МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе № 2

по дисциплине «Алгоритмы и структуры данных»

ТЕМА: АЛГОРИТМЫ СОРТИРОВКИ И ПОИСКА. Вариант: 1

Студент гр. 0302 _____ Савенко Н.С

Преподаватель _____ Тутуева А.В.

Санкт-Петербург

2021

Постановка задачи

Задачей является реализация алгоритмов сортировки и поиска.

Описание реализуемых алгоритмов

Двоичный поиск.

На каждом шаге осуществляется поиск середины отрезка по формуле

$$mid = (left + right)/2$$

Если искомый элемент равен элементу с индексом mid, поиск завершается.

В случае если искомый элемент меньше элемента с индексом mid, на место mid перемещается правая граница рассматриваемого отрезка, в противном случае — левая граница.

Quick Sort

В массиве выбирается некоторый элемент, называемый разрешающим. Затем он помещается в то место массива, где ему полагается быть после упорядочивания всех элементов. В процессе отыскания подходящего места для разрешающего элемента производятся перестановки элементов так, что слева от них находятся элементы, меньшие разрешающего, и справа — большие (предполагается, что массив сортируется по возрастанию).

InsertionSort

На каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем.

BogoSort

Перемешиваем массив, пока не получим отсортированный

CountingSort

Подсчитываем сколько раз в массиве встречается каждое значение и заполняем массив подсчитанными элементами в соответствующих количествах.

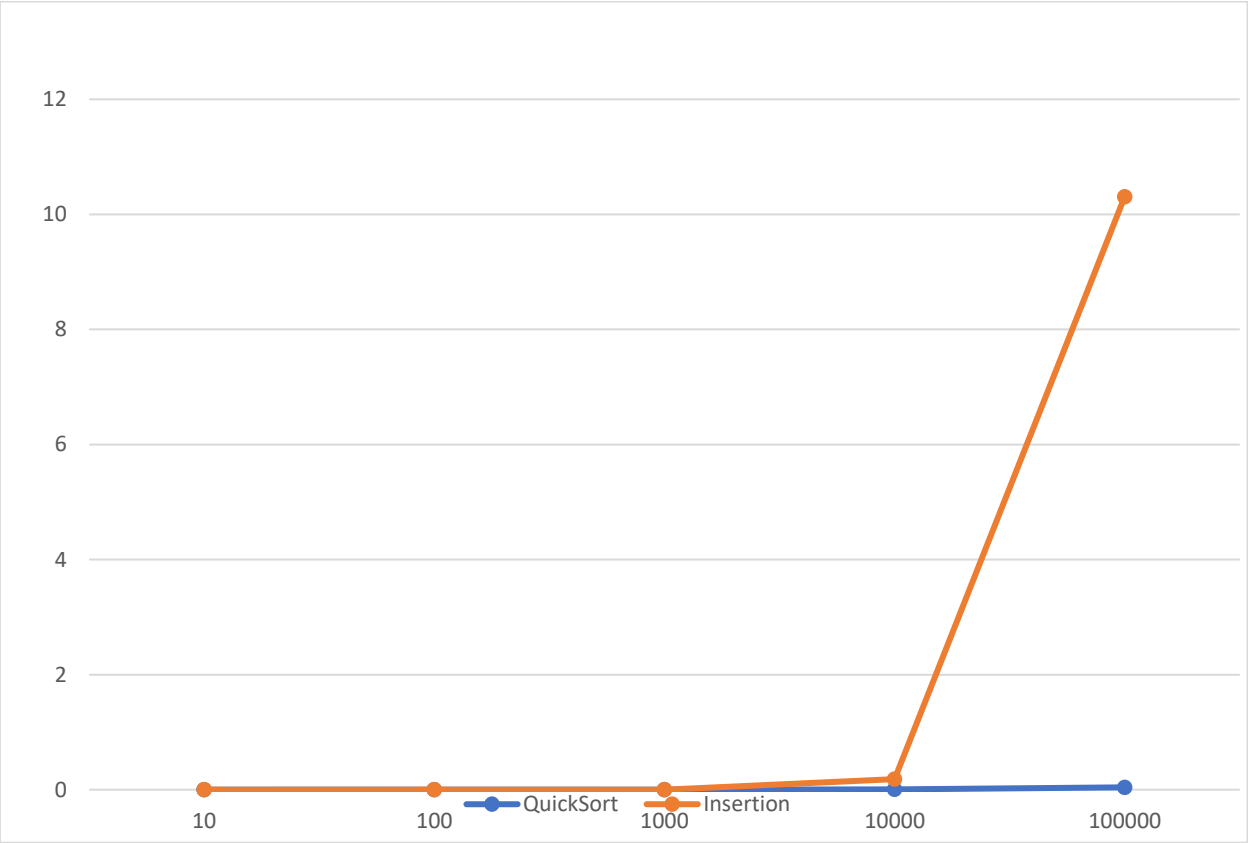
Оценка временной сложности

```
Двоичный поиск
      O(log2(n))

Quick Sort
      O(n log n)

InsertionSort
      O (n)


BogoSort
      O((n+1)!)
```

```
CountingSort
      O(n)
```

