

Sortowanie bąbelkowe $O(n^2)$

Wejście: lista t , która zawiera n elementów

Wyjście: posortowana lista t

1. $j \leftarrow n - 1$
2. Dopóki $j \geq 1$
 - 2.1. $i \leftarrow 1$
 - 2.2. Dopóki $i \leq j$
 - 2.2.1. Jeżeli $t[i] > t[i + 1]$
 - 2.2.1.1. zamień($t[i]$, $t[i + 1]$)
 - 2.2.2. $i \leftarrow i + 1$
 - 2.3. $j \leftarrow j - 1$

Sortowanie przez wstawianie $O(n^2)$

Wejście: lista t , która zawiera n elementów

Wyjście: posortowana lista t

1. $j \leftarrow n - 1$
2. Dopóki $j \geq 1$
 - 2.1. $p \leftarrow t[j]$
 - 2.2. $i \leftarrow j + 1$
 - 2.3. Dopóki $i \leq n$ oraz $p > t[i]$
 - 2.3.1. $t[i - 1] \leftarrow t[i]$
 - 2.3.2. $i \leftarrow i + 1$
 - 2.4. $t[i - 1] \leftarrow p$
 - 2.5. $j \leftarrow j - 1$

Sortowanie przez wybór $O(n^2)$

Wejście: lista t , która zawiera n elementów

Wyjście: posortowana lista t

1. $j \leftarrow 1$
2. Dopóki $j < n$
 - 2.1. $p \leftarrow j$
 - 2.2. $i \leftarrow j + 1$
 - 2.3. Dopóki $i \leq n$
 - 2.3.1. Jeżeli $t[i] < t[p]$
 - 2.3.1.1. $p \leftarrow i$
 - 2.3.2. $i \leftarrow i + 1$
 - 2.4. zamień($t[j]$, $t[p]$)
 - 2.5. $j \leftarrow j + 1$

Wyszukiwanie binarne $O(\log_2 n)$

Wejście: posortowana lista t , która zawiera n elementów oraz szukany element k .

Wyjście: indeks, na którym znajduje się poszukiwany element

1. lewo $\leftarrow 1$
2. prawo $\leftarrow n$
3. Dopóki lewo $<$ prawo
 - 3.1. srodek $\leftarrow (\text{lewo} + \text{prawo}) \div 2$
 - 3.2. Jeżeli $t[\text{srodek}] < k$
 - 3.2.1 lewo $\leftarrow \text{srodek} + 1$
 - 3.3. W przeciwnym razie
 - 3.3.1 prawo $\leftarrow \text{srodek}$
4. Jeżeli $t[\text{lewo}] = k$
 - 4.1. Zwróć lewo
5. W przeciwnym razie
 - 5.1. Zwróć Brak

Algorytm Euklidesa $O(\log_2(m + n))$

Wejście: liczby n i m .

Wyjście: największy wspólny dzielnik liczb n i m .

1. Dopóki $n \neq 0$
 - 1.1. $p \leftarrow m \bmod n$

```
1.2. m ← n
1.3. n ← p
2. Zwróć m
```

Sito Eratostenesa $O(n \cdot \log(\log n))$

Wejście: liczba całkowita $n > 1$ oraz lista t , która jest indeksowana od 2 .. n .

Wyjście: liczby pierwsze, które są mniejsze od n .

```
1. i ← 2
2. Dopóki i < n
  2.1. t[i] ← True
  2.2. i ← i + 1
3. i ← 2
4. Dopóki i < √n
  4.1. Jeżeli t[i]
    4.1.1. k ← 2
    4.1.2. j ← i · k
    4.1.3. Dopóki j < n
      4.1.3.1. t[j] ← False
      4.1.3.2. k ← k + 1
      4.1.3.3. j ← i · k
  4.2. i ← i + 1
5. i ← 2
6. Dopóki i < n
  6.1. Jeżeli t[i]
    6.1.1. Wypisz(i)
```

Zadania

Zadanie 1. Zaimplementuj algorytm sortowania bąbelkowego (dla listy indeksowanej od 0) w taki sposób, aby sortował dane rosnąco.

Wejście: Zmienna lista przechowująca dane dowolnego (tego samego) typu.

Wyjście: Na ekranie pojawiają się posortowane dane.

Warunki poprawności zadania: Na ekranie powinny pojawić się posortowane dane. Upewnij się czy algorytm został zaimplementowany prawidłowo (zgodnie z pseudokodem) oraz prawidłowo sortuje dane. Przetestuj działanie dla różnej ilości danych.

Zadanie 2. Zaimplementuj algorytm sortowania bąbelkowego (dla listy indeksowanej od 0) w taki sposób, aby sortował dane malejąco.

Wejście: Zmienna lista przechowująca dane dowolnego (tego samego) typu.

Wyjście: Na ekranie pojawiają się posortowane dane.

Warunki poprawności zadania: Na ekranie powinny pojawić się posortowane dane. Upewnij się czy algorytm został zaimplementowany prawidłowo (zgodnie z pseudokodem) oraz prawidłowo sortuje dane. Przetestuj działanie dla różnej ilości danych.

