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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Сортировки

Студент гр. 9382 _____ Субботин М. О.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург
2020

Цель работы.

Реализовать сортировку, разобраться в ее работе, а также оценить ее достоинства и недостатки.

Задание

14. Сортировка массивов слиянием – простое слияние, рекурсивная реализация.

Элементами массивов будут целые числа. После сортировки элементы должны располагаться в неубывающем порядке.

Описание алгоритма

Сортировка слиянием предполагает разделение большой задачи – сортировки всего массива, на более мелкие подзадачи. Исходный массив разделяется на две половины, затем каждая из этих двух половин разделяется еще на половины и т.д. до тех пор, пока подмассивы больше нельзя будет разделить, т.е. они будут представлять из себя один элемент. На эти “разделения” можно смотреть как на бинарное дерево, листьями которого являются подмассивы единичной длины. После такого разделения сравниваются листья, имеющие одного и того же родителя – подмассива, от которого эти элементы были разделены. В соответствии со знаком сравнения, элементы встают в “правильный” порядок в подмассиве, являющимся их родителем и он оказывается отсортированным. Такое действие, как сравнение и вставка элементов в родительский подмассив называется “слиянием” и выполняется до тех пор, пока не останется детей в дереве. Для того, чтобы слить два подмассива не единичной длины, их элементы поочередно сравниваются, подходящий элемент добавляется в родительский подмассив, а не подходящий в следующий раз будет сравниваться со следующим элементом из массива с ранее подошедшим элементом. Когда один из подмассивов опустеет, мы просто добавляем поочередно все оставшиеся элементы другого подмассива в родительский. С помощью таких действий родительский подмассив оказывается отсортированным. Получается, что операцией слияния это дерево

“восстанавливается” в корень, т.е. алгоритм сначала разделил проблему, а потом сравнивая элементы восстанавливает уже отсортированный массив.
В данной задаче, применяется рекурсивная реализация данного алгоритма. Она заключается в том, чтобы рассматривать сначала левые половины, а потом уже правые. Можно привести аналогию с обходом в ширину, когда проходятся по уровням и обход в глубину, когда идут в “глубь” слева на право.

Сортировка слиянием в лучшем, среднем и худших случаях имеет одинаковую асимптотику n*log n. Что на самом деле является как достоинством, так и не достатком. Для худшего случая такое время работы является достоинством, т.к. это можно сказать лучшее время для сортировок в принципе, к примеру сортировка пузырьком посчитает в худшем случае за n^2. В случае же, когда весь массив уже отсортирован, всё равно асимптотика будет n*log n, и это уже является недостатком, т.к. существуют сортировки, которые в таком случае отработают за n, только чтобы убедится, что массив отсортирован.
Из-за рекурсии эта сортировка требует дополнительное место в памяти, в отличии от той же сортировки вставками, где массив сортируется на месте, это является ее недостатком.
Также можно отметить, как достоинство, стабильность этой сортировки. К примеру, если бы мы сортировали не числа, а какие-то объекты, и два объекта имели бы одинаковое значение, по которому сортируется массив, то и до, и после сортировки эти элементы имеют одинаковое взаимно между собой расположение. Т.е. если A был левее B, то он и в отсортированном массиве будет левее B.

Описание функций алгоритма

void merge(int arr[], int left, int middle, int right, int depth) – эта функция отвечает за слияние, на момент вызова функции подмассивы [left, middle] и [middle+1,right] уже отсортированы.

int arr[] – исходный массив.

int left, int middle, int right – индексы для определения двух половин.

int depth – глубина рекурсии, нужна для корректного отображения промежуточных значений.

void mergeSort(int arr[], int left, int right, int &depth) – эта функция начинает сортировку, с помощью рекурсии разделяет массивы на половины и также вызывает функцию merge, которая производит их слияние.

int arr[] – исходный массив

int left, int right – индексы подмассива.

int &depth - глубина рекурсии, нужна для корректного отображения промежуточных значений.

Тестирование.

