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: Сортировки.

Студентка гр. 9304 _____ Паутова Ю.В.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с алгоритмами сортировки. Реализовать один из алгоритмов на языке программирования C++.

Задание.

Вариант 16

Сортировка массивов слиянием – естественное слияние.

Описание алгоритма работы.

Программе с помощью аргументов командной строки подается файл, содержащий строку-массив. Сначала проверяется был ли передан программе файл, если да, то проверяем, открылся ли он для считывания. Если файл был открыт, то строка-массив считывается из него и файл закрывается, иначе выводится сообщение об ошибке открытия файла и программа завершается. Если программе не был передан файл для считывания, то строку-массив предлагается ввести через терминал. Строка-массив записывается в argument, после чего проверяется с помощью функции check_argument() на то, является она массивом чисел (возвращает true) или массивом символов/строк (возвращает false). В зависимости от значения, которое вернет функция check_argument(), строка-массив разбивается по пробелам на элементы и записывается в вектор чисел или вектор строк. Вектор копируется для сравнения реализованного алгоритма сортировки естественным слиянием (функция MergeSort()) с std::sort. Затем оба отсортированных массива выводятся в терминал.

Формат входных и выходных данных.

Входные данные представлены в виде строки, которая является массивом данных.

Выходные данные представлены в виде промежуточных значений на каждой итерации сортировки и двух итоговых отсортированных (MergeSort() и std::sort) массивов.

Описание основных структур данных и функций.

Функции:

- bool check_argument() – принимает ссылку на объект класса string и проверяет является строка массивом чисел или символов.
- void print() – печатает вектор.
- std::vector<T> read() – принимает ссылку на объект класса string и создает вектор заданного типа T.
- void MergeSort() – осуществляет алгоритм сортировки массива естественным слиянием: принимает ссылку на объект класса vector, создает два вспомогательных вектора, в которые записываются неубывающие подмассивы исходного массива, затем вызывается функция Merging(), вспомогательные вектора очищаются и начинается следующая итерация. Цикл не завершается, пока весь массив не отсортирован.
- void Merging() – принимает два объекта класса vector и ссылку на объект класса vector, объединяет два первых вектора-аргумента в порядке возрастания значений элементов и записывает результат в третий вектор-аргумент, затем выводит результат слияния.

Разработанный программный код см. в приложении А.

Тестирование.

Тестирование происходит с помощью bash-скрипта. Он с помощью команд терминала запускает программу, подавая на вход файлы с тестами из директории Tests, и выводит результат. Также запускает программу без указания файла и с указанием несуществующего файла test8.txt. Для запуска тестирования в консоли используется команда **make run_tests**.

Результаты тестирования см. в приложении В.

Выводы.

Произошло ознакомление с алгоритмами сортировки. Была реализована сортировка массива естественным слиянием на языке программирования C++.

Была разработана программа, считывающая строку-массив, создающая на её основе вектор и сортирующая данный вектор алгоритмом сортировки

