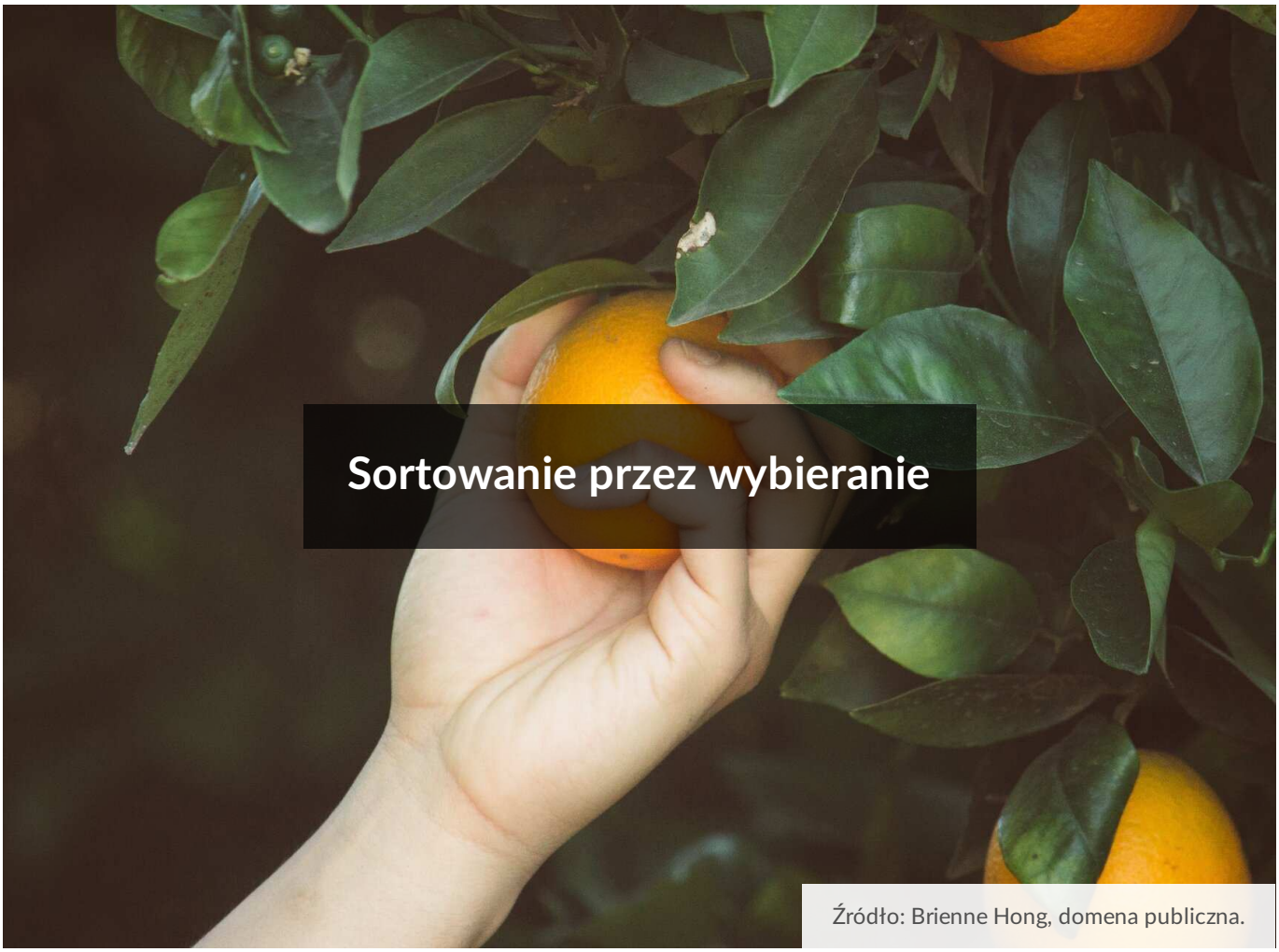
Sortowanie przez wybieranie

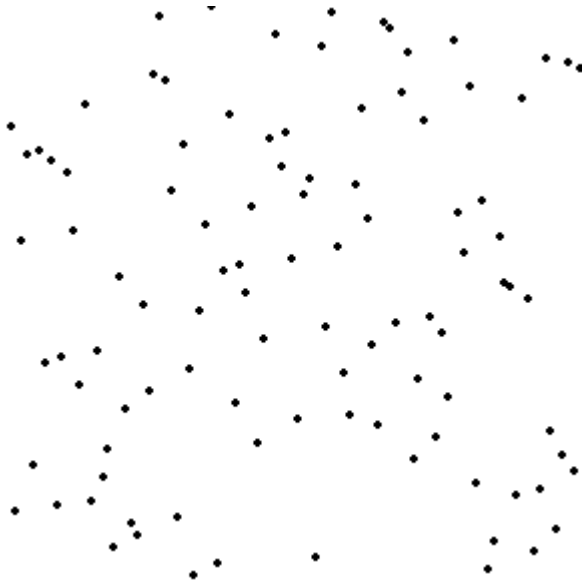
- Wprowadzenie
- Przeczytaj
- Gra edukacyjna
- Prezentacja multimedialna
- Dla nauczyciela



Źródło: Brienne Hong, domena publiczna.

> W tym e-materiale powtarzamy wiadomości ze szkoły podstawowej.

E-materiał Wstęp do algorytmów sortowania zaznajomił nas z podstawowymi informacjami dotyczącymi sortowania. Wiemy już, że różne algorytmy znajdują zastosowanie w różnych przypadkach.



Przykład działania sortowania przez wybieranie.
Źródło: Marco Polo, domena publiczna.

W tym e-materiale poznamy jeden z najprostszych algorytmów sortowania – przez wybieranie (wybór). W ten sposób sortować możemy np. wydania czasopism (chronologicznie), książki (alfabetycznie) etc.

Implementację algorytmu sortowania przez wybieranie przedstawiamy w e-materiałach:

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

Więcej zadań? Sortowanie przez wybieranie – zadania maturalne

> **Twoje cele**
>
> - Prześledzisz, czym jest sortowanie i jakie daje korzyści.
> - Przeanalizujesz przykłady algorytmów sortujących przez wybieranie.
> - Wykonasz kilka ćwiczeń związanych z tematem lekcji.

Przeczytaj

Sortowanie – przypomnienie

Sortowaniem nazywamy porządkowanie elementów w pewien ustalony sposób.

