ИТМО Кафедра Информатики и прикладной математики

Отчет по лабораторной работе №4 «Сортировки за O(nlogn)- сравнение сортировки левосторонней кучей и многопутевым слиянием»

Выполнил: студент группы Р3117
Плюхин Дмитрий
Проверил: Симоненко З. Г.

2016 год

1. Задание

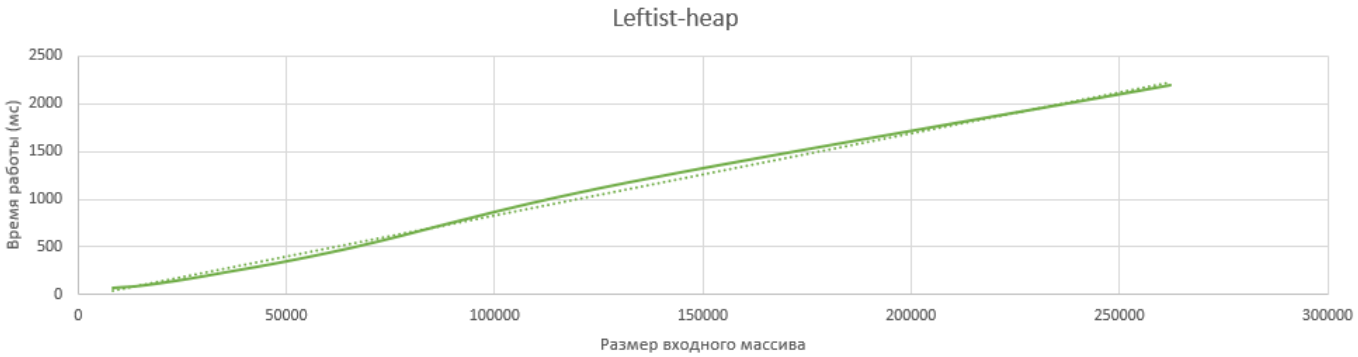
Реализовать на каком-либо языке программирования и сравнить между собой алгоритм сортировки левосторонней кучей и алгоритм сортировки многопутевым слиянием.

2. Выполнение

Для выполнения работы был выбран язык программирования C# по причине того, что алгоритмы сортировок, реализуемых в работе, используют указатели, применение которых затруднительно в некоторых других языках программирования.

После реализации алгоритмов на языке программирования был проведен их запуск на различных исходных данных, в частности, на массивах разной длины. Была выполнена сортировка массивов с использованием двух алгоритмов (время сортировки усреднено для каждого массива), построены графики, отражающие зависимость времени работы каждого алгоритма от количества элементов в сортируемом массиве. Для обеих сортировок представлен только код, относящийся непосредственно к сортировке. Так, при сортировке многопутевым слиянием используется структура данных «очередь с приоритетами», код которой не показан.

3. Результаты



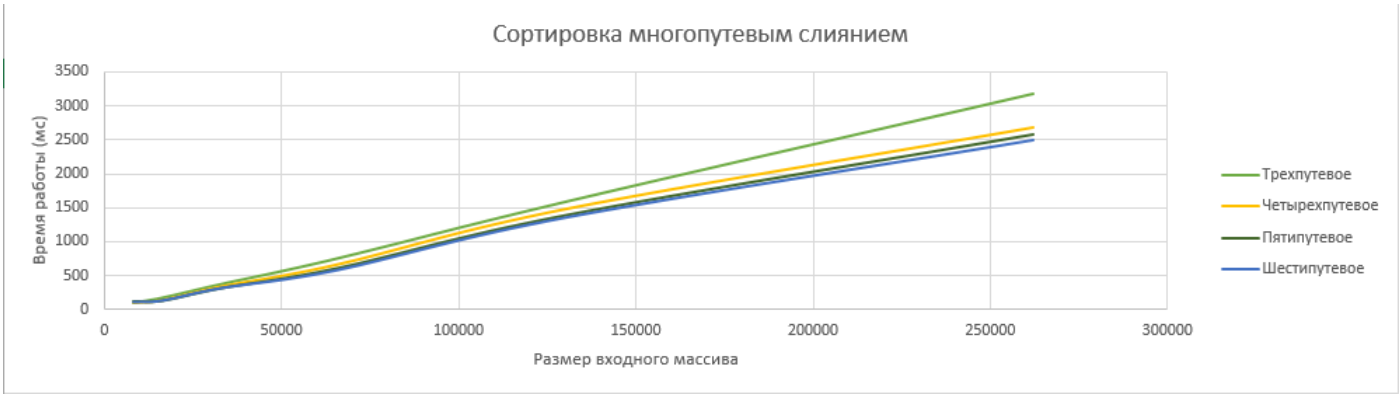

```
static Node Merge(ref Node node1, ref Node node2)
{
    if (node1 == null)
    {
        return node2;
    }
    if (node2 == null)
    {
        return node1;
    }
    int k1 = node1.key;
    int k2 = node2.key;
    if (k1 < k2)
    {
        Swap(ref node1, ref node2);
    }
    node1.rightchild = Merge(ref node1.rightchild, ref node2);
    node1.rightchild.parent = node1;
    if (Npl(node1.rightchild) > Npl(node1.leftchild))
    {
        Swap(ref node1.rightchild, ref node1.leftchild);
    }
    node1.npl = Npl(node1);
    return node1;
}

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

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

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

static int ExtractMaxKey(ref Node node)
{
    int maxkey = node.key;
    if (node.rightchild != null)
    {
        node.leftchild.parent = null;
    }
    if (node.rightchild != null)
    {
        node.rightchild.parent = null;
    }
    node = Merge(ref node.leftchild, ref node.rightchild);
    return maxkey;
}
```