№	Входные данные	Выходные данные	Результат сортировки из стандартной библиотеки
1	10 60 4 10 7 8 3 5 12 0 6	0 3 4 5 6 7 8 10 12 60	0 3 4 5 6 7 8 10 12 60
2	1 100	100	100
3	10 10 9 8 7 6 5 4 3 2 1	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10
4	10 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10	-10 -9 -8 -7 -6 -5 -4 -3 -2 -1	-10 -9 -8 -7 -6 -5 -4 -3 -2 -1
5	10 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1
6	10 500 -250 3 6 24 99 1025 44 33 0	-250 0 3 6 24 33 44 99 500 1025	-250 0 3 6 24 33 44 99 500 1025
7	-1	Некорректная длина массива!	

Обработка результатов тестирования.

Программа выдает корректные результаты на всех тестах. Первое число в входных данных представляет из себя длину массива. Последний тест предназначен для проверки работы кода в случае, когда вводится длина массива меньшая либо равная нулю.

Выводы.

Была реализована сортировка, представлена и разобрана в деталях ее работа, и были оценены ее достоинства и недостатки.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <fstream>
#include <algorithm>
using namespace std;

void shift(int depth){
    for(int i = 0; i<depth; i++){
        cout << "   ";
    }
}

void printArr(int arr[], int left, int right, int depth){
    shift(depth);
    for(int i = left; i<=right; i++){
        cout << arr[i] << " ";
    }
}

void merge(int arr[], int left, int middle, int right, int depth){
    //вычисляем размеры двух подмассивов
    int leftSubArrSize = middle - left + 1;
    int rightSubArrSize = right - middle;

    /*
     * создаем два подмассива - левая и правая половины
     * начиная с индекса left до right
     * от исходного массива
     * оба эти массива по отдельности уже отсортированы
     */

    int leftSubArr[leftSubArrSize];
    int rightSubArr[rightSubArrSize];

    for(int i = 0; i<leftSubArrSize; i++){
        leftSubArr[i] = arr[left + i];
    }

    for(int i = 0; i<rightSubArrSize; i++){
        rightSubArr[i] = arr[middle + i + 1];
```

```
        }

        int leftIdx = 0;
        int rightIdx = 0;
        int mergedIdx = left;

        /*
         * Теперь мы сравниваем поочередно элементы из двух подмассивов.
         * Минимальный из двух сравниваемых элементов мы записываем в исходный
массив под индексом mergedIdx.
         * В подмассиве, который содержал минимальный из двух сравниваемых
элементов,
         * в следующий раз будем рассматривать следующий за ним элемент.
         */
        int counter = 1;
        while(leftIdx < leftSubArrSize && rightIdx < rightSubArrSize){
            shift(depth);
            cout    <<counter<<".1    Рассматриваемые    элементы:    "    <<
leftSubArr[leftIdx] << " и " << rightSubArr[rightIdx] << " ";
            if(leftSubArr[leftIdx] <= rightSubArr[rightIdx]){
                arr[mergedIdx] = leftSubArr[leftIdx];
                leftIdx++;
            }
            else {
                arr[mergedIdx] = rightSubArr[rightIdx];
                rightIdx++;
            }

            cout << ", min = "<< arr[mergedIdx] << "\n";
            shift(depth);

            cout << counter << ".2 Массив, образующийся в результате слияния:
[ ";
            printArr(arr,left,mergedIdx,0);
            cout << "]\n";

            shift(depth);
            cout << counter <<".3 Оставшиеся элементы в левом [ ";
            printArr(leftSubArr, leftIdx,leftSubArrSize-1, 0);
            cout << "], правом [ ";
            printArr(rightSubArr, rightIdx, rightSubArrSize-1, 0);
            cout << "] подмассивах\n";
```

```
            counter++;
            mergedIdx++;
        }

        /*
         * Когда мы запишем полностью один из подмассивов, цикл закончится,
         * но у нас еще останутся не записанные элементы из другого массива
         * здесь мы и займемся дозаполнением
         */
        shift(depth);
        if(leftIdx >= leftSubArrSize){
            cout << "Левый подмассив пуст. ";
        }
        else {
            cout << "Оставшиеся элементы из левого подмассива в том же порядке
вставляем в результирующий. ";
            while (leftIdx < leftSubArrSize) {
                arr[mergedIdx] = leftSubArr[leftIdx];
                leftIdx++;
                mergedIdx++;
            }
        }
        if(rightIdx >= rightSubArrSize){
            cout << "Правый подмассив пуст. ";
        }
        else {
            cout << "Оставшиеся элементы из правого подмассива в том же
порядке вставляем в результирующий. ";
            while (rightIdx < rightSubArrSize) {
                arr[mergedIdx] = rightSubArr[rightIdx];
                rightIdx++;
                mergedIdx++;
            }
        }
        cout << "\n";
        shift(depth);
        cout <<"Результирующий массив: [ ";
        printArr(arr,left,right,0);
        cout << "] \n";
    }