Istnieje wiele metod sortowania, różniących się zarówno sposobem działania, jak i efektywnością. W związku z tym algorytmy sortujące, w zależności od swojej złożoności algorytmicznej, różnią się czasem rozwiązywania danego problemu.

Do czego służy sortowanie?

Sortowanie to inaczej porządkowanie danych uwzględniające jakąś ich cechę. Jeżeli sortujemy liczby, taką cechą jest zazwyczaj ich wartość, więc sortujemy je rosnąco, nierosnąco, malejąco oraz niemalejąco. Sortować możemy nie tylko liczby, ale również litery czy ciągi znaków.

Algorytmy stabilne i niestabilne

Algorytmy stabilne to takie, w których elementy o takiej samej wartości nie zmienią pozycji względem siebie, natomiast algorytmy niestabilne mogą zmienić ich kolejność. Jeżeli algorytm niestabilny napotka dwa elementy o wartości 7, nie musi zamieniać ich miejscami, aby uzyskać posortowany zbiór. W zależności od implementacji, może to jednak zrobić.

Sortowanie przez wybieranie

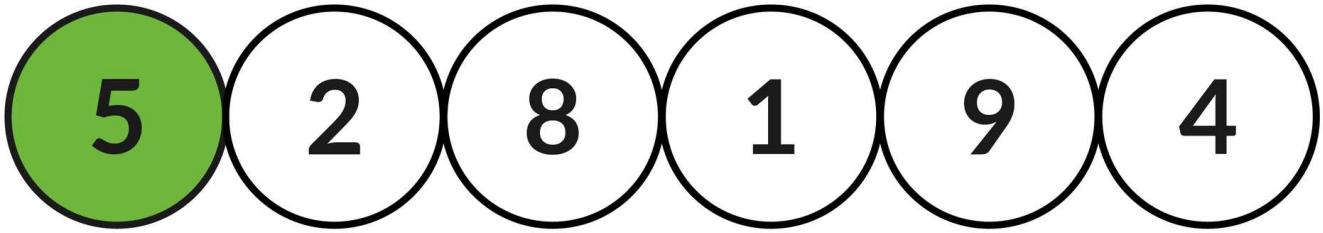
Metoda sortowania przez wybieranie w porządku rosnącym polega na znalezieniu najmniejszego elementu zbioru i umieszczeniu go na pierwszym miejscu. Obiekt, który się tam wcześniej znajdował, trafia na miejsce zajmowane dotychczas przez element o najmniejszej wartości klucza.

