

УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
имени первого Президента России Б. Н. Ельцина

Институт радиоэлектроники и информационных технологий-РТФ

ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине «Программирование на JavaScript»
на тему «Знакомство с JavaScript»

Студент группы РИМ-240950
Плотников Егор Дмитриевич

Преподаватель
Сайчик Евгений Дмитриевич

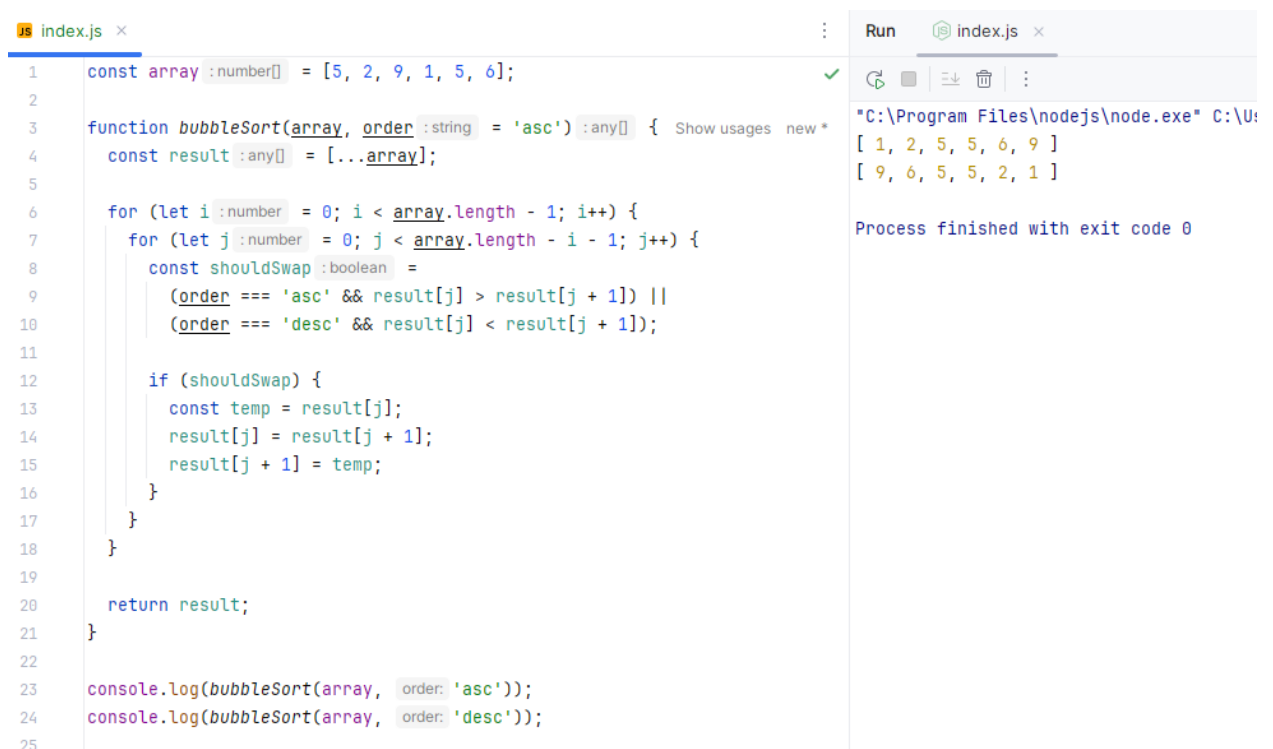
Екатеринбург 2025

ОПИСАНИЕ ЗАДАЧИ

0. Создать проект
1. Массив (придумываем сами). Написать алгоритм сортировки массива. Оценить его с точки зрения сложности (времени выполнения и памяти)
2. Реализовать функцию бинарного поиска в массиве
3. Строка, где встречаются круглые, фигурные и квадратные скобки. Функция проверки корректности постановки открывающихся и закрывающихся скобок

