Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

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

Отчет по лабораторной работе №3

по дисциплине «Программирование на JavaScript»

«Знакомство с JavaScript»

Преподаватель Сайчик Е.Д.

Студент гр. РИМ-240950 Зверев А. Д.

Екатеринбург

2025

Цель работы: применить знания о базовых конструкциях JavaScript для решения ряда учебных задач

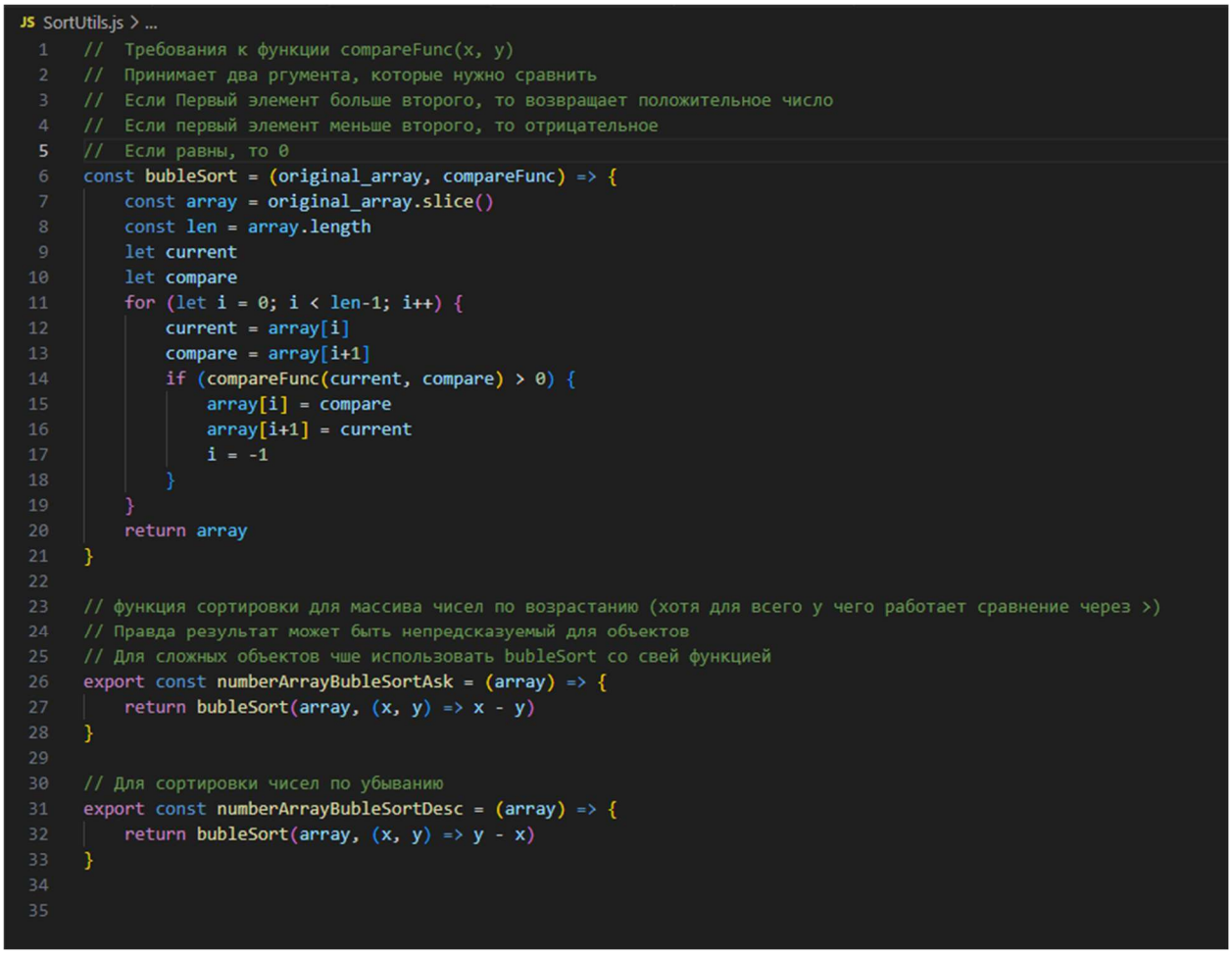
Задачи:

1. Создать функцию сортировки произвольного массива и оценить ее сложность по времени и памяти
2. Реализовать функцию бинарного поиска в массиве произвольных элементов
3. Решить задачу «Есть текст (строка) в ней встречаются разные скобки. Проверить, что кол-во открывающихся и закрывающихся кавычек повторяется в нужном порядке»

Задание 1

Формулировка: создать функцию сортировки произвольного массива и оценить ее сложность по времени и памяти