Сравнение временной сложности алгоритмов

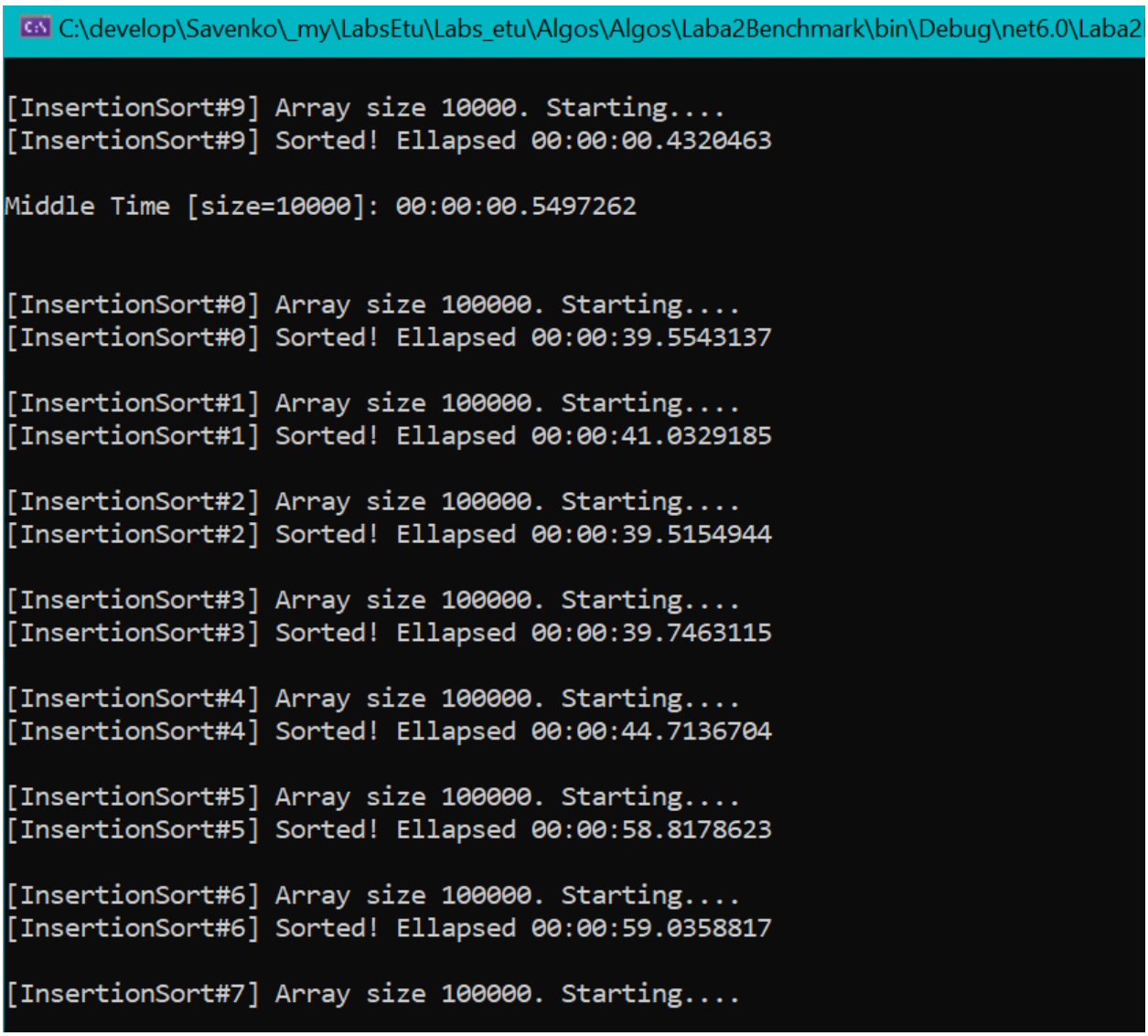
C	QuickSort T	Insertion
10	0,0004454	0,0000794
100	0,0000213	0,0000167
1000	0,0002276	0,0020203
10000	0,0027227	0,1856237
100000	0,0405926	10,3072052