Następnie wśród pozostałych (jeszcze nieposortowanych) elementów znowu wyszukuje się ten o najmniejszej wartości klucza i umieszcza go na drugim miejscu (ponownie dokonując zamiany z dotychczasowym obiektem) itd.

W każdym cyklu zmniejsza się liczba nieposortowanych elementów. Opisane czynności wykonuje się aż do momentu, w którym pozostanie tylko jeden element w części nieposortowanej – na pewno wartość jego klucza jest największa w całym zbiorze.

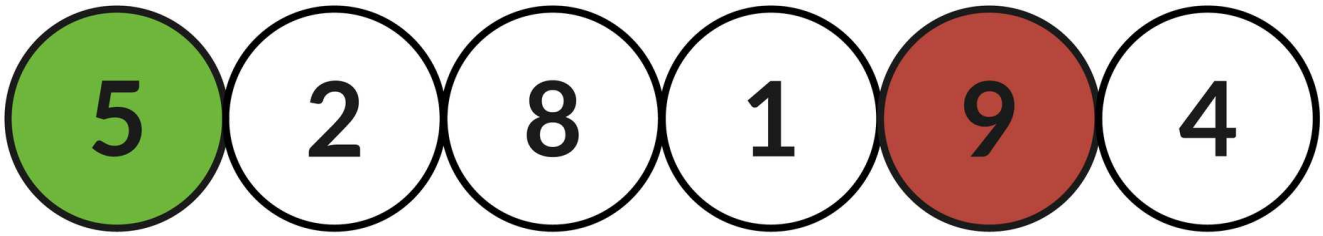
Kiedy sortujemy tablicę malejąco, rozpoczynamy od znalezienia elementu największego. Zamieniamy go miejscami z elementem, który znajdował się na pierwszej pozycji.

Kolorem zielonym zaznaczono miejsce przeznaczone dla pierwszego elementu. Ponieważ sortujemy zbiór malejąco, powinna się tu znaleźć największa liczba.



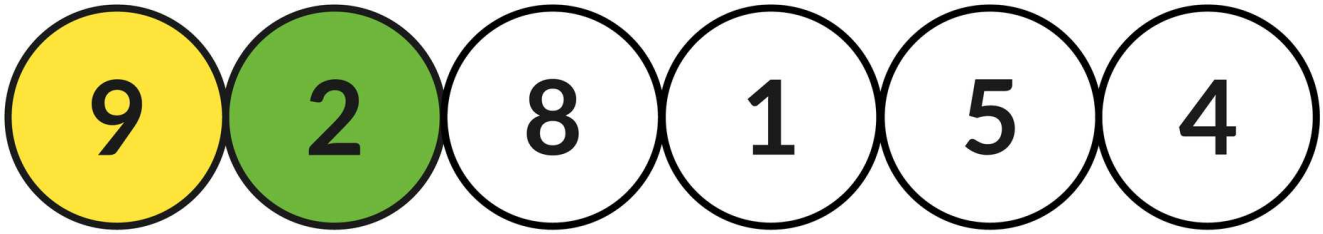
Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Jest nią zaznaczony kolorem czerwonym element o wartości 9. Zamieniamy go miejscami z będącą na pierwszej pozycji liczbą 5.



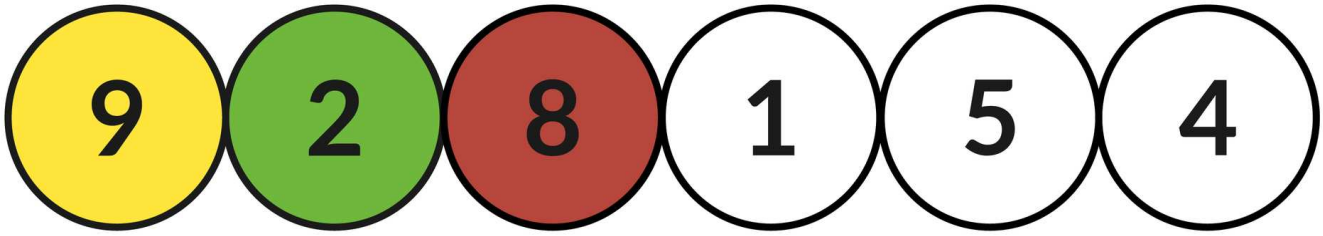
Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Na razie umieściliśmy jeden element we właściwym miejscu. Elementy już posortowane oznaczymy barwą żółtą. Teraz musimy znaleźć obiekt o największej wartości klucza w nieposortowanej części i zamienić go miejscami z liczbą 2 (pole to ma obecnie barwę zieloną).

