



dr inż. Michał Malinowski
michal.malinowski@uth.edu.pl

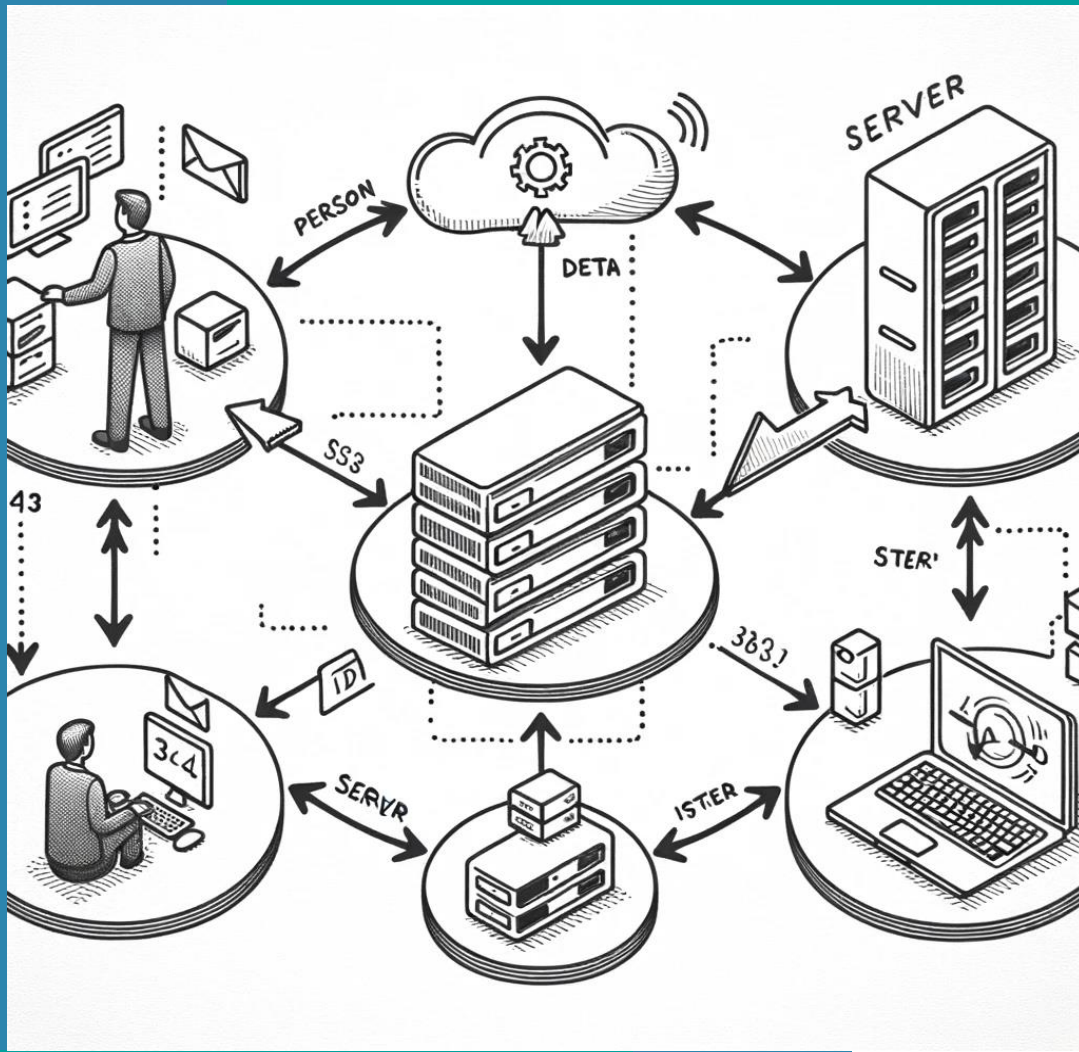


[https://www.linkedin.com/in/
michał-malinowski-malkon/](https://www.linkedin.com/in/michał-malinowski-malkon/)

Sortowanie

Ćwiczenie 04

ALGORYTMY
I STRUKTURY DANYCH



Zagadnienia (2h)