Решение:



```
JS SortUtils.js > ...
1   //  Требования к функции compareFunc(x, y)
2   //  Принимает два ргумента, которые нужно сравнить
3   //  Если Первый элемент больше второго, то возвращает положительное число
4   //  Если первый элемент меньше второго, то отрицательное
5   //  Если равны, то 0
6   const bubleSort = (original_array, compareFunc) => {
7       const array = original_array.slice()
8       const len = array.length
9       let current
10      let compare
11      for (let i = 0; i < len-1; i++) {
12          current = array[i]
13          compare = array[i+1]
14          if (compareFunc(current, compare) > 0) {
15              array[i] = compare
16              array[i+1] = current
17              i = -1
18          }
19      }
20      return array
21  }
22
23  // функция сортировки для массива чисел по возрастанию (хотя для всего у чего работает сравнение через >)
24  // Правда результат может быть непредсказуемый для объектов
25  // Для сложных объектов чше использовать bubleSort со свей функцией
26  export const numberArrayBubleSortAsk = (array) => {
27      return bubleSort(array, (x, y) => x - y)
28  }
29
30  // Для сортировки чисел по убыванию
31  export const numberArrayBubleSortDesc = (array) => {
32      return bubleSort(array, (x, y) => y - x)
33  }
34
35
```

Сильно заморачиваться с сортировкой не хотелось, поэтому была реализована сортировка пузырьком. Она имеет временную сложность n^2 и сложность по памяти n.

Первое обусловлено тем, что мы должны для каждого из n элементов массива обойти n-1 элементов (в худшем случае). Получаем n*(n – 1) = n^2 – n, что в нотации O(n) равно n^2.

Сложность по памяти всего n, поскольку мы работам в рамках одного массива из n элементов. (Да, в моей реализации создается копия массива, чтобы не изменять входной, но это все равно сложность 2*n, что превращается в n в рамках O(n))

Когда речь идет о сортировке объектов важно, чтобы мы знали, как их нужно сравнивать. Поскольку в условиях задачи не обговаривалось, массив каких объектов мы сортируем, то была сделана универсальная функция, которая

требует двух аргументов – непосредственно массива и функции сравнения, требования к которой написаны в комментариях к коду.

На основании базовой функции сортировки были сделаны две частные функции для сортировки массива чисел по возрастанию и по убыванию за счет механизма замыкания

Для ее проверки в файле index.js сравниваются результаты сортировки встроенной функции JS и моей на тестовом массиве чисел.

Задание 2

Формулировка: Реализовать функцию бинарного поиска в массиве произвольных элементов

Решение:

SeqrchUtils.js > binarySearch

```
//  Требования к функции compareFunc(x, y)
//  Принимает два ргумента, которые нужно сравнить
//  Если Первый элемент больше второго, то возвращает 1
//  Если первый элемент меньше второго, то -1
//  Если равны, то 0

const binarySearch = (targer, original_array, compareFunc) => {
    if (original_array.length === 0) {
        console.log("Массив пустой")
        return -1
    }
    if (typeof targer != typeof original_array[0]) {
        console.log("Тип искомого элемента не совпадает с типом элементов массива")
        return -1
    }
    if (original_array.length > 1 && compareFunc(original_array[0], original_array[1]) > 0) {
        console.log("Массив не отсортирован по возрастанию элементов в соответствии с функцией сравнения")
        return -1
    }
    const array = original_array.slice()
    let start = 0
    let end = array.length - 1
    let center
    let compareRes
    while (start <= end) {
        center = Math.floor((start + end)/2)
        compareRes = compareFunc(array[center], targer)
        if (compareRes == 0) {
            return center
        } else if (compareRes > 0) {
            end = center - 1
        } else {
            start = center + 1
        }
    }
    return -1
}
```

Для корректной работы бинарного поиска входной массив должен быть отсортирован в соответствии с той же функцией сравнения, которой производилась сортировка, поэтому добавлена проверка, которая в случае невыполнения этого условия выдает ошибку в консоль и возвращает -1. Также добавлены проверки на пустой массив и совпадение типов объектов массива и искомого объекта, чтобы не выполнять поиск без необходимости.

Аналогично функции сортировки функция бинарного поиска, помимо искомого элемента и массива, требует функция сравнения для корректной работы.

В остальном это классическая функция бинарного поиска, которая сравнивает искомый элемент с серединой массива и в случае, если элемент в центре больше искомого, то берется левая половина массива, если меньше, то

правая, и действия повторяются до тех пор, пока не будет найден элемент, либо индексы начала и конца не встретятся. В случае успеха будет возвращен индекс по которому найден элемент, в случае неудачи -1.

Также реализован частный случай бинарного поиска для массива чисел.

