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

ЛАБОРАТОРНАЯ РАБОТА №4

«Сортировки левосторонней кучей и многопутевым 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. **Левосторонняя куча**

```
static Node leftistHeap;

public static void Sort(T [] items)
{
        Node node;
        for (int i = 0; i < items.Length; i++)
        {
                node = new Node(items[i]);
                AddNode(ref leftistHeap, ref node);
        }
        for (int i = 0; i < items.Length; i++)
        {
                items[items.Length - 1 - i] = ExtractMax(ref leftistHeap);
        }
}
```

```
class Node
{
	public T key { get; private set; }
	public int npl = 0;
	public Node parent;
	public Node lchild;
	public Node rchild;
	public Node(T key)
	{
		this.key = key;
	}
}

static T ExtractMax(ref Node node)
{
	T maxkey = node.key;
	if (node.rchild != null)
	{
		node.lchild.parent = null;
	}
	if (node.rchild != null)
	{
		node.rchild.parent = null;
	}
	node = Merge(ref node.lchild, ref node.rchild);
	return maxkey;
}

static Node Merge(ref Node node1, ref Node node2)
{
	if (node1 == null)
	{
		return node2;
	}

	if (node2 == null)
	{
		return node1;
	}

	T H1 = node1.key;
	T H2 = node2.key;

	if (H1.CompareTo(H2) < 0)
	{
		Swap(ref node1, ref node2);
	}

	node1.rchild = Merge(ref node1.rchild, ref node2);
```

```
            node1.rchild.parent = node1;

            if (Npl(node1.rchild) > Npl(node1.lchild))
            {
                  Swap(ref node1.rchild, ref node1.lchild);
            }
            node1.npl = Npl(node1);
            return node1;
      }

      static void Swap(ref Node node1, ref Node node2)
      {
            Node temp = node1;
            node1 = node2;
            node2 = temp;
      }

      static void AddNode(ref Node leftistHeap, ref Node node)
      {
            leftistHeap = Merge(ref leftistHeap, ref node);
      }

      static int Npl(Node node)
      {
            if (node == null)
            {
                  return -1;
            }
            if (node.rchild == null)
            {
                  return 0;
            }
            return node.rchild.npl + 1;
      }
```

2. **Многопутевое слияние**

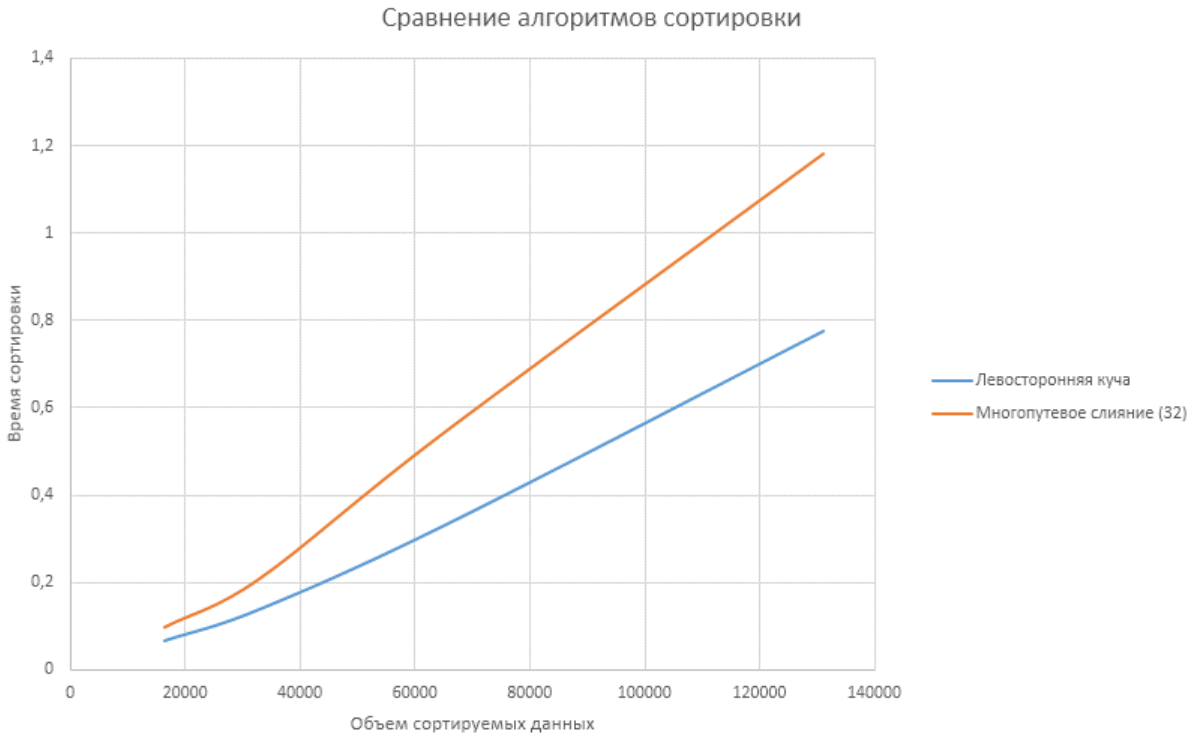
```
      public void Sort(T [] items)
      {
            this.items = items;
            Sort(0, items.Length - 1);
      }

      void Sort (int start, int end)
      {
            if (start < end)
            {
                  int length = (int)Math.Ceiling((double)(end - start + 1) / Ways);
```

```
for (int i = 0; i < Ways - 1; i++)
                {
                        int l, r;
                        l = start + length * i;
                        r = start + length * i + length - 1;
                        Sort(l, r);
                }
                Sort(start + length * (Ways - 1), end);

                Merge(start, end, length);
        }
}

void Merge(int start, int end, int length)
{
        T[] temp = new T[end - start + 1];
        // Creating lists for merging
        List<List<T>> merging = new List<List<T>>(Ways);
        List<T> tmp;
        int l, r;
        int j;

        for (int i = 0; i < Ways - 1; i++)
        {
                l = start + length * i;
                r = start + length * i + length - 1;
                tmp = new List<T>(length);
                for (j = l; j < r + 1; j++)
                        tmp.Add(items[j]);
                merging.Add(tmp);
        }
        if ((r = start + length * (Ways - 1)) <= end)
        {
                tmp = new List<T>(end - r + 1);
                for (j = r; j < end + 1; j++)
                        tmp.Add(items[j]);
                merging.Add(tmp);
        }

        j = 0;
        while (merging.Count != 0)
        {
                int min = 0;
                for (int i = 1; i < merging.Count; i++)
                {
                        if (merging[i][0].CompareTo(merging[min][0]) < 0)
                        {
                                min = i;
                        }
```

```
                }
                temp[j] = merging[min][0];
                merging[min].RemoveAt(0);
                if (merging[min].Count == 0)
                        merging.RemoveAt(min);
                j++;
        }

        j = 0;
        for (int i = start; i < end + 1; i++)
        {
                items[i] = temp[j];
                j++;
        }
}
```

Результаты

	Время сортировки, с	
Кол-во эл-в	Левосторонняя куча	Многопутевое слияние (32)
16384	0,0631	0,0959
32768	0,1281	0,2054
65536	0,3323	0,5317
131072	0,7594	1,1792



Вывод

Среди этих двух алгоритмов сортировки самым эффективным по времени оказался алгоритм сортировки левосторонней кучей.