- [Wprowadzenie do sortowania](#)
- [Sortowanie bąbelkowe](#)
 - [Zadanie: Sortowanie bąbelkowe – kod](#)
 - [Zadanie: Sortowanie bąbelkowe - schemat](#)
- [Sortowanie przez wybieranie](#)
 - [Zadanie: Sortowania przez wybieranie - kod](#)
- [Sortowanie przez wstawianie](#)
 - [Zadanie: Sortowanie przez wstawianie - kod](#)

Wyrażenia regularne

RegExp (Javascript)

reprezentuje wyrażenia regularne, które są wzorcami służącymi do dopasowywania kombinacji znaków w ciągach tekstowych. Wykorzystywane są one do wyszukiwania, walidacji, i manipulacji tekstem.

test(): Testuje, czy wzorzec występuje w podanym ciągu i zwraca true lub false.

exec(): Wykonuje wyszukiwanie dopasowania w ciągu i zwraca wynik w postaci tablicy lub null.

flags: Zwraca ciąg znaków oznaczających flagi wyrażenia regularnego.

source: Zwraca tekst wzorca wyrażenia regularnego.

```
let wzorzec = /ab+c/i;
console.log('Czy wzorzec pasuje do "Abc"?', wzorzec.test('Abc')); // Wynik: true

let wynik = wzorzec.exec('Pamiętaj o abcd i ABC');
console.log('Wynik dopasowania:', wynik[0]); // Wynik: "Abc"

console.log('Flagi wzorca:', wzorzec.flags); // Wynik: "i"
console.log('Tekst wzorca:', wzorzec.source); // Wynik: "ab+c"

let regexGlobalny = /abc/g;
console.log('Czy wzorzec globalny?', regexGlobalny.global); // Wynik: true
```

Wyrażenia regularne

Symbol/Wzorzec	Opis
.	Dopasowuje dowolny pojedynczy znak, z wyjątkiem znaku nowej linii.
^	Dopasowuje początek ciągu znaków lub linii.
\$	Dopasowuje koniec ciągu znaków lub linii.
*	Dopasowuje zero lub więcej wystąpień poprzedzającego elementu.
+	Dopasowuje jedno lub więcej wystąpień poprzedzającego elementu.
?	Dopasowuje zero lub jedno wystąpienie poprzedzającego elementu.
{n}	Dopasowuje dokładnie n wystąpień poprzedzającego elementu.
{n,}	Dopasowuje n lub więcej wystąpień poprzedzającego elementu.
{n,m}	Dopasowuje od n do m wystąpień poprzedzającego elementu.
[abc]	Dopasowuje dowolny pojedynczy znak z podanych (w tym przypadku a, b lub c).
[^abc]	Dopasowuje dowolny pojedynczy znak, który nie jest wymieniony (w tym przypadku nie a, b ani c).
(abc)	Dopasowuje dokładnie sekwencję abc (grupowanie).
\s	Dopasowuje dowolny biały znak, w tym spację, tabulację, znak nowej linii.
\S	Dopasowuje dowolny znak inny niż biały znak.
\d	Dopasowuje dowolną cyfrę (0-9).
\D	Dopasowuje dowolny znak, który nie jest cyfrą.
\w	Dopasowuje dowolny znak alfanumeryczny, w tym cyfrę lub podkreślenie (_).
\W	Dopasowuje dowolny znak, który nie jest znakiem alfanumerycznym ani podkreśleniem.
\b	Dopasowuje granicę słowa, czyli pozycję pomiędzy znakiem \w a \W (lub na początku/koniec ciągu).
\B	Dopasowuje pozycję, która nie jest granicą słowa.
\\	Używany do escapowania znaków specjalnych, np. \\ dopasowuje literał kropki.

```
const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;  
const testEmail1 = "przykladowy.email@example.com";  
const testEmail2 = "nieprawidlowyemail@przyklad";
```

```
console.log(emailRegex.test(testEmail1)); // Wynik: true  
console.log(emailRegex.test(testEmail2)); // Wynik: false
```

```
const kodPocztowyRegex = /^[0-9]{2}-[0-9]{3}$/;  
const testKodPocztowy1 = "00-123";  
const testKodPocztowy2 = "00123";
```

```
console.log(kodPocztowyRegex.test(testKodPocztowy1)); // Wynik: true  
console.log(kodPocztowyRegex.test(testKodPocztowy2)); // Wynik: false
```

```
const peselRegex = /^[0-9]{11}$/;  
const testPesel1 = "85050515836";  
const testPesel2 = "1234567890A";
```

```
console.log(peselRegex.test(testPesel1)); // Wynik: true  
console.log(peselRegex.test(testPesel2)); // Wynik: false
```