ХОД ВЫПОЛНЕНИЯ

0. Создал проект
1. Пузырьковая сортировка



```
index.js x
1 const array : number[] = [5, 2, 9, 1, 5, 6];
2
3 function bubbleSort(array, order : string = 'asc') : any[] { Show usages new *
4   const result : any[] = [...array];
5
6   for (let i : number = 0; i < array.length - 1; i++) {
7     for (let j : number = 0; j < array.length - i - 1; j++) {
8       const shouldSwap : boolean =
9         (order === 'asc' && result[j] > result[j + 1]) ||
10        (order === 'desc' && result[j] < result[j + 1]);
11
12       if (shouldSwap) {
13         const temp = result[j];
14         result[j] = result[j + 1];
15         result[j + 1] = temp;
16       }
17     }
18   }
19
20   return result;
21 }
22
23 console.log(bubbleSort(array, order: 'asc'));
24 console.log(bubbleSort(array, order: 'desc'));
25
```

Run index.js x

"C:\Program Files\nodejs\node.exe" C:\U:
[1, 2, 5, 5, 6, 9]
[9, 6, 5, 5, 2, 1]

Process finished with exit code 0

Пузырьковая сортировка так называется, потому что значения «всплывают» наверх. Сложность алгоритма по времени выполнения могу оценить в $O(n^2)$, потому что здесь два вложенных цикла. Получается, мы проходим по массиву n^2 раз. А сложность по памяти оцениваю в $O(n)$, так как мы создаём копию

массива и работаем с ней. Если нам не важно сохранять оригинал массива, то можно не создавать клона — тогда сложность будет $O(1)$.

2. Бинарный поиск



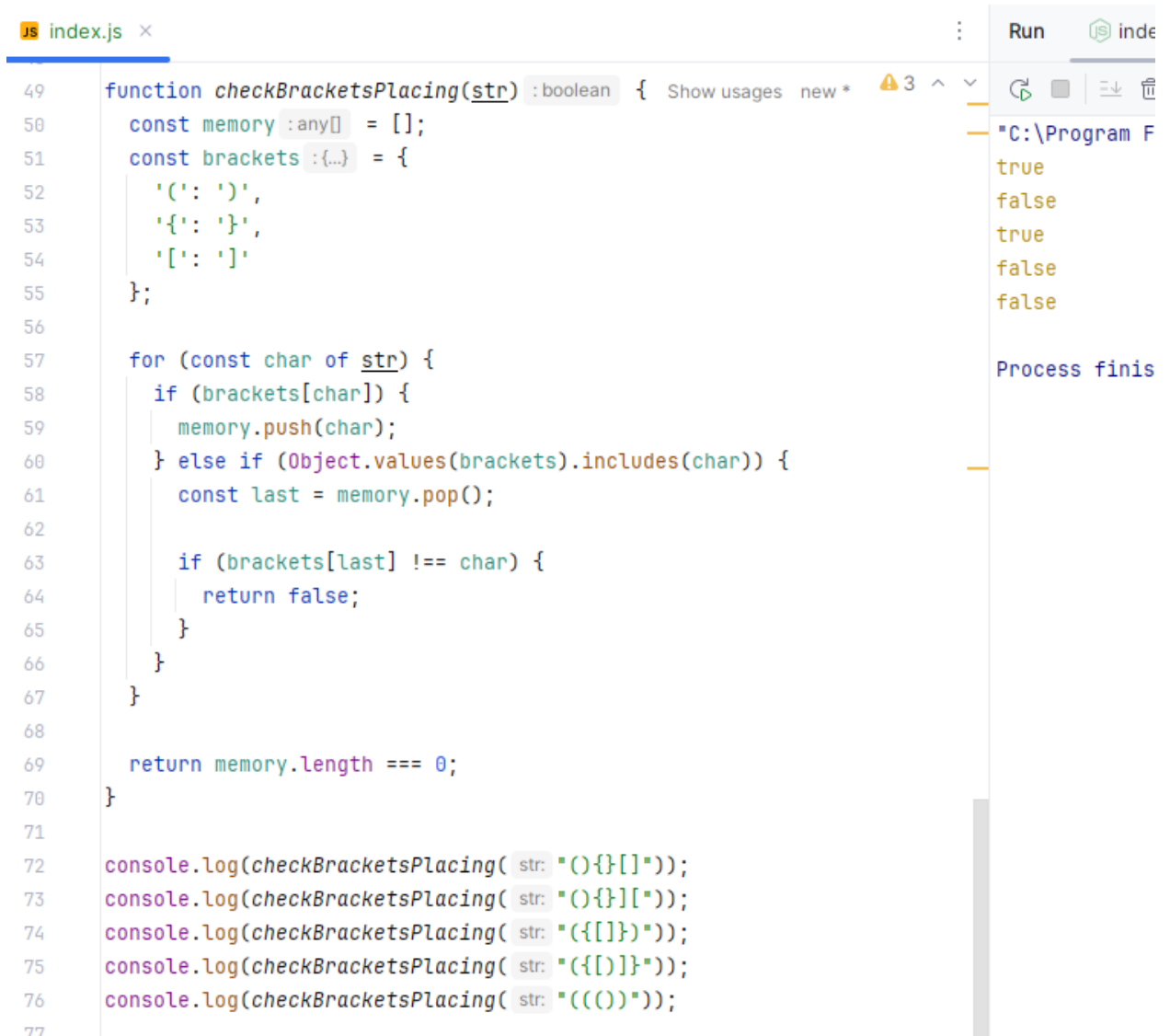
```
25
26 function binarySearch(array, target) : number { Show usages new *
27   let left : number = 0;
28   let right : number = array.length - 1;
29
30   while (left <= right) {
31     const mid : number = Math.floor((left + right) / 2);
32
33     if (array[mid] === target) {
34       return mid;
35     } else if (array[mid] < target) {
36       left = mid + 1;
37     } else {
38       right = mid - 1;
39     }
40   }
41
42   return -1;
43 }
44
45 const sortedArray : any[] = bubbleSort(array, { order: 'asc' });
46 console.log(sortedArray);
47 console.log(binarySearch(sortedArray, { target: 5 }));
48
```