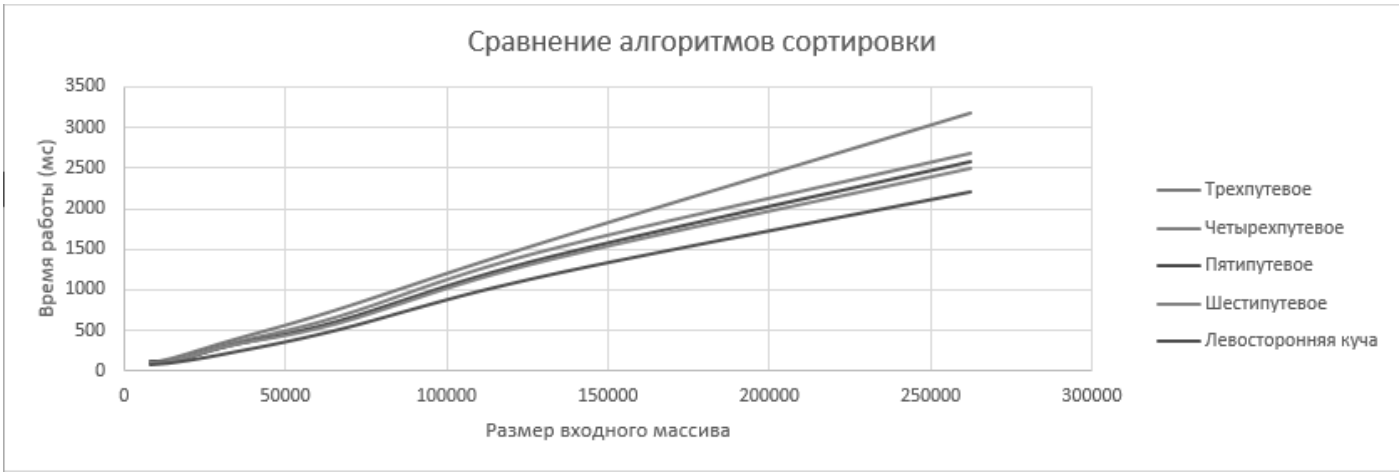



```
static List<Node> MMergeSort(List<Node> array, int numberOfWays)
{
    if (array.Count() <= 1)
    {
        return array;
    }
    int initialSerial = array[0].serial;
    List<List<Node>> series = new List<List<Node>>();
    for (int i = 0; i < numberOfWays; i++)
    {
        series.Add(new List<Node>());
    }
    for (int i = 0; i < array.Count(); i++)
    {
        Node node = new Node();
        node.key = array[i].key;
        node.serial = i % numberOfWays;
        series[i % numberOfWays].Add(node);
    }
    for (int i = 0; i < numberOfWays; i++)
    {
        series[i] = MMergeSort(series[i], numberOfWays);
    }
    List<Node> merged = Merge(series);
    for (int i = 0; i < merged.Count(); i++)
    {
        Node node = new Node();
        node.key = merged[i].key;
        node.serial = initialSerial;
        merged[i] = node;
```

```
    }
    return merged;
}
static List<Node> Merge(List<List<Node>> series)
{
    List<int> indexes = new List<int>();
    MinQueue queue = new MinQueue();
    List<Node> merged = new List<Node>();
    int numberOfSteps = 0;
    for (int i = 0; i < series.Count(); i++)
    {
        indexes.Add(0);
        if (series[i].Count() > 0)
        {
            numberOfSteps += series[i].Count();
            queue.Append(series[i][0]);
            indexes[i] += 1;
        }
    }
    for (int i = 0; i < numberOfSteps; i++)
    {
        merged.Add(queue.ExtractMin());
        if (indexes[merged.Last().serial] < series[merged.Last().serial].Count())
        {
            queue.Append(series[merged.Last().serial][indexes[merged.Last().serial]]);
            indexes[merged.Last().serial] += 1;
        }
    }
    return merged;
}
```

4. Анализ

Время работы (мс)					
Сортировка левосторонней кучей	Сортировка многопутевым слиянием				Размер массива
	Трехпутевое	Четырехпутевое	Пятипутевое	Шестипутевое	
75	105	100	121	114	8192
106	180	134	130	137	16384
219	378	333	312	318	32768
496	756	657	606	583	65536
1173	1605	1486	1399	1367	131072
2201	3186	2679	2583	2499	262144



Так, сортировка левосторонней кучей оказывается эффективнее сортировки многопутевым слиянием вне зависимости от количества путей. Однако, несмотря на это, не столь большое отличие во времени работы многопутевого слияния от левосторонней кучи, а также удобство его использования во внешней сортировке, делают многопутевое слияние крайне полезным в определенных случаях.