Университет информационных технологий, механики и оптики
Факультет компьютерных технологий и управления
Кафедра информатики и прикладной математики

ЛАБОРАТОРНАЯ РАБОТА №3
«СОРТИРОВКИ ПРОСТОЙ КУЧЕЙ И СЛИЯНИЕМ»

Выполнил:
студент гр. Р3118
Петкевич К. В.

Принял:
к.т.н старший
преподаватель
Симоненко З. Г.

г. Санкт-Петербург
2016 год

Цель работы

Для выполнения лабораторной работы необходимо сгенерировать тестовые файлы (используя генераторы случайных чисел), содержащие целые числа, в количестве от 2^6 до 2^{20} (можно и больше), при этом количество элементов в следующем файле в два раза больше чем в предыдущем, реализовать алгоритмы используя один из следующих языков программирования: C++, C#, C, Python, для каждого тестового файла из набора выполнить сортировку данных, вычислить среднее время сортировки по одному файлу, построить график зависимости времени сортировки от количества элементов в файле, выполнить сравнение алгоритмов

Текст генератора исходных данных

```
static public TimeSpan FileCreator(int n, string path)
{
        Random rnd = new Random((int)DateTime.Now.Ticks);
        Stopwatch timer = new Stopwatch();
        TimeSpan time;
        string s = path + "/TestFile";
        int i = 0, j = 0;
        timer = Stopwatch.StartNew();
        for (i = 0; i < n; i++)
        {
                string str = @s + i + ".txt";
                StreamWriter stream = File.AppendText(str);
                for (j = 0; j < (Math.Pow(2, 6 + i)); j++)
                {
                        string line = Convert.ToString(rnd.Next(0, Convert.ToInt32(Math.Pow(2, 6 + i))));
                        stream.WriteLine(line);
                }
                stream.Close();
        }
        Console.WriteLine("\nGenerated!\n");
        timer.Stop();
        time = timer.Elapsed;
        return (time);
}
```

Коды сортировок

1. Сортировка кучей

```
public static void heapify(int[] array, int parent, int end)
{
        while (2 * parent + 1 < end)
        {
                int child1 = 2 * parent + 1;
                int child2 = 2 * parent + 2;
                if (child2 < end && array[child2] >= array[child1])
                {
                        child1 = child2;
                }
                if (array[parent] < array[child1])
                {
                        swap(ref array[parent], ref array[child1]);
                        parent = child1;
                }
                else break;
        }
}

public static void heap_make(int[] array, int count)
{
```

```
for (int i = count - 1; i >= 0; i--)
        {
                heapify(array, i, count);
        }
}

public static void heap_sort(int[] array, int count)
{
        heap_make(array, count);
        while (count > 0)
        {
                swap(ref array[0], ref array[count - 1]);
                count--;
                heapify(array, 0, count);
        }
}
```

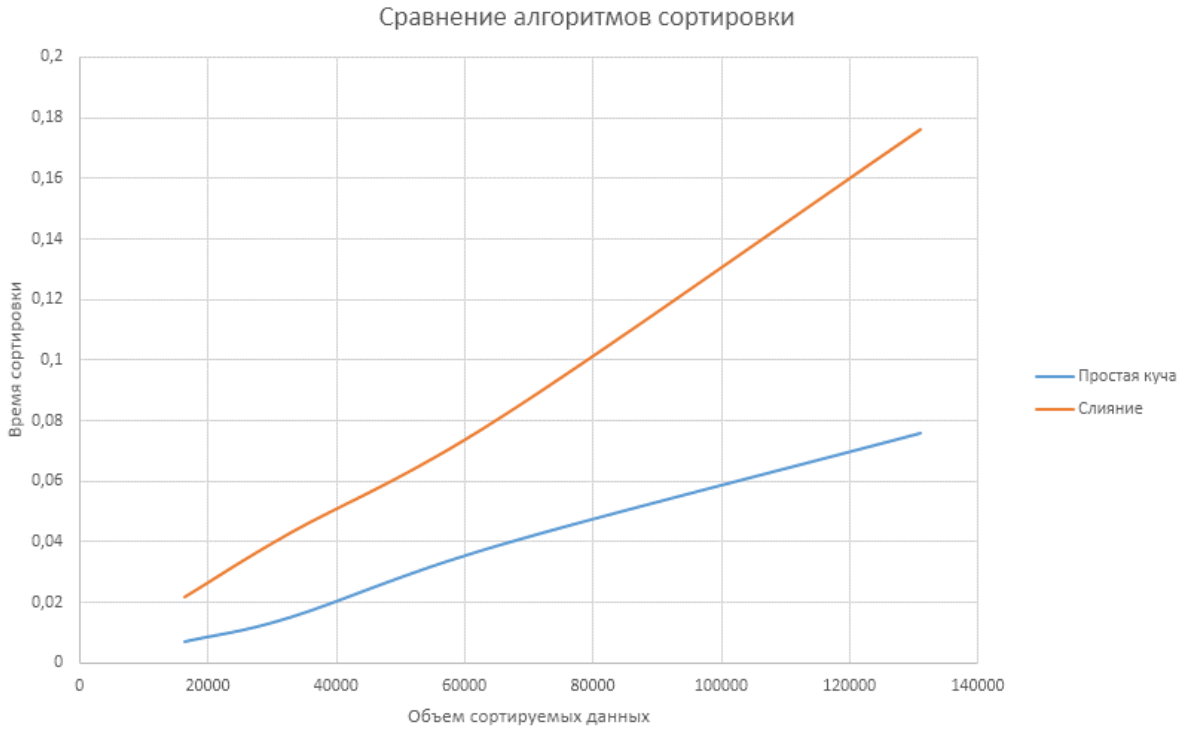
2. **Сортировка слиянием**

```
static public int[] Merge_Sort(int[] array)
{
        if (array.Length == 1)
                return array;
        int mid_point = array.Length / 2;
        return Merge(Merge_Sort(array.Take(mid_point).ToArray()), Merge_Sort(array.Skip(mid_point).ToArray()));
}

static public int[] Merge(int[] array1, int[] array2)
{
        int a = 0, b = 0;
        int[] merged = new int[array1.Length + array2.Length];
        for (int i = 0; i < array1.Length + array2.Length; i++)
        {
                if (b < array2.Length && a < array1.Length)
                {
                        if (array1[a] > array2[b])
                                merged[i] = array2[b++];
                        else
                                merged[i] = array1[a++];
                }
                else
                {
                        if (b < array2.Length)
                                merged[i] = array2[b++];
                        else
                                merged[i] = array1[a++];
                }
        }
        return merged;
}
```

Результаты

	Время сортировки, с	
Кол-во эл-в	Простая куча	Слияние
16384	0,005	0,021
32768	0,012	0,042
65536	0,040	0,079
131072	0,073	0,177



Вывод

Среди этих двух алгоритмов сортировки самым эффективным по времени оказался алгоритм сортировки кучей, но у этого алгоритма есть недостаток: на почти отсортированных данных работает столь же долго, как и на хаотических данных.