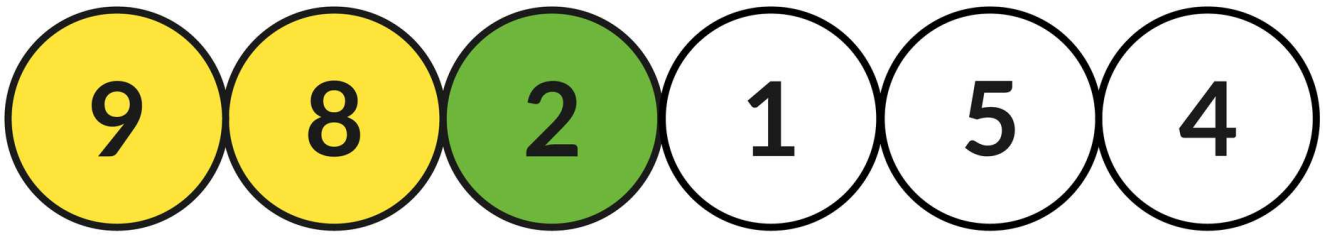


Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Element o wartości 8 zamieniamy miejscami z liczbą 2. Rozpoczynamy szukanie wartości, która powinna znaleźć się na trzeciej pozycji.

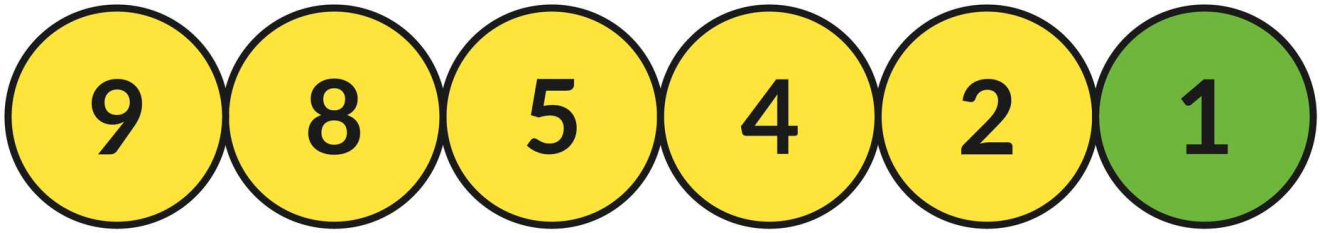


Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.



Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Cały proces trwa aż do odnalezienia przedostatniego elementu. Ostatnia wartość na pewno jest najmniejsza:



Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Algorytm w pseudokodzie

Zanim przejdziemy do przełożenia opisanego algorytmu na język C++, przedstawimy go w postaci pseudokodu. Załóżmy, że sortujemy zbiór liczb w porządku rosnącym.

Użyjemy przy tym mechanizmu pętli zagnieżdżonych. Pierwsza z nich, zewnętrzna, będzie odpowiadać za wskazywanie kolejnych pozycji w tablicy. Zapełnimy je liczbami o coraz większych wartościach.

O tym, która liczba powinna znaleźć się w miejscu wskazanym przez zewnętrzną pętlę, poinformuje nas pętla wewnętrzna. Dzięki niej wyszukamy najmniejsze liczby w nieposortowanej części tablicy.

Zaczniemy od zdefiniowania zmiennych. Będzie nam potrzebna tablica zawierająca elementy do posortowania:

```
zbior[] = tablica o rozmiarze n
```

Następnie tworzymy pierwszą, zewnętrzną pętlę. Wskaże ona kolejne elementy tablicy oprócz ostatniego. Jeżeli poprzednie liczby zostaną ułożone we właściwym porządku, to ostatnia na pewno znajdzie się tam, gdzie być powinna.

```
zbior[] = tablica o rozmiarze n

dla i = 0, 1, ..., długość(zbior) - 1 wykonuj

```

Dla każdej wartości `i` należy przeszukać nieposortowaną część tablicy i znaleźć element, który powinien zostać umieszczony w miejscu wskazywanym przez zmienną `i`.
W pierwszym cyklu będzie to pierwsze miejsce tablicy, w drugim drugie itd.