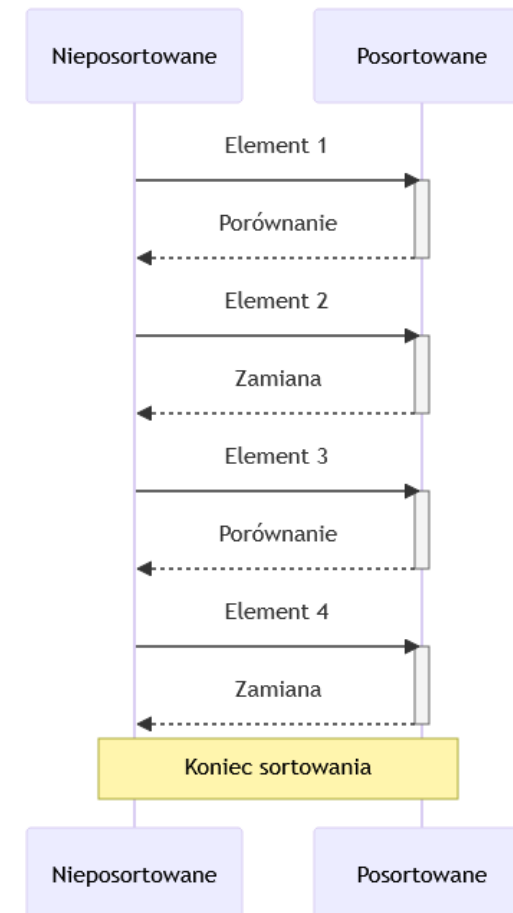
Wprowadzenie do sortowania

Sortowanie

to proces uporządkowania zbioru danych według określonego kryterium, np.: wartości liczbowej, alfabetycznej itd.

Cel Sortowania

- Ułatwienie przeszukiwania danych
- Optymalizacja wykorzystania zasobów
- Przygotowanie danych do dalszych operacji



