Государственное образовательное учреждение высшего профессионального образования
Санкт-Петербургский национальный исследовательский университет
Информационных Технологий, Механики и Оптики
Факультет инфокоммуникационных технологий

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

По алгоритмам и структурам данных
На тему: «Сортировки»

Проверил(а):

_____

Дата: «___»_____201__г.

Оценка: _____

Работу выполнил(а):

Никончук А.П.
Студент(ка) группы К3242
Дневного отделения

_____

2018 г.

Цель:
Получить начальные знания об алгоритмах, применяемых для сортировки массивов, деревьев.
Задание:

1. Освоить или укрепить навыки написания алгоритмических единиц на новом языке программирования;
2. написать алгоритмы сортировки.

Задание согласно варианту:

1. Сортировка слиянием
2. Пирамидальная сортировка

Реализация:

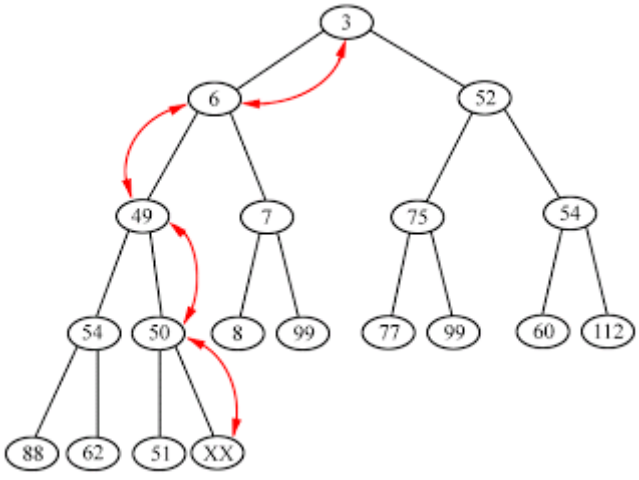
1. Описание принципа работы алгоритма

1.1. Сортировка слиянием

В основе алгоритма сортировки слиянием лежит возможность быстрого объединения упорядоченных массивов (или списков) так, чтобы результат оказался упорядоченным. Используется стратегия разделяй и властвуй. Алгоритм разделяет исходный массив на две части произвольным образом (обычно пополам), рекурсивно сортирует их и объединяет результат. Разделение происходит до тех пор, пока размер массива больше единицы, т.к. пустой массив и массив из одного элемента всегда отсортированы. На каждом шаге слияния из обоих списков выбирается первый необработанный элемент и помещается в рабочий массив, который использует для объединения отсортированных половин. Элементы сравниваются, наименьший из них добавляется к результату и помечается как обработанный. Слияние происходит до тех пор, пока один из списков не окажется пуст.

1.2. Пирамидальная сортировка

Особенность алгоритма заключается в представлении данных структурой пирамиды, в которой к вершине располагаются наибольшие элементы. Зачастую работу сортировки показывают на основе древовидной структуры и при создании из массива новой структуры – кучи. Последовательно добавляется по одному элементу массива, при этом проверяется не больше ли он, чем его родиитель, если больше, то они меняются месами. При изменении также затрагиваются другие родительские подветки. Так проверки будут доходить до корня дерева или до тех пор, пока не найдется предок больше, чем добавленный элемент.
В представленной вариации массив не преобразовывается в кучу и она вовсе не создается. Происходит иметация структуры засчет выбора 3х элементов – локального родителя и его детей, если они существуют (те длинна массива позволяет). Так как есть риск выйти за пределы массива задание значений индексов происходит в завсимости от проверки четности длины массива.



