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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Алгоритмы сортировки

Студент гр. 9303 _____ Микулик Д.П.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург

2020

Цель работы.

Написание алгоритма сортировки в соответствии с вариантом задания, теоретическая оценка сложности алгоритма.

Задание.

Вариант 14

Написать алгоритм сортировки простым слиянием (рекурсивная реализация).

Основные теоретические положения.

Сортировка слиянием – алгоритм сортировки, который упорядочивает списки(или другие структуры данных, доступ к элементам которых можно получать только последовательно) в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

Решение задачи сортировки делится на следующие этапы:

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

Описание алгоритма

Массив будем считать упорядоченным, если его элементы отсортированы в порядке возрастания. Также, массив из одного элемента будем считать уже отсортированным, откуда получим условие выхода из

функции сортировки (если длина сортируемого массива равна 1). Изначально функции передается указатель на массив, его размер, указатель на массив-буфер, где будут храниться промежуточные данные. Далее, массив делится пополам, для каждой половины запускается функция сортировки, притом, в качестве «точек входа» в массив для левой половины передается начало исходного массива, для правой половины: текущая правая часть массива. Далее происходит слияние двух отсортированных частей: создаются 2 индекса, которые указывают на текущий номер элемента для левой и правой части соответственно, в цикле проходимся по элементам левой и правой частей, за одну итерацию сравниваются элемент из левой части с элементом из правой части, меньший из них записывается в буфер, а индекс увеличивается для той части массива, которая содержала меньший элемент. Далее происходит проверка, все ли элементы из обоих частей записаны в буфер. Если в одной из частей остались незаписанные в буфер элементы, то они просто копируются в буфер в том же порядке, в котором они были в исходном массиве. Последнее, что выполняется в функции – копирование элементов из буфера в текущий массив.

Теоретическая оценка сложности алгоритма: так как мы постоянно делим текущий массив пополам (пока не получим элементы длины 1), а эта операция выполнается за $O(\log n)$, также, для каждой части выполняется слияние, которое занимает $O(n)$ времени, то общая оценка сложности алгоритма $O(n*\log n)$. Сложность по памяти: $O(n)$, так как создается буфер длины, равной длине исходного массива.

Выполнение работы.

Для реализации задачи была реализована единственная функция – merge_sort(T* arr, T* buf, size_t size). Функция принимает указатели на исходный массив и массив-буфер, куда будут записываться промежуточные значения. Последним аргументом передается размер массива, притом, рамзер исходного массива и буфера обязаны совпадать.

Исходный код программы представлен в приложении А. Результаты тестирования включены в приложение Б

Выводы.

Был реализован набор классов, позволяющий работать с бинарным деревом, который обеспечивает базовый функционал для работы: чтение, вывод, обработка в соответствии с заданием бинарного дерева-формулы. Также было проведено тестирование программы. Следует отметить, в данном случае некорректными считаются те данные, которые не содержать искомых поддеревьев-формул.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <time.h>
#include <stdlib.h>

//#define N 13

template<typename T>
void merge_sort(T* array, T* buffer, size_t size){
    if(size > 1){
        T* arr = array;
        T* buf = buffer;
        size_t left_size = size / 2;
        size_t right_size = size - left_size;

        merge_sort(array, buffer, left_size);
        merge_sort(&array[left_size], buffer, right_size);

        size_t left_ind = 0;
        size_t right_ind = left_size;
        size_t i = 0;
        while((left_ind < left_size) || (right_ind < (size))){
            if(arr[left_ind] < arr[right_ind]){
                buf[i] = arr[left_ind];

                left_ind++;
            }
            else{
                buf[i] = arr[right_ind];

                right_ind++;
            }
            i++;
            if (left_ind == left_size){
                for(int j = right_ind; j < (size); j++){
                    buf[i] = arr[j];
                    i++;
                }
                break;
            }
            if (right_ind == size){
                for(int j = left_ind; j < left_size; j++){
                    buf[i] = arr[j];
                    i++;
                }
                break;
            }
        }

        for(int j = 0; j < size; j++){
            arr[j] = buf[j];
```

```
        }


    }
    else{
        return;
    }
}

int main(){
    srand(time(NULL));

    int N = rand()%20 + 1;

    int* arr = new int[N];
    int* buf = new int[N];

    for(int i = 0; i < N; i++){
        arr[i] = rand()%100;
    }
    for(int i = 0; i < N; i++){
        std::cout << arr[i] << " ";
    }
    std::cout << "\n";
    merge_sort(arr, buf, N);
    for(int i = 0; i < N; i++){
        std::cout << arr[i] << " ";
    }
    std::cout << "\n";
    delete [] arr;
    delete [] buf;
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Тестовые данные генерируются случайным образом.

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	5 96 89 23 25 35 87 82 0 37 52 63 67 17	0 5 17 23 25 35 37 52 63 67 82 87 89 96	Программа работает корректно.
2.	83 40 25 76 32 80 46 78 67 36 9 12 23 32 90 69	9 12 23 25 32 32 36 40 46 67 69 76 78 80 83 90	Программа работает корректно.
3.	22 7 14 74	7 14 22 74	Программа работает корректно.
4.	28 56 68 21 91 53 92 86 13 62 93 74 80 92 28 35 69	13 21 28 28 35 53 56 62 68 69 74 80 86 91 92 92 93	Программа работает корректно.
5.	52 74 73 59 99 71 36 31 6	6 31 36 52 59 71 73 74 99	Программа работает корректно.
6.	29 73 0 45 11 83 31 99 8 18 35 49 34 51 61 24 21 48	0 8 11 18 21 24 29 31 34 35 45 48 49 51 61 73 83 99	Программа работает корректно.
7.	51	51	Программа работает корректно.