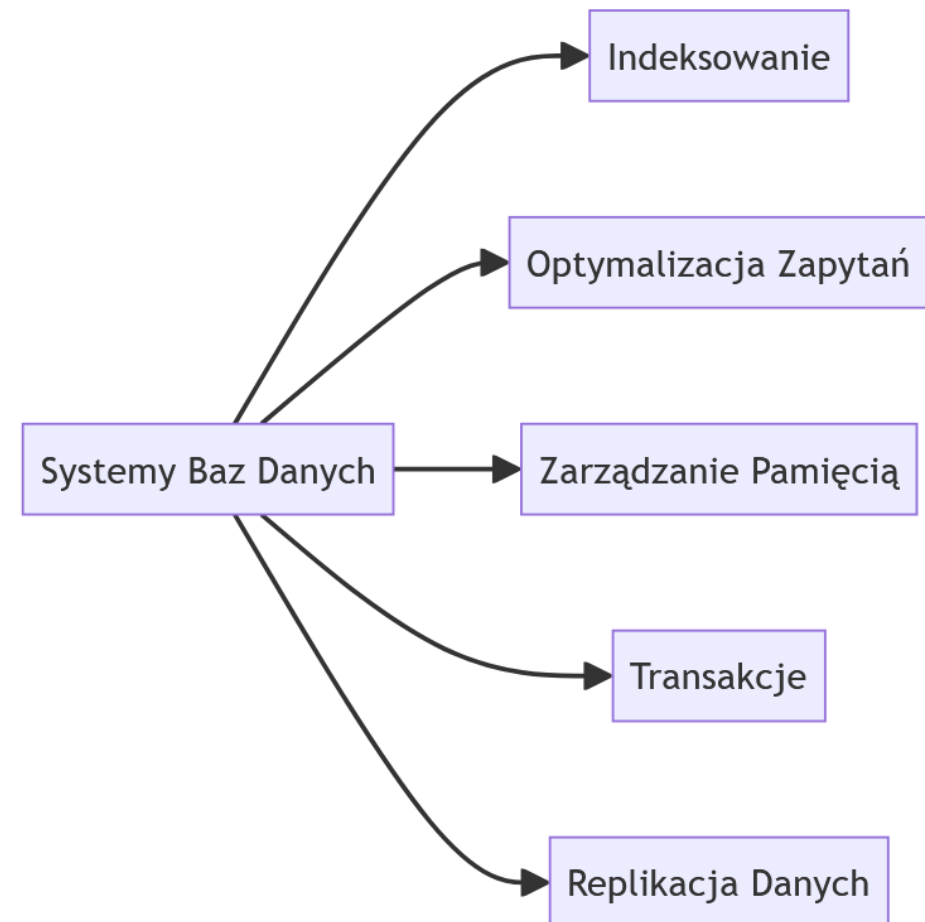
Wprowadzenie do sortowania

Zastosowania w Praktyce

- Wyszukiwarki internetowe
- Systemy baz danych
- Analiza danych naukowych
- Algorytmy grafowe

Kryteria Oceny Algorytmów Sortujących

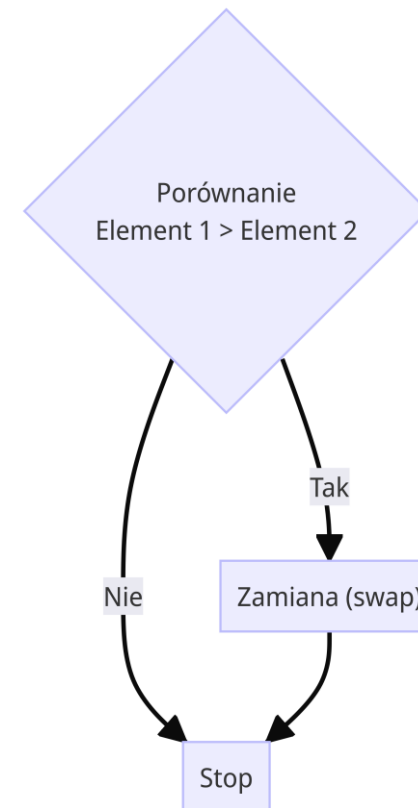
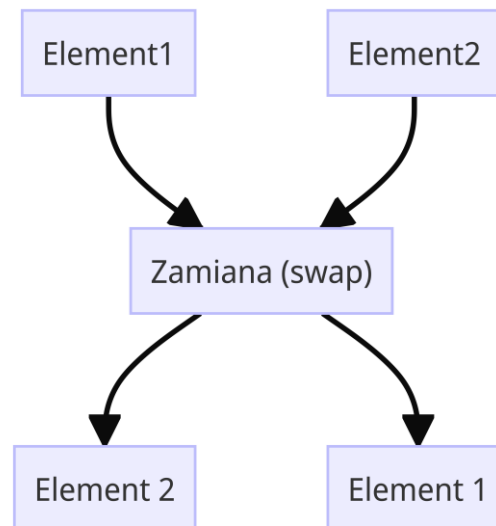
- Złożoność czasowa: Jak szybko algorytm sortuje dane?
- Złożoność pamięciowa: Ile dodatkowej pamięci jest potrzebne?



Wprowadzenie do sortowania

Podstawowe Terminy

- Klucz/Atrybut Elementu, według którego sortujemy
- Porównanie: Operacja ustalająca, który element jest większy/mniejszy
- Zamiana (swap): Operacja zamiany miejscami dwóch elementów



Sortowanie bąbelkowe

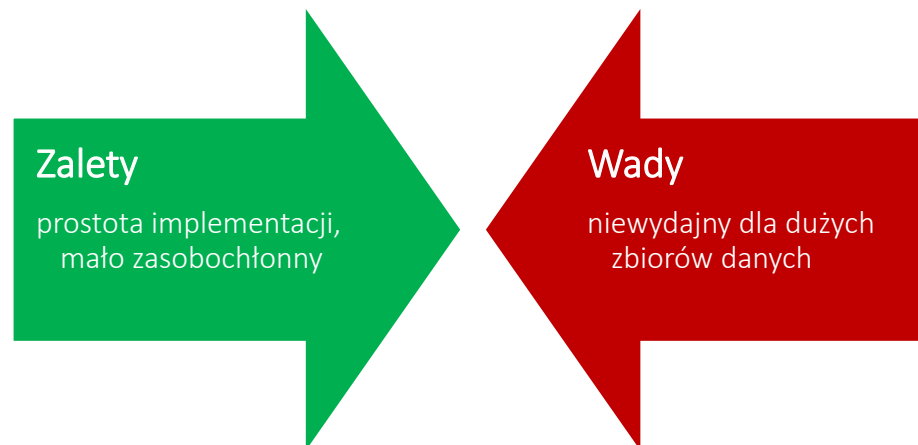
Sortowanie bąbelkowe (Bubble Sort)

jest jednym z najstarszych algorytmów sortujących.

