

Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина

Институт радиоэлектроники и информационных технологий

Магистратура «Прикладной анализ данных»

Курс «Программирование на JS»

Лабораторная работа №3

«Знакомство с JavaScript»

Преподаватель курса:

Сайчик Е. Д.

Выполнил работу:

Бургарт Артем Андреевич

Екатеринбург

2025

Цель:

Получение базовых навыков программирования на языке JavaScript путем реализации и анализа фундаментальных алгоритмов и структур данных.

Освоение работы с массивами и строками, а также изучение основ анализа сложности алгоритмов.

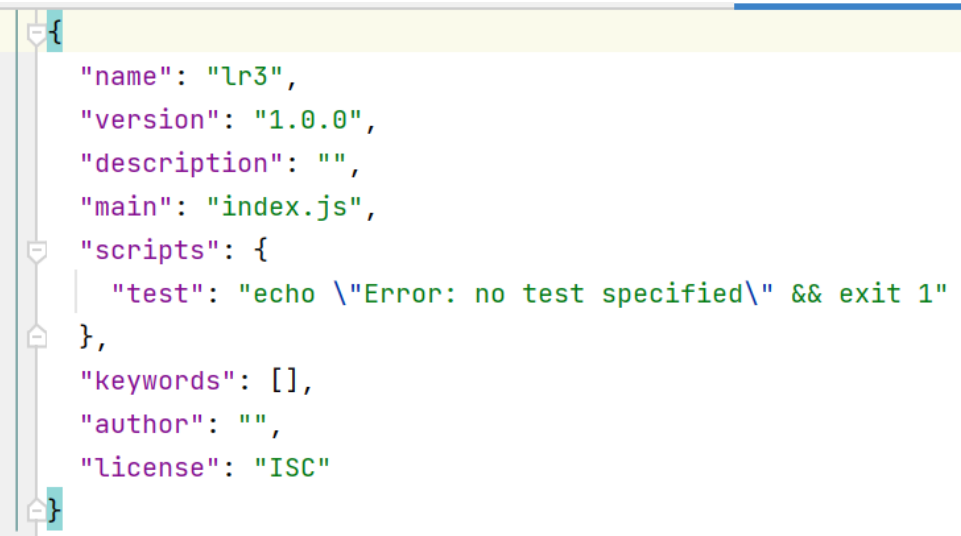
Постановка задачи в рамках задания:

Задание заключается в практической реализации на языке JavaScript нескольких фундаментальных алгоритмов. Необходимо разработать функцию для сортировки массива с последующим анализом её временной и пространственной сложности. Далее, для работы с отсортированными данными, следует реализовать алгоритм бинарного поиска. В завершение требуется написать функцию, проверяющую корректность расстановки различных типов скобок в строке.

Ход решения:

0. Инициализация проекта

Выполняем команду ``npm init -y``



```
{
  "name": "lr3",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

1. Сортировка подсчетом

Сортировка подсчетом работает только для целых чисел.

Оценка сложности:

- n - число элементов в массиве
- k - максимальное значение в массиве

Временная сложность:

- $O(n)$ - поиск максимального элемента (проходим 1 раз по исходному массиву)
- $O(k)$ - создание и заполнение массива нулями
- $O(n)$ - подсчет частот элементов (проходим 1 раз по исходному массиву)
- $O(n + k)$ - формирование отсортированного массива (внешний цикл `for` выполняется $k + 1$ раз, а суммарное количество операций внутри `while` равно n)

Итоговая временная сложность: $O(n) + O(k) + O(n) + O(n+k) = O(n+k)$

Пространственная сложность:

- $O(k)$ - дополнительный массив для подсчета размером $k + 1$
- $O(n)$ - результирующий массив для хранения отсортированных элементов

Итоговая пространственная сложность: $O(k) + O(n) = O(k + n)$

```
function countingSort(numbers) {  
  if (numbers.length <= 1)  
    return numbers;  
  
  let max = Math.max(...numbers);  
  let array = new Array(arrayLength: max + 1).fill(value: 0);  
  
  for (let num of numbers)  
    array[num]++;  
  
  let result = [];  
  for (let i = 0; i < array.length; i++) {  
    while (array[i] > 0) {  
      result.push(i);  
      array[i]--;  
    }  
  }  
  
  return result;  
}  
  
console.log(countingSort(numbers: [12, 33, 2, 87, 216, 7, 5, 367]))
```

2. Бинарный поиск

Работает только на отсортированных массивах

Оценка сложности:

- n - число элементов в массиве

Временная сложность:

- $O(\log(n))$ - на каждой итерации область поиска сокращается вдвое.
После k итераций - $n / 2^k$. Цикл завершается, когда размер области становится равен 1. $n / 2^k = 1 \Rightarrow k = \log_2(n)$

Итоговая временная сложность: $O(\log(n))$

Пространственная сложность:

- $O(1)$ - объем памяти константный и не растет с увеличением n , не создается дополнительных массивов или структур данных

Итоговая пространственная сложность: $O(1)$

```
function binarySearch(number, array) {  
  if (array.length === 0)  
    return -1;  
  
  let left = 0;  
  let right = array.length - 1;  
  
  while (left < right) {  
    let center = Math.floor((right + left) / 2);  
    if (number <= array[center])  
      right = center;  
    else  
      left = center + 1;  
  }  
  
  if (array[left] === number)  
    return left;  
  
  return -1;  
}  
  
const list = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144];  
console.log(binarySearch(number: 34, list));
```

3. Проверка правильности скобочной последовательности

Оценка сложности:

- n - длина входной строки

Временная сложность:

- $O(n)$ - проход по строке происходит всего 1 раз

Итоговая временная сложность: $O(n)$

Пространственная сложность:

- $O(n)$ - размер стека может быть равен размеру входной строки

Итоговая пространственная сложность: $O(n)$

```

function isValidBrackets(text) {
  let stack = [];
  let brackets = {
    '(': ')',
    '{': '}',
    '[': ']'
  };

  for (let char of text) {
    if (char in brackets)
      stack.push(char);

    else if (Object.values(brackets).includes(char)) {
      if (stack.length === 0)
        return false;

      let last = stack.pop();
      if (brackets[last] !== char)
        return false;
    }
  }

  return stack.length === 0;
}

console.log(isValidBrackets( text: "abc(def)ghi[jklm(nop[qrstuvw)xyz"]);
console.log(isValidBrackets( text: "a(bcd[efgh}ijklm(nopq)rst]uv]wx)yz");|

```


Вывод:

В ходе выполнения лабораторной работы были освоены базовые алгоритмы и структуры данных на JavaScript. Были успешно реализованы функции сортировки массива, бинарного поиска и проверки корректности скобочной последовательности. Это позволило на практике закрепить навыки работы с массивами и строками, применить структуру данных "стек", а также получить понимание основ анализа алгоритмической сложности.

Github:

<https://github.com/aburgart02/JS3>