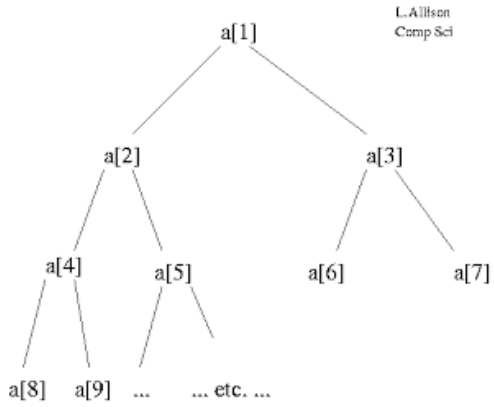
Индекс вычисляется по формуле:

Root = 0

Parent = каждый третий элемент массива начиная с 0 элемента

left child = parent index * 2 + 1

right child = parent index * 2 + 2



2. Код программы на 2 языках программирования

2.1. JavaScript

Функция Mergesort рекурсивно проходится разделяя массив напополам пока не доходит до массива из одного элемента, затем при возвращении сортирует и объединяет в единый массив половины при помощи merge. Merge в свою очередь сравнивает элементы, если не была окончена одна из отсортированных частей – в этом случае оставшаяся часть второй половины перемещается в рабочий массив, и добавляет наименьшую перемещая “курсор” массива, из которого добавлен элемент.

```
function Mergesort(main_array){
	if (main_array.length>1){
		var mlength = main_array.length;
		var part1 = main_array.splice(0, mlength/2);
		var part2 = main_array;
		console.log('part1 ' + part1);
		console.log('part2 ' + part2);
		part1 = Mergesort(part1);
		part2 = Mergesort(part2);
		var res = merge(part1, part1.length, part2, part2.length);
	} else{
		return main_array;
	}
	return res;
}
function merge(array1, size1, array2, size2){
	var res = [];
	var i = 0;
	var j = 0;
	while(true)
	{
		//console.log(i + ' ' + j);
		if (i>=size1){
			//console.log('i>=size1');
			for (var j0=j; j0<size2; j0++){
				res.push(array2[j0]);
			}
			break;
		}
		if (j>=size2){
			//console.log('j>=size2');
			for (var i0=i; i0<size1; i0++){
				res.push(array1[i0]);
			}
			break;
		}
		console.log('if '+ array1[i]+ '<'+ array2[j]);
```

```
                if (array1[i] < array2[j]){
                        res.push(array1[i]);
                        i++;
                } else{
                        res.push(array2[j]);
                        j++;
                }
                console.log(res);
        }
        console.log(res);
        return res;
}

function HeapSort(array)
{
        console.log(array);
        var result = [];
        var parent, left_child, right_child;
        var len = array.length;
        while (result.length<len){
                for (i=array.length-1; i>0; i=i-2){
                        if (i%2==0){
                                parent = (i-2)/2;
                                left_child = i-1;
                                right_child = i;
                        }
                        if (i%2==1){
                                parent = (i-1)/2;
                                left_child = i;
                                right_child = parent*2+2;
                        }
                        console.log(left_child, parent, right_child);
                        console.log(array[left_child], array[parent], array[right_child]);

                        if (array[parent] < array[right_child]){
                                console.log('switch parent and right child');
                                [array[parent], array[right_child]] = [array[right_child], array[parent]];

                        }
                        if (array[parent] < array[left_child]){
                                console.log('switch parent and left child');
                                [array[parent], array[left_child]] = [array[left_child], array[parent]];

                        }
                        if (array[0] < array[parent]){
                                console.log('switch root and parent');
                                [array[0], array[parent]] = [array[parent], array[0]];
                        }
                        console.log('i='+i+' /'+array);
                }
                result.push(array.shift());
        }
        return result;
}
```