Run index.js

```
"C:\Program Files\nodejs\
[ 1, 2, 5, 5, 6, 9 ]
2
Process finished with exit
code 0
```

Алгоритм клёвый, мне понравился. Так как бинарный поиск работает только с отсортированными массивами, мы можем начать поиск со среднего элемента, если целевое число меньше нашего, то делаем то же самое с левой частью массива, иначе — с правой

3. Проверка корректности постановки скобок



```
49 function checkBracketsPlacing(str) : boolean { Show usages new * 3 ^ v
50   const memory : any[] = [];
51   const brackets : {...} = {
52     '(': ')',
53     '{': '}',
54     '[': ']'
55   };
56
57   for (const char of str) {
58     if (brackets[char]) {
59       memory.push(char);
60     } else if (Object.values(brackets).includes(char)) {
61       const last = memory.pop();
62
63       if (brackets[last] !== char) {
64         return false;
65       }
66     }
67   }
68
69   return memory.length === 0;
70 }
71
72 console.log(checkBracketsPlacing( str: "(){}[]"));
73 console.log(checkBracketsPlacing( str: "(){}[("));
74 console.log(checkBracketsPlacing( str: "{[]}{}"));
75 console.log(checkBracketsPlacing( str: "{([)]}"));
76 console.log(checkBracketsPlacing( str: "((()))"));
77
```

Run Console

"C:\Program F

true

false

true

false

false

Process finis

Здесь использовал такой алгоритм:

- на вход функции приходит строка
- объявляем «память», куда мы будем класть скобки
- словарь со скобками, где ключ — это открывающаяся, а значение — закрывающаяся
- проходим по каждому символу строки
- проверяем, что символ — это открывающаяся скобка, если есть значение в словаре по текущему символу
- если есть, то запоминаем его
- если нет, проверяем на закрывающую скобку
- если это она, то берём последний элемент из памяти

- возвращаем false, если в памяти было пусто, или если в памяти была закрывающаяся скобка
- когда прошли по всей строке, проверяем, что память пустая

ССЫЛКА НА РЕПОЗИТОРИЙ

<https://github.com/pltnkve/JS-Lab-3>

ВЫВОД

Была выполнена третья лабораторная работа по предмету «Программирование на JavaScript». Поработал с сортировкой массивов, бинарным поиском и проверкой корректности постановки скобок в строке.