Zdefiniujemy teraz zmienną tymczasową `najmniejszyIndeks`. Ma ona wskazywać najmniejszy odnaleziony element; wstępnie przypiszemy jej wartość `i`. Zrobimy tak, ponieważ najmniejsza liczba od razu może znajdować się w miejscu wskazywanym przez zmienną `i` (w przypadku omówionej wcześniej przykładowej tablicy liczba 5 była tam, gdzie powinna zostać umieszczona).

Pora uruchomić pętlę wewnętrzną. Posłuży ona do odnalezienia najmniejszej liczby w nieposortowanej jeszcze części tablicy. Poszukiwania zaczną się od elementu o indeksie `i + 1`:

```
zbior[] = tablica o rozmiarze n

dla i = 0, 1, ..., długość(zbior) - 1 wykonuj
    najmniejszyIndeks = i

    dla j = i + 1, i + 2, ..., długość(zbior) wykonuj

```

Aby znaleźć najmniejszy element, musimy porównać każdy obiekt w nieposortowanej części tablicy z najmniejszą odszukaną dotychczas liczbą:

```
zbior[] = tablica o rozmiarze n

dla i = 0, 1, ..., długość(zbior) - 1 wykonuj
    najmniejszyIndeks = i

    dla j = i + 1, i + 2, ..., długość(zbior) - 1 wykonuj
        jeżeli zbior[j] < zbior[najmniejszyIndeks] to
            najmniejszyIndeks = j
```

Jeżeli okaże się, że sprawdzany właśnie element jest jeszcze mniejszy, uznajemy go za najmniejszą obecnie liczbę. Przenosimy ją na miejsce wskazane przez zmienną `i`. Oto gotowy algorytm zapisany w pseudokodzie:

```
zbior[] = tablica o rozmiarze n

dla i = 0, 1, ..., długość(zbior) - 1 wykonuj
    najmniejszyIndeks = i

```

```
    dla j = i + 1, i + 2, ..., długość(zbior) - 1 wykonuj
        jeżeli zbior[j] < zbior[najmniejszyIndeks] to
            najmniejszyIndeks = j

    zamienMiejscami(zbior[i], zbior[najmniejszyIndeks])
```

Gdybyśmy chcieli natomiast posortować zbiór w porządku malejącym, należałoby wyszukiwać największe liczby w nieposortowanej części zbioru.

Słownik

> **algorytmy niestabilne**
>
> algorytmy, w których elementy o równej wartości nie występują po posortowaniu w tej samej kolejności jaką miały w zbiorze nieposortowanym
> **algorytmy stabilne**
>
> w przypadku algorytmów stabilnych w uporządkowanej tablicy nie zmienia się kolejność elementów o tych samych wartościach klucza – przykładowo, gdy w tablicy znajdą się obok siebie dwie liczby o wartości 6, to nie zostaną one zamienione miejscami
> **sortowanie szybkie**
>
> algorytm rekurencyjny wynaleziony w 1962 r. przez brytyjskiego informatyka Tony'ego Hoare'a; opiera się na zasadzie dziel i zwyciężaj
> **wyszukiwanie binarne**
>
> metoda odnajdowania elementów w uporządkowanych zbiorach; polega ona na wielokrotnym dzieleniu przeszukiwanego zbioru na dwie części i porównywania wartości szukanej z elementem znajdującym się w środku dzielonego obszaru
> **zagnieżdżona pętla**
>
> pętla działająca wewnątrz innej pętli

Gra edukacyjna

Polecenie 1

Sprawdź swoją wiedzę, biorąc udział w grze.

Test

Sprawdź swoją wiedzę, biorąc udział w grze

Poziom trudności:

łatwy

Limit czasu:

7 min

Twój ostatni wynik:

-

Uruchom

Polecenie 2

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

Prezentacja multimedialna

> **Polecenie 1**
>
> Przeanalizuj zaprezentowaną implementację algorytmu sortowania przez wybór w porządku niemalejącym. Porównaj efektywność takiego sortowania, gdy dane są ułożone w przypadku pesymistycznym oraz optymistycznym.