2.2. Java

```
static int[] MergeSort(int[] main_array) {

    int[] res;
    if (main_array.length > 1) {
        int mlenght = main_array.length;
        int[] part1 = Arrays.copyOfRange(main_array, 0, main_array.length / 2);
        int[] part2 = Arrays.copyOfRange(main_array, main_array.length / 2,
main_array.length);
        printArr(part1);
        printArr(part2);
        part1 = MergeSort(part1);
        part2 = MergeSort(part2);
        res = merge(part1, part1.length, part2, part2.length);
    } else {
        return main_array;
    }
    return res;
}
private static int[] push(int[] array, int value)
{
    int[] new_arr = new int[array.length + 1];
    for (int i=0; i < array.length; i++){
        new_arr[i] = array[i];
    }
    new_arr[array.length] = value;
    return  new_arr;
}
private static int[] firstElemDelete(int[] array)
{
    int[] res = new int[array.length - 1];
    for (int i=0; i < array.length - 1; i++){
        res[i] = array[i + 1];
    }
    //System.out.print("\n");
    //printArr(res);
    return res;
}
private static void printArr(int[] arr)
{
    System.out.print("\n[");
    for (int i=0; i < arr.length; i++){
        System.out.print(" " + arr[i]);
    }
    System.out.print(']');
}
private static int[] merge(int[] array1, int size1, int[] array2, int size2) {
    //System.out.println("in MergeSort");
    int[] res = new int[0];
    int i = 0;
    int j = 0;
    while (true) {
        if (i >= size1) {
            //System.out.println("i >= size1");
            for (int j0 = j; j0 < size2; j0++) {
                res = push(res, array2[j0]);
            }
            break;
        }
        if (j >= size2) {
            //System.out.println("j >= size2");
            for (int i0 = i; i0 < size1; i0++) {
                res = push(res, array1[i0]);
            }
            break;
        }
        //System.out.println("if " + array1[i] + " < " + array2[j]);
```

```
        if (array1[i] < array2[j]) {
            res = push(res, array1[i]);
            i++;
        } else {
            res = push(res, array2[j]);
            j++;
        }
        printArr(res);
    }
    //printArr(res);
    return res;
}
private static int[] HeapSort(int[] arr)
{
    int[] res = new int[0];
    int parent = 0;
    int left_child = 0;
    int right_child = 0;
    int some_elem = 0;
    int len = arr.length;
    while (res.length < len) {
        //System.out.print( "\n " + arr.length + " " + left_child + " " + parent + " " +
right_child);
        for (int i=arr.length - 1; i > 0; i=i-2){
            if (i%2 == 0) {
                parent = (i - 2) /2;
                if (i - 1 < arr.length) {
                    left_child = i - 1;
                }
                if (i < arr.length) {
                    right_child = i;
                }
            }
            if (i%2 == 1) {
                parent = (i - 1) / 2;
                if (i < arr.length) {
                    left_child = i;
                }
                if (parent*2+2 < arr.length) {
                    right_child = parent * 2 + 2;
                } else {
                    right_child = parent;
                }
            }
            if ( arr[parent] < arr[right_child] ) {
                some_elem = arr[parent];
                arr[parent] = arr[right_child];
                arr[right_child] = some_elem;
            }
            if ( arr[parent] < arr[left_child] ) {
                some_elem = arr[parent];
                arr[parent] = arr[left_child];
                arr[left_child] = some_elem;
            }
            if ( arr[0] <arr[left_child]) {
                some_elem = arr[0];
                arr[0] = arr[parent];
                arr[parent] = some_elem;
            }
            printArr(arr);
        }
        res = push(res, arr[0]);
        arr = firstElemDelete(arr);
    }
    return res;
}
```

3. Оценка сложности алгоритма

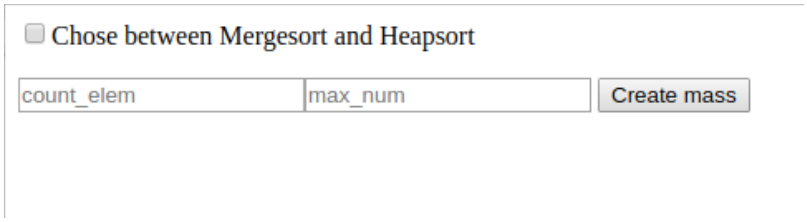
Алгоритмы выбраны примерно равные – логорифм от количества элементов.

