakończeniu prac postanowił ułożyć je na miejscach, na których wcześniej się znajdowały.

W tym celu sporządził trzy listy – wysokość medalu, szerokość medalu oraz kształt. Dane zostały zapisane kolejno w dwóch plikach: `wysokość.txt`, `szerokość.txt`.

Dane z tych samych wierszy odnoszą się do jednego medalu, np. pierwszy wiersz w pliku `wysokość.txt` opisuje wysokość medalu, który opisuje pierwszy wiersz w pliku `szerokość.txt`.

W sytuacji, gdy wysokość równa jest szerokości, medal ma kształt koła.

Napisz program, który posortuje medale w kolejności od najmniejszego do największego, w zależności od ich pola powierzchni i wypisze w tej kolejności ich wysokości i szerokości.

> **Ważne!**
>
> Do rozwiązania zadania przyjmij, że $\pi = 3{,}14$

W plikach `wysokość.txt` oraz `szerokość.txt` znajduje się 80 wierszy z liczbami rzeczywistymi z przedziału $[1, 11)$ opisującymi kolejno wysokość oraz szerokość medali.

Plik `wysokość.txt`
Plik o rozmiarze 487.00 B w języku polskim

Plik `szerokość.txt`
Plik o rozmiarze 487.00 B w języku polskim

> Przykładowe dane:

```
3,90
4,95
5,11
9,66
```

Do oceny oddajesz:

- pliki `wysokość2.txt` oraz `szerokość2.txt` zawierające odpowiedź (dane w kolejności takiej, że pierwszy wiersz w pliku `wysokość.txt` opisuje wysokość medalu, który opisuje pierwszy wiersz w pliku `szerokość.txt`)
- plik(i) z komputerową realizacją zadania (kodem programu)

W przedstawionych ćwiczeniach część danych została umieszczona w tablicach. Wykorzystaj je do rozwiązania zadania.

> **Praca domowa**
>
> Napisz implementację zadania lokalnie na swoim komputerze w języku, w którym programujesz. Zadbaj o prawidłowe wczytanie danych z pliku tekstowego do swojego programu. Odpowiedź do zadania dla danych z pliku znajdziesz na samym końcu zadania.

Pokaż ćwiczenia: 

> C++

Ćwiczenie 1

Używając języka C++, napisz program, który – w zaprezentowany sposób – posortuje dane opisujące medale.

Twoje zadania

1. Program powinien dla sprawdzenia wypisać pole powierzchni trzeciego, szóstego oraz dziewiątego z kolei medalu z pojedynczymi znakami odstępu pomiędzy liczbami.

```cpp
#include <iostream>

using namespace std;

int main(){

    float wysokosci[] = {10.24, 5.63, 9.12, 9.39, 9.43, 10.54, 10.81, 7.00, 9.58, 1.36, 3.09};
    float szerokosci[] = {3.44, 2.99, 10.71, 2.35, 7.55, 5.45, 8.01, 8.90, 10.25, 1.00, 3.09};
    int dlugosc = 11;
}
```

```cpp

```

Java

Ćwiczenie 2

Używając języka Java, napisz program, który – w zaprezentowany sposób – posortuje dane opisujące medale.

Twoje zadania

1. Program powinien dla sprawdzenia wypisać pole powierzchni trzeciego, szóstego oraz dziewiątego z kolei medalu z pojedynczymi znakami odstępu pomiędzy liczbami.

```
1  public class Main  {
2      // Tutaj wpisz kod
3
4      public static void main(String[] args) {
5          float[] wysokosci = {10.24f, 5.63f, 9.12f, 9.39f,
   9.43f, 10.54f, 10.81f, 7.00f, 9.58f, 1.36f, 3.09f};
6          float[] szerokosci = {3.44f, 2.99f, 10.71f, 2.35f,
   7.55f, 5.45f, 8.01f, 8.90f, 10.25f, 1.00f, 3.09f};
7
8          // Tutaj wpisz kod
9      }
10 }
```

```
1
```

Python

Ćwiczenie 3

Używając języka Python, napisz program, który – w zaprezentowany sposób – posortuje dane opisujące medale.

Twoje zadania

1. Program powinien dla sprawdzenia wypisać pole powierzchni trzeciego, szóstego oraz dziewiątego z kolei medalu z pojedynczymi znakami odstępu pomiędzy liczbami.

```
1 wysokosci = [10.24, 5.63, 9.12, 9.39, 9.43, 10.54, 10.81,
  7.00, 9.58, 1.36, 3.09]
2 szerokosci = [3.44, 2.99, 10.71, 2.35, 7.55, 5.45, 8.01, 8.90,
  10.25, 1.00, 3.09]
```

```
1
```

Praca domowa

Odpowiedź dla danych zawartych w pliku tekstowym:

Wysokości posortowanych medali:

Plik o rozmiarze 475.00 B w języku polskim

Szerokości posortowanych medali:

Plik o rozmiarze 476.00 B w języku polskim

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie przez scalanie – zadania maturalne

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

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

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

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

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,

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 implementację algorytmu sortowania przez scalanie w pseudokodzie.
- Rozwiążesz zadania typu maturalnego dotyczące sortowania przez scalanie.
- Scharakteryzujesz algorytm sortowania przez scalanie i prześledzisz jego działanie.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- metody aktywizujące (burza mózgów);
- 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 C++, w tym kompilator GCC/G++ 4.5 (lub nowszej wersji) i Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio;
- oprogramowanie dla języka Java SE 8 (lub nowszej wersji), w tym Eclipse 4.4 (lub nowszej wersji);
- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Sortowanie przez scalanie – zadania maturalne". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

Faza wstępna:

1. Wyświetlenie przez nauczyciela tematów i celów zajęć, przejście do wspólnego ustalenia kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytania dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu. Uczniowie metodą burzy mózgów przypominają sobie najważniejsze informacje dot. sortowania przez scalanie.

Faza realizacyjna:

1. **Praca z tekstem.** Uczniowie w parach analizują rozwiązanie zadania z sekcji „Przeczytaj". Następnie implementują je w wybranym języku programowania.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Prezentacja multimedialna". Uczniowie wspólnie zapoznają się z przedstawionym w niej rozwiązaniem zadania nr 2 w postaci pseudokodu.
3. **Ćwiczenia umiejętności.** Uczniowie wykonują ćwiczenia z sekcji „Sprawdź się". Ich zadaniem jest napisanie programu, który posortuje dane opisujące medale w podany w zadaniu sposób.
   Ćwiczenie realizują w jednym z dostępnych języków programowania.

Faza podsumowująca:

1. Nauczyciel zadaje pytania podsumowujące, np.
   – czym jest rekurencja?
   – co oznacza indeks komórki?
   –co to jest inkrementacja?
   – na czym polega sortowanie przez scalanie?
2. Nauczyciel prosi uczniów o podsumowanie zgromadzonej wiedzy w zakresie programowania.

Praca domowa:

1. Uczniowie wykonują zadanie z sekcji „Prezentacja multimedialna" w postaci programu w języku C++, Java lub Python.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka C++.
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

- Oficjalna dokumentacja techniczna dla języka Java SE 8 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Eclipse 4.4 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści w sekcjach: „Przeczytaj”, „Prezentacja multimedialna”, „Sprawdź się” jako materiał do lekcji powtórkowej.