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna
- `liczby` – `n`-elementowa tablica liczb naturalnych rozłożonych losowo

Wynik:

Program sortuje tablicę `liczby` w porządku niemalejącym.

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PT5YE0lsB

Tworzymy tablicę liczb z wartościami rozłożonymi losowo.

```
liczby[7] = {2, 65, 21, 17, 50, 1, 9}
```

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PT5YE0lsB

Definiujemy zmienne: `min_index` oznaczamy jako najmniejszy indeks aktualnie sprawdzanej

wartości, `temp` – zmienna pomocnicza, która pozwoli wygodnie zamieniać liczby miejscami w tablicy.

```
liczby[7] = {2, 65, 21, 17, 50, 1, 9}
min_index = 0
temp = 0
```

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PT5YE0lsB

Tworzymy pierwszą pętlę `for`, która umożliwi sprawdzenie każdej liczby w tablicy.

```
liczby[7] = {2, 65, 21, 17, 50, 1, 9}
min_index = 0
temp = 0

dla i = 1, 2, ..., długość (liczby) wykonuj:
```

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PT5YE0lsB

Przypisujemy zmiennej `min_index` wartość indeksu sprawdzanej liczby.

```
liczby[7] = {2, 65, 21, 17, 50, 1, 9}
min_index = 0
temp = 0

```

```
5 dla i = 1, 2, ..., długość
  (liczby) wykonuj:
6     min_index = i
```

5

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PT5YE0lsB

Tworzymy zagnieżdżoną pętlę `for`, która będzie znajdować odpowiednie miejsce w tablicy dla sprawdzanej liczby.

```
1  liczby[7] = {2, 65, 21,
   17, 50, 1, 9}
2  min_index = 0
3  temp = 0
4
5  dla i = 1, 2, ..., długość
   (liczby) wykonuj:
6      min_index = i
7
8      dla j = i + 1, i + 2,
   ..., długość (liczby)
   wykonuj:
9          jeżeli liczby[j] <
   liczby[min_index] wykonaj:
10             min_index = j
```

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PT5YE0lsB

Jeśli aktualnie sprawdzana liczba jest mniejsza niż zdefiniowane dotychczas minimum, zastępujemy dotychczasowe minimum nową wartością oraz modyfikujemy minimalny indeks.

Tym samym zakończyliśmy pisanie naszego algorytmu.

```
liczby[7] = {2, 65, 21, 17, 50, 1, 9}
min_index = 0
temp = 0

dla i = 1, 2, ..., długość (liczby) wykonuj:
    min_index = i

    dla j = i + 1, i + 2, ..., długość (liczby) wykonuj:
        jeżeli liczby[j] < liczby[min_index] wykonaj:
            min_index = j

    jeżeli min_index != i wykonaj:
        temp = liczby[i]
        liczby[i] = liczby[min_index]
        liczby[min_index] = temp
```

7

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PT5YE0IsB

Twoim zadaniem jest teraz sprawdzić i porównać, ile razy algorytm wykona operacje zamiany miejsc liczb dla trzech zbiorów danych, z których każdy ma zupełnie inne początkowe ułożenie liczb.

I. Ułożenie pesymistyczne (dane są ułożone odwrotnie niż powinny):

{65, 50, 21, 17, 9, 2, 1}

II. Ułożenie średnie (losowe):

{2, 65, 21, 17, 50, 1, 9}

III. Ułożenie optymistyczne (dane są wstępnie ułożone dokładnie tak, jak powinny):

{1, 2, 9, 17, 21, 50, 65}

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PT5YE0lsB

Po sprawdzeniu działania algorytmu dla każdego z powyższych zbiorów liczb otrzymujemy następujące wyniki:

I. Ułożenie pesymistyczne: 3 zmiany miejsc

II. Ułożenie losowe: 5 zmian miejsc

III. Ułożenie optymistyczne: 0 zmian miejsc

Im więcej operacji musi wykonać algorytm, tym dłuższy będzie czas jego wykonywania, w tym przypadku sortowania. Żeby uniknąć takich sytuacji, warto zadbać o zbliżone do oczekiwanego ułożenie danych.