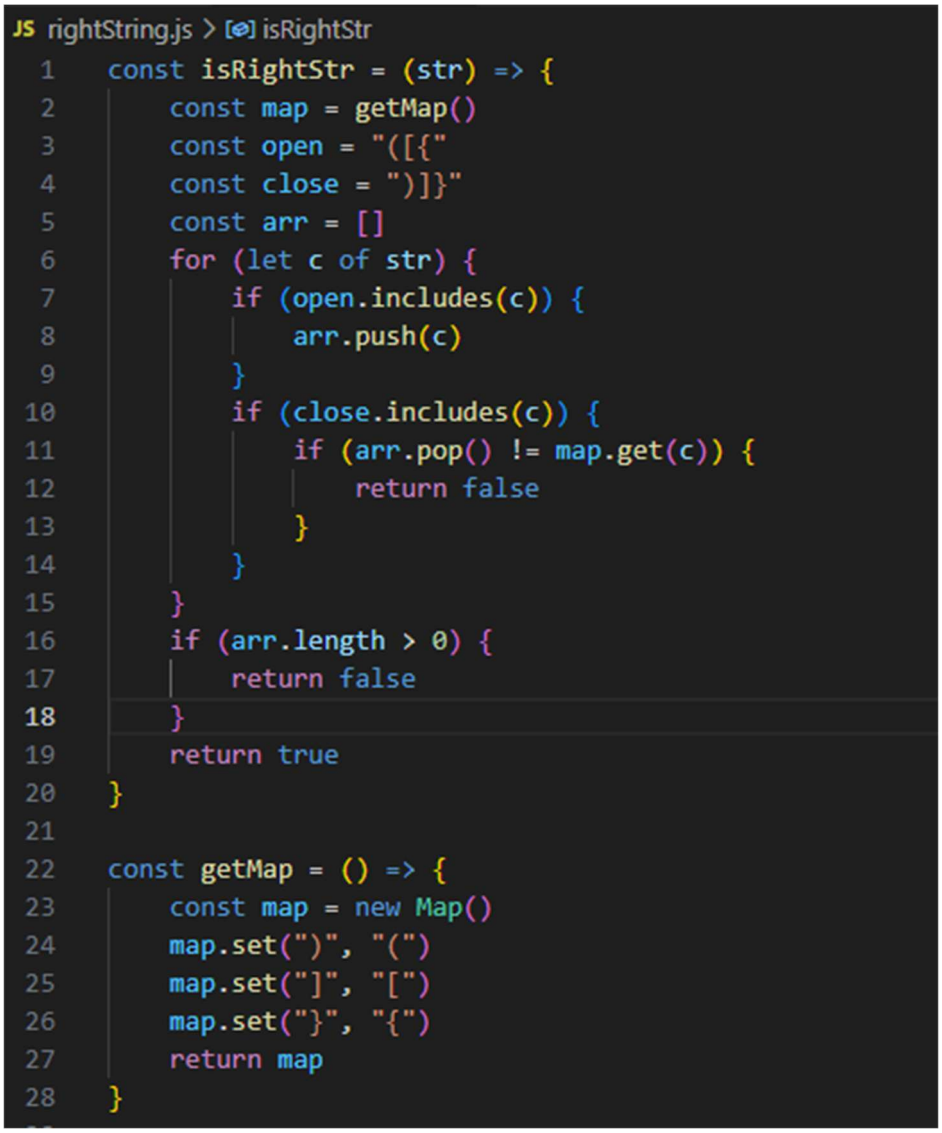
```
38
39  export const numberBinarySeaarch = (target, array) => {
40      return binarySearch(target, array, (x ,y) => x - y)
41  }
42
43
```

С ней в файле index.js есть ряд тестов для различных ситуаций.

Задание 3

Формулировка: решить задачу «Есть текст (строка) в ней встречаются разные скобки. Проверить, что кол-во открывающихся и закрывающихся кавычек повторяется в нужном порядке»

Решение:



Для наглядности вынес интересующие нас символы в отдельную функцию (при желании можно добавить и все будет работать). Назначение данной мапы в том, чтобы для любого закрывающего символа получать его открывающую версию (пытался не дублировать в функции isRightStr закрывающие и открывающие символы, но метод values() у Map возвращает MapIterator с которым очень неудобно работать)

Идея простая – каждый открывающийся символ мы помещаем в конец нашего контрольного списка arr. Когда находим в строке закрывающий символ, то берем соответствующий ему открывающий символ и сравниваем с последним символом в массиве arr. Если совпадают, то правила верной

строки не нарушаются и можно идти дальше, но если соответствия нет, то возвращаем false.

Когда вся строка пройдена, то проверяем размер контрольного массива arr. Если он пустой, то всем открывающим символам были найдены закрывающие в нужном порядке и строка является правильной. Если же нет, то открывающих символов было больше и строка неверная.

Также в файле rightString.js есть ряд тестов, охватывающих различные условия.

Ссылка на GitHub

https://github.com/UnderAlex59/JS-Lab-3

Выводы

В ходе лабораторной работы применили знания о базовых типах данных и управляющих конструкциях JavaScript для решения ряда учебных заданий. Попрактиковались в оценке сложности алгоритмов по памяти и времени в нотации O(n)