МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования
УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
имени первого Президента России Б.Н. Ельцина
ИНСТИТУТ РАДИОЭЛЕКТРОНИКИ
И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Информационные системы и технологии

ЗНАКОМСТВО С JAVASCRIPT

Направление подготовки: 09.04.02 «Прикладной анализ данных»

Лабораторная работа магистра Сергеева Егора

Екатеринбург, 2025

Постановка задач

Необходимо написать три алгоритма на чистом JS, а также оценить их по сложности времени/памяти в O-нотации:

1) Алгоритм сортировки массива (Bubble sort)
2) Алгоритм бинарного поиска в отсортированном массиве
3) Алгоритм проверки текста на правильность скобочной последовательности

Ход решения

Алгоритм пузырьковой сортировки массива
Сложность по времени O(n^2), сложность по памяти O(1)

bubble-sort.js

```
// Сложность O(n^2) по времени (вложенный цикл), O(1) по памяти

function bubbleSort(arr) {  Show usages  Gloccium
    const a = arr.slice();
    for (let i :number  = 0; i < a.length - 1; i++) {
        let swapped :boolean  = false;
        for (let j :number  = 0; j < a.length - 1 - i; j++) {
            if (a[j] > a[j + 1]) {
                [a[j], a[j + 1]] = [a[j + 1], a[j]];
                swapped = true;
            }
        }
        if (!swapped) break;
    }
    return a;
}

const arr :number[]  = [5, 3, 2, 4, 1];
console.log("Исходный массив:", arr);
console.log("Отсортированный массив:", bubbleSort(arr));
```

```
Исходный массив: [ 5, 3, 2, 4, 1 ]
Отсортированный массив: [ 1, 2, 3, 4, 5 ]

Process finished with exit code 0
```

Алгоритм бинарного поиска в отсортированном массиве
Сложность по времени O(logn), сложность по памяти O(1)

```
// Сложность по времени O(log(n)), сложность по памяти O(1)

// Возвращаем индекс найденного элемента, если не нашли, то -1
function binarySearch(arr, target) : any | number  {  Show usages  Gloccium
    let left : number  = 0;
    let right : number  = arr.length - 1;

    while (left <= right) {
        const mid : number  = left + Math.floor( x: (right - left) / 2);
        const midVal = arr[mid];

        if (midVal === target) return mid;
        if (midVal < target) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}

const arr : number[]  = [1, 3, 3, 4, 6, 8, 10];
console.log("Индекс искомого элемента:", binarySearch(arr,  target: 4));
console.log("Индекс элемента, которого нет в массиве:", binarySearch(arr,  target: 5));
```

```
Индекс искомого элемента: 3
Индекс элемента, которого нет в массиве: -1

Process finished with exit code 0
```

Алгоритм проверки текста на правильность скобочной последовательности
Сложность O(n) по времени и O(n) по памяти

```
// Сложность O(n) по времени и O(n) по памяти

// Если последовательность правильная, возвращаем true, иначе false
function isBalancedBrackets(text) :boolean {  Show usages  Gloccium
    const opens :Set<string> = new Set( values: ['(', '[', '{']);
    const pair :{...} = { ')': '(', ']': '[', '}': '{' };
    const stack :any[] = [];

    for (let i :number = 0; i < text.length; i++) {
        const ch = text[i];

        if (opens.has(ch)) {
            stack.push(ch);
        } else if (pair[ch]) {
            if (stack.length === 0) return false;
            const top = stack.pop();
            if (top !== pair[ch]) return false;
        }
    }
    return stack.length === 0;
}

console.log(isBalancedBrackets( text: "Текст (с [правильной {вложенностью}])"));
console.log(isBalancedBrackets( text: "([)]"));
console.log(isBalancedBrackets( text: "(()"));
console.log(isBalancedBrackets( text: "(()){[]}([])"));
console.log(isBalancedBrackets( text: "какой-то текст вообще без скобок"));
```

```
true
false
false
true
true

Process finished with exit code 0
```

Ссылка на репозиторий GitHub

https://github.com/Gloccium/JS-Lab-3

Вывод

Было реализовано три алгоритма: пузырьковая сортировка массива, бинарный поиск в отсортированном массиве, а также проверка правильности скобочной последовательности в тексте. Алгоритмы были оценены по времени и по памяти в терминах O-нотации.