9

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PT5YE0lsB

Jak najkrótszy czas sortowania ma szczególne znaczenie w programach, w których mamy do czynienia z różnymi rankingami i porównaniami (np. serwisy transakcyjne, aplikacje giełdowe, gry sportowe online).

Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Pokaż ćwiczenia:

Ćwiczenie 1

Zdecyduj, które stwierdzenia są prawdziwe, a które fałszywe.

Stwierdzenie	Prawda	Fałsz
Dwa dowolne stabilne algorytmy sortujące malejąco, które są zaimplementowane w różny sposób, zwracają taki sam rezultat.	○	○
Każdy algorytm sortujący ma dokładnie taki sam sposób działania.	○	○
Efektywność i szybkość wykonywania algorytmów zależy wyłącznie od używanego sprzętu komputerowego	○	○

Ćwiczenie 2

Zaznacz wszystkie poprawne odpowiedzi. Wskaż czynniki, jakie nie są brane pod uwagę podczas porównywania efektywności algorytmów.

- [ ] skomplikowanie algorytmu
- [ ] czas wykonania oraz zużycie zasobów
- [ ] parametry sprzętu komputerowego

Ćwiczenie 3

Wskaż, jaki przypadek ułożenia danych reprezentuje tablica: [9, 7, 6, 5, 4, 3, 2, 1], przy założeniu, że chcemy posortować je rosnąco.

- ○ przypadek optymistyczny
- ○ przypadek pesymistyczny
- ○ przypadek typowy

Ćwiczenie 4

Wyjaśnij, jaki wpływ na sortowanie mają ilość danych i ich struktura.

Ćwiczenie 5

Wyjaśnij, w jaki sposób działa sortowanie przez wybieranie.

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie przez wybieranie

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy

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. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

1) planuje kolejne kroki rozwiązywania problemu, z uwzględnieniem podstawowych etapów myślenia komputacyjnego (określenie problemu, definicja modeli i pojęć, znalezienie rozwiązania, zaprogramowanie i testowanie rozwiązania).

4) porównuje działanie różnych algorytmów dla wybranego problemu, analizuje algorytmy na podstawie ich gotowych implementacji;

5) sprawdza poprawność działania algorytmów dla przykładowych danych.

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):

- Prześledzisz, czym jest sortowanie i jakie daje korzyści.
- Przeanalizujesz przykłady algorytmów sortujących przez wybieranie.
- Wykonasz kilka ćwiczeń związanych z tematem lekcji.

Strategie nauczania:

- konstruktywizm;
- konektywizm;
- myślenie komputacyjne.

Metody i techniki nauczania:

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

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 przez wybieranie". Uczniowie mają zapoznać się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć. Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.

Faza realizacyjna:

1. **Praca z tekstem.** Uczniowie przystępują do cichego czytania tekstu zawartego w sekcji „Przeczytaj", jeśli nauczyciel - na podstawie raportu na platformie - uważa, że

przygotowanie uczniów jest wystarczające, może pominąć tę czynność.
2. Uczniowie zapoznają się z prezentacją w sekcji „Prezentacja multimedialna Sprawdź się” i wykonują dołączone do niej polecenie.
3. **Ćwiczenie umiejętności.** Prowadzący zapowiada uczniom, że w kolejnym kroku będą rozwiązywać ćwiczenia nr 1-8 z sekcji „Prezentacja multimedialna”. Każdy z uczniów robi to samodzielnie. Po ustalonym czasie wybrani uczniowie przedstawiają rozwiązania. Nauczyciel w razie potrzeby koryguje odpowiedzi, dopowiada istotne informacje, udziela uczniom informacji zwrotnej.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat lekcji zawarty w sekcji „Wprowadzenie” i inicjuje krótką rozmowę na temat zrealizowanych celów (czego uczniowie się nauczyli).
2. Nauczyciel prosi uczniów o podsumowanie zgromadzonej wiedzy.

Praca domowa:

1. Uczniowie wykonują interaktywny test „Sprawdź swoja wiedzę” z sekcji „Gra edukacyjna”.

Wskazówki metodyczne:

- Nauczyciel może wykorzystać multimedium w sekcji „Przeczytaj” do pracy przed lekcją. Uczniowie zapoznają się z jego treścią i przygotowują do pracy na zajęciach w ten sposób, żeby móc samodzielnie rozwiązać zadania dołączone do e-materiału „Sortowanie przez wybieranie”.