Zasada działania opiera się na cyklicznym porównywaniu par sąsiadujących elementów i zamianie ich kolejności w przypadku niespełnienia kryterium porządkowego zbioru. Operację tę wykonujemy dotąd, aż cały zbiór zostanie posortowany.

Klasa złożoność czasowej

$O(n^2)$



Sortowanie bąbelkowe

Przykład

Mamy zbiór liczb {4, 5, 3, 3, 6}

Iteracja	Przed	Porównanie	Zamiana	Po
1	{4, 5, 3, 3, 6}	4, 5	Nie	{4, 5, 3, 3, 6}
1	{4, 5, 3, 3, 6}	5, 3	Tak	{4, 3, 5, 3, 6}
1	{4, 3, 5, 3, 6}	5, 3	Tak	{4, 3, 3, 5, 6}
1	{4, 3, 3, 5, 6}	5, 6	Nie	{4, 3, 3, 5, 6}
2	{4, 3, 3, 5, 6}	4, 3	Tak	{3, 4, 3, 5, 6}
2	{3, 4, 3, 5, 6}	4, 3	Tak	{3, 3, 4, 5, 6}
2	{3, 3, 4, 5, 6}	4, 5	Nie	{3, 3, 4, 5, 6}
3	{3, 3, 4, 5, 6}	3, 3	Nie	{3, 3, 4, 5, 6}
3	{3, 3, 4, 5, 6}	3, 4	Nie	{3, 3, 4, 5, 6}
4	{3, 3, 4, 5, 6}	3, 3	Nie	{3, 3, 4, 5, 6}

Sortowanie bąbelkowe

Algorytm

1. Inicjalizacja:
 - zainicjalizuj zmienną rozmiar jako długość tablicy tablica.
2. Iteracja zewnętrzna:
 - iteruj przez tablicę z indeksem i od 0 do rozmiar - 1.
3. Iteracja wewnętrzna:
 - w każdym przejściu pętli zewnętrznej, iteruj od j = 0 do rozmiar - i - 1.
4. Porównanie i zamiana:
 - porównaj tablica[j] i tablica[j + 1].
 - zamień je miejscami, jeśli tablica[j] > tablica[j + 1].
5. Zakończenie iteracji zewnętrznej:
 - wróć do kroku 3 aż przejdiesz przez całą tablicę.
6. Zwrócenie wyniku:
 - zwróć posortowaną tablicę.

Zadanie: Sortowanie bąbelkowe - kod

Zadanie

Napisz kod JavaScript sortowania bąbelkowego

Przetestuj go dla zbioru liczb {4, 5, 3, 3, 6}

Zadanie: Sortowanie bąbelkowe - kod

Rozwiązanie

```
function sortujBabelkowo(tablica) {  
  let rozmiar = tablica.length;  
  
  // Zewnetrzna petla iteruje przez cala tablice  
  for (let i = 0; i < rozmiar; i++) {  
    // Wewnetrzna petla porownuje sasiednie elementy  
    for (let j = 0; j < rozmiar - i - 1; j++) {  
      // Jesli sasiednie elementy sa w zlej kolejnosci, zamien je miejscami  
      if (tablica[j] > tablica[j + 1]) {  
        [tablica[j], tablica[j + 1]] = [tablica[j + 1], tablica[j]];  
      }  
    }  
  }  
  
  return tablica;  
}  
  
// Testowanie algorytmu  
const nieposortowanaTablica = [4, 5, 3, 3, 6];  
const posortowanaTablica = sortujBabelkowo(nieposortowanaTablica);  
console.log("Posortowana tablica:", posortowanaTablica);
```



Zadanie: Sortowanie bąbelkowe - schemat

Zadanie (2 pkt)

Narysuj schemat blokowy sortowania bąbelkowego

Narzędzie: <https://online.visual-paradigm.com/pl/diagrams/features/flowchart-tool/>