Описание Unit тестов

Во всех тестах формируют массив случайных чисел, делаю его копию. Затем применяю на нем требуемый алгоритм. Потом применяю аналогичный гарантированно верный алгоритм из CLR. Сравниваю результаты.

Примеры работы



Листинг

```
public static class Sort
    {
        static void Swap<T>(ref T x, ref T y)
        {
            T t = x;
            x = y;
            y = t;
        }

        static int Partition<T>(T[] array, int minIndex, int maxIndex) where T :
IComparable<T>
        {
            var pivot = minIndex - 1;
            for (var i = minIndex; i < maxIndex; i++)
            {
                if (array[i].CompareTo(array[maxIndex]) < 0)
                {
                    pivot++;
                    Swap(ref array[pivot], ref array[i]);
                }
```

```
            }

            pivot++;
            Swap(ref array[pivot], ref array[maxIndex]);
            return pivot;
        }

        public static T[] QuickSort<T>(this T[] array, int minIndex, int maxIndex)
    where T : IComparable<T>
        {
            if (minIndex >= maxIndex)
            {
                return array;
            }

            var pivot = Partition(array, minIndex, maxIndex);
            array.QuickSort(minIndex, pivot - 1);
            array.QuickSort(pivot + 1, maxIndex);

            return array;
        }

        public static T[] InsertionSort<T>(this T[] array) where T : IComparable<T>
        {
            for (int i = 0; i < array.Length - 1; i++)
            {
                for (int j = i + 1; j > 0; j--)
                {
                    if (array[j - 1].CompareTo(array[j]) > 0)
                    {
                        T temp = array[j - 1];
                        array[j - 1] = array[j];
                        array[j] = temp;
                    }
                }
            }
            return array;
        }

        public static bool IsSorted<T>(this T[] array) where T : IComparable<T>
        {
            if (array.Length < 2)
                return true;

            for (int i = 1; i < array.Length; i++)
            {
                if (array[i].CompareTo(array[i - 1]) < 0)
                {
                    return false;
                }
            }
            return true;
        }

        private static void Shuffle<T>(this T[] array)
        {
            Random rand = new Random();
            for (int i = 0; i < array.Length; i++)
            {
                int swapIndex = rand.Next(array.Length);
                Swap(ref array[swapIndex], ref array[i]);
            }
        }

        public static T[] BogoSort<T>(this T[] array) where T : IComparable<T>
```

```
        {
            while (!array.IsSorted())
            {
                array.Shuffle();
            }
            return array;
        }

        public static void CountingSort(this char[] array)
        {
            int n = array.Length;

            char[] output = new char[n];
            int[] count = new int[256];

            for (int i = 0; i < 256; ++i)
                count[i] = 0;

            for (int i = 0; i < n; ++i)
                ++count[array[i]];

            for (int i = 1; i <= 255; ++i)
                count[i] += count[i - 1];

            for (int i = n - 1; i >= 0; i--)
            {
                output[count[array[i]] - 1] = array[i];
                --count[array[i]];
            }

            for (int i = 0; i < n; ++i)
                array[i] = output[i];
        }
}

public static class Search
    {
        // Extension Method
        // For details read CLR via C# by Jeffrey Richter
        public static int BinarySearch<T>(this T[] array, T value) where T :
IComparable<T>
        {
            int min = 0;
            int max = array.Length - 1;
            while (min <= max)
            {
                int mid = (min + max) / 2;
                if (value.CompareTo(array[mid]) == 0)
                {
                    return mid;
                }
                else if (value.CompareTo(array[mid]) < 0)
                {
                    max = mid - 1;
                }
                else
                {
                    min = mid + 1;
                }
            }
            return -1;
        }
    }
```