Zadanie 3. Zaimplementuj algorytm sortowania przez wstawianie (dla listy indeksowanej od 0) w taki sposób, aby sortował dane rosnąco.

Wejście: Zmienna lista przechowująca dane dowolnego (tego samego) typu.

Wyjście: Na ekranie pojawiają się posortowane dane.

Warunki poprawności zadania: Na ekranie powinny pojawić się posortowane dane. Upewnij się czy algorytm został zaimplementowany prawidłowo (zgodnie z pseudokodem) oraz prawidłowo sortuje dane. Przetestuj działanie dla różnej ilości danych.

Zadanie 4. Zaimplementuj algorytm sortowania przez wstawianie (dla listy indeksowanej od 0) w taki sposób, aby sortował dane malejąco.

Wejście: Zmienna lista przechowująca dane dowolnego (tego samego) typu.

Wyjście: Na ekranie pojawiają się posortowane dane.

Warunki poprawności zadania: Na ekranie powinny pojawić się posortowane dane. Upewnij się czy algorytm został zaimplementowany prawidłowo (zgodnie z pseudokodem) oraz prawidłowo sortuje dane. Przetestuj działanie dla różnej ilości danych.

Zadanie 5. Zaimplementuj algorytm sortowania przez wybór (dla listy indeksowanej od 0) w taki sposób, aby sortował dane rosnąco.

Wejście: Zmienna lista przechowująca dane dowolnego (tego samego) typu.

Wyjście: Na ekranie pojawiają się posortowane dane.

Warunki poprawności zadania: Na ekranie powinny pojawić się posortowane dane. Upewnij się czy algorytm został zaimplementowany prawidłowo (zgodnie z pseudokodem) oraz prawidłowo sortuje dane. Przetestuj działanie dla różnej ilości danych.

Zadanie 6. Zaimplementuj algorytm sortowania przez wybór (dla listy indeksowanej od 0) w taki sposób, aby sortował dane malejąco.

Wejście: Zmienna lista przechowująca dane dowolnego (tego samego) typu.

Wyjście: Na ekranie pojawiają się posortowane dane.

Warunki poprawności zadania: Na ekranie powinny pojawić się posortowane dane. Upewnij się czy algorytm został zaimplementowany prawidłowo (zgodnie z pseudokodem wybrane przez Ciebie algorytmu) oraz prawidłowo sortuje dane. Przetestuj działanie dla różnej ilości danych.

Zadanie 7. Zaimplementuj iteracyjną wersję algorytmu przeszukiwania binarnego (dla listy indeksowanej od 0) w taki sposób, aby zwrócił indeks elementu, na którym znajduje się element. Jeżeli element się nie znajduje powinna zostać zwrócone -1 lub wartość None

Wejście: Zmienna lista przechowująca dane dowolnego (tego samego) typu.

Wyjście: Na ekranie pojawiają się informacja czy element znajduje się w liście oraz indeks na którym znajduje się element (jeżeli istnieje).

Warunki poprawności zadania: Na ekranie powinna pojawić się informacja czy element znajduje się w liście. Upewnij się czy algorytm został zaimplementowany prawidłowo (zgodnie z pseudokodem) oraz prawidłowo sprawdza czy element znajduje się w liście. Przetestuj działanie dla różnej ilości danych.

Zadanie 8. Zaimplementuj rekurencyjną wersję algorytmu przeszukiwania binarnego (dla listy indeksowanej od 0) w taki sposób, aby zwrócił indeks elementu, na którym znajduje się element. Jeżeli element się nie znajduje powinna zostać zwrócone -1 lub wartość None

Wejście: Zmienna lista przechowująca dane dowolnego (tego samego) typu.

Wyjście: Na ekranie pojawiają się informacja czy element znajduje się w liście oraz indeks na którym znajduje się element (jeżeli istnieje).

Warunki poprawności zadania: Na ekranie powinna pojawić się informacja czy element znajduje się w liście. Upewnij się czy algorytm został zaimplementowany prawidłowo oraz prawidłowo sprawdza czy element znajduje się w liście. Przetestuj działanie dla różnej ilości danych.

Zadanie 9. Zaimplementuj rekurencyjną wersję algorytmu Euklidesa.

Wejście: Zmienne m i n przechowujące liczby całkowite.

Wyjście: Na ekranie pojawiają się największy wspólny dzielnik liczb m i n.

Warunki poprawności zadania: Po podaniu liczb powinien pojawić się ich największy wspólny dzielnik. Upewnij się czy algorytm został zaimplementowany prawidłowo oraz wyznacza największy wspólny dzielnik.

Zadanie 10. Korzystając z algorytmu wyznaczania liczb pierwszych - sito Eratostenesa (dla listy indeksowanej od 0) wyznacz wszystkie liczby pierwsze mniejsze od n .

Podpowiedź: W celu wyliczenia pierwiastka zaimportuj z biblioteki `math` i skorzystaj z funkcji `sqrt`

Wejście: Zmienna `n` przechowująca liczbę całkowitą.

Wyjście: Na ekranie pojawiają się wszystkie liczby mniejsze od n .

Warunki poprawności zadania: Po podaniu liczby powinny się pojawić wszystkie liczby pierwsze mniejsze od podanej liczby.