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 _____ Молодцев Д.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

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

Задание.

Вариант 15

Написать алгоритм сортировки простым слиянием (итеративная 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 n/2 блоков по 2 элемента, где n – длина массива;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. После сортируются части из двух пар, четырех и так пока не отсортируются две половины исходного массива.

Описание алгоритма

Массив будем считать упорядоченным, если его элементы отсортированы в порядке возрастания. Сначала программа разбивает исходный массив на пары элементов, которые сортируются в порядке возрястания. После сортировки каждой пары, сортируются уже буферы из 4-х элементов (двух отсортированных ранее пар). Далее, таким же образом сортируется буфер из 8-ми элементов и так далее. Последним шагом сортировки будет сортировка буфера длиной, равной длине исходного массива, и являющегося двумя уже остортированными половинами

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* array, int size). Функция принимает указатель на исходный массив и размер данного массива.

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

Выводы.

Была реализована функция MergeSort, которая сортирует массив чисел любого типа способом сортировки слиянием, итеративным методом. Была проанализирована сложность алгоритма сортировки.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>

static int iter=0;
template<typename T>
void merge_sort(T* array, int& size){
    int left_index;
    int right_index;
    int left_bord;
    int mid_bord;
    int right_bord;
    for (int i = 1; i < size; i *= 2){
        for (int j = 0; j < size - i; j = j+2*i){
            iter++;
            left_index = 0;
            right_index = 0;
            left_bord = j;
            mid_bord = j + i;
            right_bord = j+2*i;
            if(right_bord >= size){
                right_bord = size;
            }
            T* sorted_array = new T[right_bord - left_bord];
            while (left_bord + left_index < mid_bord && mid_bord
+ right_index < right_bord){
                if (array[left_bord + left_index] <
array[mid_bord + right_index]){
                    sorted_array [left_index + right_index] =
array[left_bord + left_index];
                    left_index += 1;
                }
                else{
```

```
sorted_array[left_index+right_index]=array[mid_bord+right_index]
;
                    right_index += 1;
                }
            }
            while (left_bord + left_index < mid_bord){
                sorted_array [left_index + right_index] =
array[left_bord + left_index];
                left_index += 1;
            }
            while (mid_bord + right_index < right_bord){
                sorted_array[left_index + right_index] =
array[mid_bord + right_index];
                right_index += 1;
            }
            std::cout<<"Sorted part:\n";
            for (int k = 0; k < left_index + right_index; k++){
                array[left_bord + k] = sorted_array[k];
                std::cout<<sorted_array[k]<<" ";
            }
            std::cout<<"\nArray on "<<iter<<" iteration.\n";
            for(int k=0;k<size;k++){
                std::cout<<array[k]<<" ";
            }
            std::cout<<"\n";
            delete sorted_array;
        }
    }
}

int main() {
    int size_of_arr;
    std::cout<< "Enter size of sorting array:"<<std::endl;
```

```
    std::cin>>size_of_arr;
    float* array=new float[size_of_arr];
    std::cout<< "Enter your array:"<<std::endl;
    for(int i=0;i<size_of_arr;i++){
        std::cin>>array[i];
    }
    merge_sort(array,size_of_arr);
    std::cout<<"Array:\n";
    for(int i=0;i<size_of_arr;i++){
        std::cout<<array[i]<<" ";
    }
    std::cout<<"\nWas sorted by "<<iter<<" iterations.\n";
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Тестовые данные генерируются случайным образом.

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	11 8 3.3 4.5 12 1 9.4 45 99 194 0.2 -34	-34 0.2 1 3.3 4.5 8 9.4 12 45 99 194	Программа работает корректно.
2.	6 -4.3 5 0.5 23.1 -45.4 3	-45.4 -4.3 0.5 3 5 23.1	Программа работает корректно.
3. (тест на некорректный размер массива)	0	Size cannot be les than 1!	Программа работает корректно.
4.	15 -4.5 40 66 -12.3 4 89 10002 -3444 3 21 45 67 -450.6 123 90	-3444 -450.6 -12.3 -4.5 3 4 21 40 45 66 67 89 90 123 10002	Программа работает корректно.
5.	9 52 74 -1.73 59 99 71 36 31 -6	-6 -1.73 31 36 52 59 71 74 99	Программа работает корректно.
6.	17 29 73 0.45 -11 83 31 99 8 -0.0018 35 49 34 51 61 24 21 4.8	-11 -0.0018 0.45 4.8 8 21 24 29 31 34 35 49 51 61 73 83 99	Программа работает корректно.
7.	1 34	34	Программа работает корректно.