Przedstawienie wyniku

- Dokument tekstowy (np.: PDF, Word)
- Umieszczenie wyniku w Moodle

Termin

- Podany przy zadaniu w Moodle (około 2 tygodnie)

Sortowania przez wybieranie

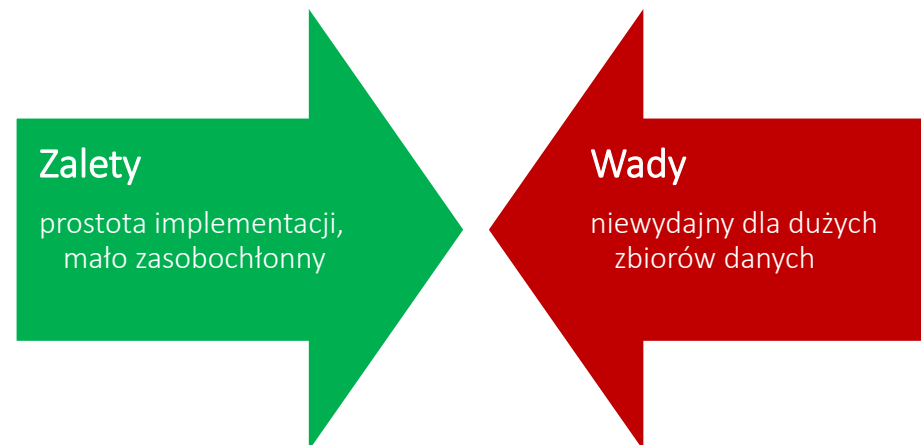
Sortowania przez wybieranie (Selection Sort)

jest jednym z najprostszych algorytmów sortujących.

Jego działanie polega na wybieraniu najmniejszego (lub największego, w zależności od rodzaju sortowania) elementu z niesortowanej części tablicy i zamianie go z pierwszym elementem. Następnie, algorytm przechodzi do kolejnej iteracji, pomijając już posortowane elementy.

Klasa złożoność czasowej

$O(n^2)$



Sortowania przez wybieranie

Przykład

Mamy zbiór liczb {4, 5, 3, 3, 6}

Iteracja	Przed	Elementy do analizy	Wybrany min. element	Zamiana	Po
1	{4, 5, 3, 3, 6}	4, 5, 3, 3, 6	3	Tak	{3, 5, 4, 3, 6}
2	{3, 5, 4, 3, 6}	5, 4, 3, 6	3	Tak	{3, 3, 4, 5, 6}
3	{3, 3, 4, 5, 6}	4, 5, 6	4	Nie	{3, 3, 4, 5, 6}
4	{3, 3, 4, 5, 6}	5, 6	5	Nie	{3, 3, 4, 5, 6}
5	{3, 3, 4, 5, 6}	6	6	Nie	{3, 3, 4, 5, 6}

Sortowania przez wybieranie

Algorytm

1. Inicjalizacja:
 - zainicjalizuj zmienną rozmiar jako długość tablicy.
2. Iteracja zewnętrzna:
 - iteruj przez tablicę z indeksem i od 0 do rozmiar - 1.
3. Znajdowanie minimum:
 - znajdź indeks elementu o najmniejszej wartości (minIndeks).
4. Zamiana elementów:
 - zamień tablica[i] i tablica[minIndeks].
5. Zakończenie iteracji:
 - powtórz kroki 3-4 aż tablica zostanie posortowana.
6. Zwrócenie wyniku:
 - zwróć posortowaną tablicę.

Zadanie: Sortowania przez wybieranie - kod

Zadanie

Napisz kod JavaScript sortowania przez wybieranie

Przetestuj go dla zbioru liczb {4, 5, 3, 3, 6}

Zadanie: Sortowania przez wybieranie - kod

Rozwiązanie

```
function sortujPrzezWybieranie(tablica) {
  let rozmiar = tablica.length;

  // Iteracja zewnętrzna: przejście przez całą tablicę
  for (let i = 0; i < rozmiar - 1; i++) {
    // Znajdowanie minimum w niesortowanej części tablicy
    let minIndeks = i;
    for (let j = i + 1; j < rozmiar; j++) {
      if (tablica[j] < tablica[minIndeks]) {
        minIndeks = j;
      }
    }

    // Zamiana elementów
    if (minIndeks !== i) {
      [tablica[i], tablica[minIndeks]] = [tablica[minIndeks], tablica[i]];
    }
  }

  // Testowanie algorytmu
  const nieposortowanaTablica = [4, 5, 3, 3, 6];
  const posortowanaTablica = sortujPrzezWybieranie(nieposortowanaTablica);
  console.log("Posortowana tablica:", posortowanaTablica);

  return tablica;
}
```