3.1. Слияния $Nlog_2N$

3.2. Кучей NlogN

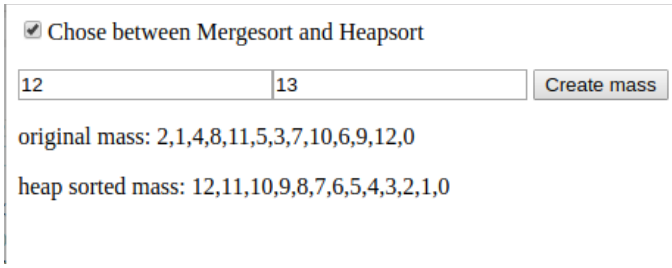
4. Пример входных данных

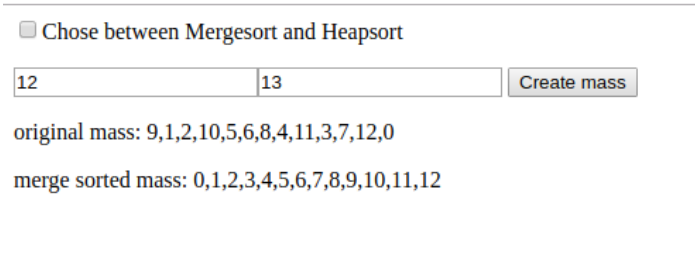
Вместо самостоятельного ввода массива чисел для сортировок происходит их генерация. Для создания массива на языке js первоначально происходит набор чисел, затем они перемешиваются. На вход требуется два числа – размер этого исходного массива для выборки и количество выбираемых элементов.

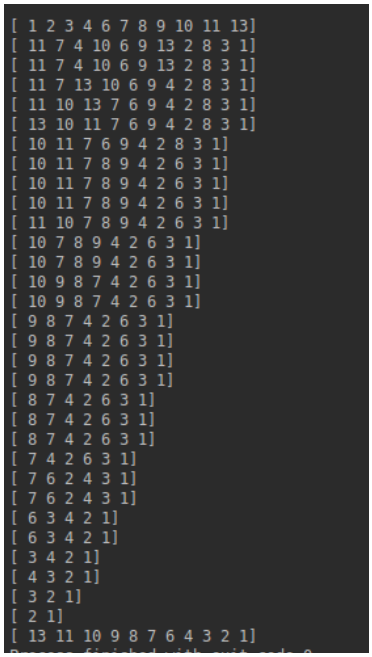


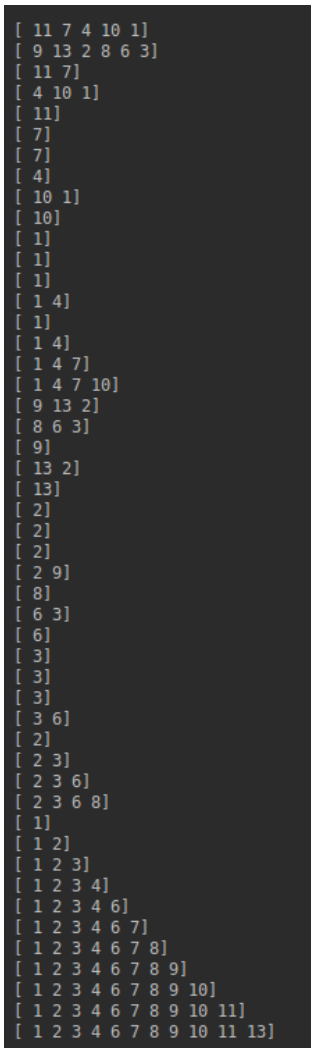
5. Примеры результатов работы

В первую очередь показаны скрины Кучей, затем Слиянием (слева вниз)









Выводы:

В ходе написания сортировок на языках программирования Java и javascript получены навыки работы с представлением структур данных при истолькозовании разных парадигм программирования, также были изучены 2 способа сортировки массива без лишних входных и выходных данных, проиллюстрирована работа алгоритмов сотрировок кучей и слиянием и показана разница в скорости работы алгоритмов.

http://studlab.com/news/algoritmy_poiska/2011-05-31-110

Сортировка слиянием: визуализация