    void mergeSort(int arr[], int left, int right, int &depth){
```

```
        /*
         * если сортируемый подмассив состоит больше чем из 1 элемента,
         * то его надо разделить и слить
         * 1 элементный подмассив - база рекурсии.
         */
        if(left < right){
            /*
             * мы считаем средний индекс таким образом из-за возможного
переполнения
             * при больших значениях left и right
             */
            int middle = left + (right - left)/2;
            printArr(arr,left,middle,depth);
            cout << " - рассматриваем левый подмассив \n";

            depth++;

            //сортируем левую часть
            mergeSort(arr, left, middle,depth);
            depth--;

            printArr(arr,middle+1,right,depth);
            cout <<" - рассматриваем правый подмассив  \n";

            depth++;
            //сортируем правую часть
            mergeSort(arr, middle + 1, right,depth);
            depth--;

            shift(depth);
            cout << "Левый [ ";
            printArr(arr,left,middle,0);
            cout<<"] и правый [ ";
            printArr(arr,middle+1,right,0);
            cout<<"] подмассивы отсортированы, сливаем их : \n";

            //сливаем две отсортированные части
            merge(arr, left, middle, right,depth);
        }
        else {
            shift(depth);
```

```
            cout << "Подмассив состоит из одного элемента " << arr[left] << "
и отсортирован \n";
        }
    }


    bool sortFunction(int i, int j){
        return (i<=j);
    }

    int main() {
        char inputCh='0';
        while(inputCh != 'c' && inputCh != 'f'){
            cout << "Напишите f/c для получения данных из файла/консоли: ";
            cin >> inputCh;
        }
        int arraySize = 0;
        if(inputCh == 'f') {
            ifstream infile("../in.txt");
            infile >> arraySize;
            if (arraySize <=0){
                cout << "Некорректная длина массива!";
                return 0;
            }
            infile.close();
        }
        else if (inputCh == 'c'){
            cout << "Введите количество элементов в массиве: ";
            cin >> arraySize;
            if(arraySize <=0){
                cout <<"Некорректная длина массива!";
                return 0;
            }
        }

        int arr[arraySize];

        if(inputCh == 'f') {
            ifstream infile("../in.txt");
            int passLength;
            infile >> passLength;
            for (int i = 0; i < arraySize; i++) {
```

```
        infile >> arr[i];
    }
    infile.close();
}
else if (inputCh == 'c'){
    cout << "Введите массив: ";
    for (int i = 0; i < arraySize; i++) {
        cin >> arr[i];
    }
}

cout << "Сортируемый массив: ";
for(int i = 0; i<arraySize; i++){
    cout << arr[i] << " ";
}
cout << "\n";

int arrChecker[arraySize];
for(int i = 0; i<arraySize; i++){
    arrChecker[i] = arr[i];
}

int depth = 1;
mergeSort(arr,0, arraySize-1,depth);

cout << "\nМассив после сортировки слиянием: ";
for(int i = 0; i<arraySize; i++){
    cout << arr[i] << " ";
}
cout << "\n";

sort(arrChecker,arrChecker+arraySize,sortFunction);

cout << "Массив после сортировки из стандартной библиотеки: ";
for(int i = 0; i<arraySize; i++){
    cout << arrChecker[i] << " ";
}
cout << "\n";

bool equal = true;
int idx = 0;
while(equal && idx<arraySize){
```

```
            if(arr[idx] != arrChecker[idx]){
                equal = false;
            }
            idx++;
        }
        if(equal){
            cout << "Массивы сортировки слиянием и из стандартной библиотеки
равны";
        }
        else {
            cout << "Массивы сортировки слиянием и из стандартной библиотеки
не равны";
        }
        return 0;
}
```