Sortowanie przez wstawianie

Sortowanie przez wstawianie (Insertion Sort)

jest jednym z prostszych algorytmów sortujących.

Można go porównać do układania kart w rękę podczas gry w karty.

Algorytm zaczyna od uznania pierwszego elementu tablicy za posortowany. Następnie, iteruje przez niesortowaną część tablicy, traktując każdy element jako "klucz" do wstawienia w posortowanej sekcji. Dla każdego "klucza", algorytm przesuwa wszystkie większe elementy w posortowanej części o jedno miejsce w prawo. "Klucz" jest wówczas wstawiany na zwolnione miejsce, co gwarantuje, że posortowana sekcja pozostaje posortowana.

Klasa złożoność czasowej

$O(n^2)$

Zalety

prostota implementacji,
mało zasobochłonny

Wady

niewydajny dla dużych
zbiorów danych

Sortowanie przez wstawianie

Przykład

Mamy zbiór liczb {5, 4, 3, 8, 3, 6}

Iteracja	Przed	Klucz	Elementy do przesunięcia	Zamiana	po
1	{5, 4, 3, 8, 3, 6}	4	5	Tak	{4, 5, 3, 8, 3, 6}
2	{4, 5, 3, 8, 3, 6}	3	4, 5	Tak	{3, 4, 5, 8, 3, 6}
3	{3, 4, 5, 8, 3, 6}	8	-	Nie	{3, 4, 5, 8, 3, 6}
4	{3, 4, 5, 8, 3, 6}	3	8, 5, 4	Tak	{3, 3, 4, 5, 8, 6}
5	{3, 3, 4, 5, 8, 6}	6	8	Tak	{3, 3, 4, 5, 6, 8}

Sortowanie przez wstawianie

Algorytm

1. Inicjalizacja:

- uznaj pierwszy element tablicy za posortowany.

2. Iteracja zewnętrzna:

- rozpocznij pętlę zewnętrzną z indeksem i , który zaczyna się od 1 i kończy na ostatnim indeksie tablicy.

3. Wstawianie elementu:

- ustal wartość elementu na pozycji i jako "klucz".
- porównuj "klucz" z elementami w posortowanej części tablicy (od $i-1$ do 0), przesuwając je o jedną pozycję w prawo, dopóki są większe od "klucza".

4. Wstawienie klucza:

- wstaw "klucz" na odpowiednią pozycję w posortowanej części tablicy.

5. Kontynuacja iteracji:

- kontynuuj pętlę zewnętrzną aż dojdiesz do końca tablicy.

6. Zwrócenie wyniku:

- zwróć posortowaną tablicę.

Zadanie: Sortowanie przez wstawianie - kod

Zadanie

Napisz kod JavaScript sortowania przez wstawianie

Przetestuj go dla zbioru liczb {5, 4, 3, 8, 3, 6}

Zadanie: Sortowanie przez wstawianie - kod

Rozwiązanie

```
function sortujPrzezWstawianie(tablica) {
  let rozmiar = tablica.length;

  // Iteracja zewnętrzna: przejście przez całą tablicę
  for (let i = 1; i < rozmiar; i++) {
    let klucz = tablica[i];
    let j = i - 1;

    // Iteracja wewnętrzna: przesunięcie elementów większych od klucza
    while (j >= 0 && tablica[j] > klucz) {
      tablica[j + 1] = tablica[j];
      j = j - 1;
    }

    // Wstawienie klucza na odpowiednią pozycję
    tablica[j + 1] = klucz;
  }

  return tablica;
}

// Testowanie algorytmu
const nieposortowanaTablica = [5, 4, 3, 8, 3, 6];
const posortowanaTablica = sortujPrzezWstawianie(nieposortowanaTablica);
console.log("Posortowana tablica:", posortowanaTablica);
```