массивов естественным слиянием. Сложность алгоритма сортировки массива естественным слиянием составляет $O(n^2)$. Для данного алгоритма требуется дополнительная память равная размеру исходного массива. Std::sort из стандартной библиотеки C++ в основном имеет сложность $O(n\log_2 n)$, так как выбирает наилучший алгоритм сортировки на основе переданных ему данных. Таким образом std::sort эффективнее сортировки естественным слиянием.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: **source/sort.cpp**

```
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <fstream>
#include <algorithm>

template<typename T>
void print(std::vector<T>& array,int start ,int end){
    for (int i = start; i < end; i++)
        std::cout << array[i] << " ";
    std::cout << "\n";
}

template<typename T>
void Merging(std::vector<T> vec1, std::vector<T> vec2, std::vector<T>&
 array, int pos){
    int i = 0;
    int j = 0;
    int index = pos;
    while(i < vec1.size()){
        if (vec1[i] < vec2[j]){
            array[index] = vec1[i];
            i += 1;
            index += 1;
        }
        else{
            array[index] = vec2[j];
            j += 1;
            index += 1;
            if (j == vec2.size())
                break;
        }
    }
    if (i < vec1.size()){
        while (i < vec1.size()){
            array[index] = vec1[i];
            index += 1;
            i += 1;
        }
    }
    if (j < vec2.size()){
        while (j < vec2.size()){
            array[index] = vec2[j];
            index += 1;
            j += 1;
        }
    }
    std::cout << "merging = ";
    print<T>(array, pos,index);

    std::cout << "array after merging = ";
```

```
    print(array, 0, array.size());
    std::cout << '\n';
}

template<typename T>
void MergeSort(std::vector<T>& array){
    std::vector<T> vec1{};
    std::vector<T> vec2{};
    bool end = false;
    int iteration = 0;
    do{
        int pos = 0;
        std::cout << "Iteration = " << iteration << "\n";
        std::cout << "array befor merging = ";
        print(array, 0, array.size());
        int i = 0;
        while(i < array.size()){
            vec1.clear();
            vec2.clear();

            vec1.push_back(array[i++]);
            while (array[i-1] <= array[i] && i < array.size()){
                vec1.push_back(array[i]);
                i += 1;
            }

            if (vec1.size() == array.size()){
                end = true;
                break;
            }
            else{
                if (i == array.size())
                    break;
            }

            vec2.push_back(array[i++]);
            while(array[i-1] <= array[i] && i < array.size()){
                vec2.push_back(array[i]);
                i += 1;
            }

            std::cout << "vec1 = ";
            print<T>(vec1, 0, vec1.size());
            std::cout << "vec2 = ";
            print<T>(vec2, 0, vec2.size());

            if (vec1.size() >= vec2.size())
                Merging<T>(vec1, vec2, array, pos);
            else
                Merging<T>(vec2, vec1, array, pos);

            pos += vec1.size() + vec2.size();

        }
        iteration += 1;
    }while(!end);
}
```

```
template<typename T>
std::vector<T> read(std::string& argument){
    std::stringstream ss(argument);
    std::vector<T> array{};
    T value;

    while(ss >> value){
        array.push_back(value);
        if (ss.peek() == ' '){
            ss.ignore();
        }
        if(ss.peek() == '\n'){
            break;
        }
    }
    return array;
}

bool check_argument(std::string& argument);

int main(int argc, char** argv){
    std::string argument;
    if (argc < 2){
        std::cout << "array = ";
        std::getline(std::cin, argument);
        std::cout << '\n';
    }
    else{
        std::ifstream in(argv[1]);
        if (in.is_open()){
            std::getline(in, argument);
            std::cout << "array = " << argument << "\n\n";
        }
        else{
            std::cout << "Faild to open " << argv[1] << std::endl;
            return 1;
        }
        in.close();
    }
    bool all_is_number = check_argument(argument);

    std::vector<std::string> array_string{};
    //std::vector<float> array_float{};
    std::vector<int> array_int{};
    if (all_is_number){
        array_int = read<int>(argument);
        std::vector<int> copy_array_int = array_int;
        MergeSort<int>(array_int);
        std::cout << "\narray_sort = ";
        print<int>(array_int, 0, array_int.size());
        std::cout << "\nstd::sort = ";
        std::sort(std::begin(copy_array_int),std::end(copy_array_int));
        print<int>(copy_array_int, 0, copy_array_int.size());

    }
    else{
        array_string = read<std::string>(argument);
        std::vector<std::string> copy_array_string = array_string;
```

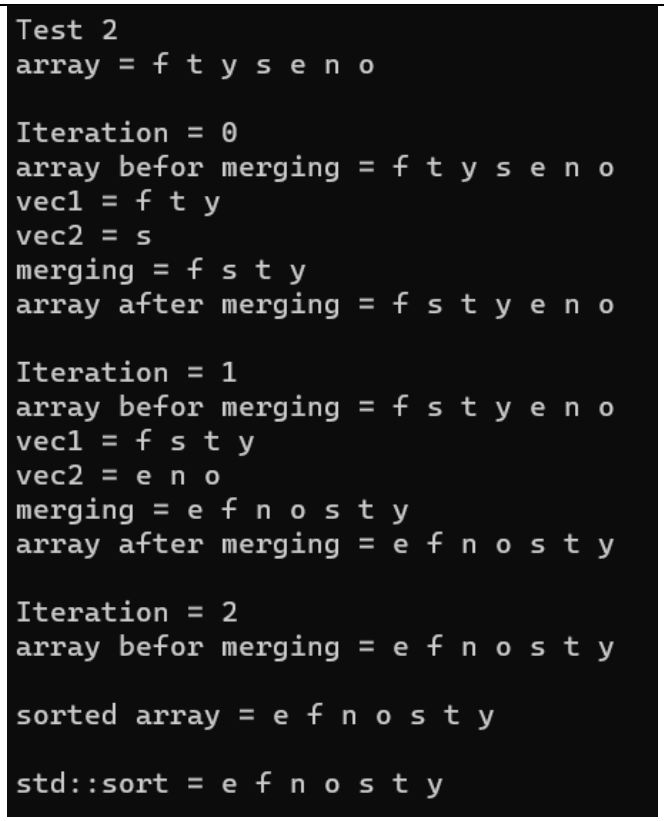
```
        MergeSort<std::string>(array_string);
        std::cout << "\narray_sort = ";
        print<std::string>(array_string, 0, array_string.size());
        std::cout << "\nstd::sort = ";
        std::sort(std::begin(copy_array_string),std::end(copy_array_st
ring));
        print<std::string>(copy_array_string, 0, copy_array_string.siz
e());
    }
    return 0;
}

bool check_argument(std::string& argument){
    auto iterator = argument.cbegin();
    while(iterator != argument.cend()){
        if(*iterator == '-'){
            iterator++;
        }
        if(!isdigit(*iterator)){
            return false;
        }
        while(isdigit(*iterator)){
            iterator++;
        }
        if((*iterator != ' ') && (iterator != argument.cend()) && (*it
erator != '.')){
            return false;
        }
        while(*iterator == ' '){
            iterator++;
        }
    }
    return true;
}
```

ПРИЛОЖЕНИЕ В
ТЕСТИРОВАНИЕ

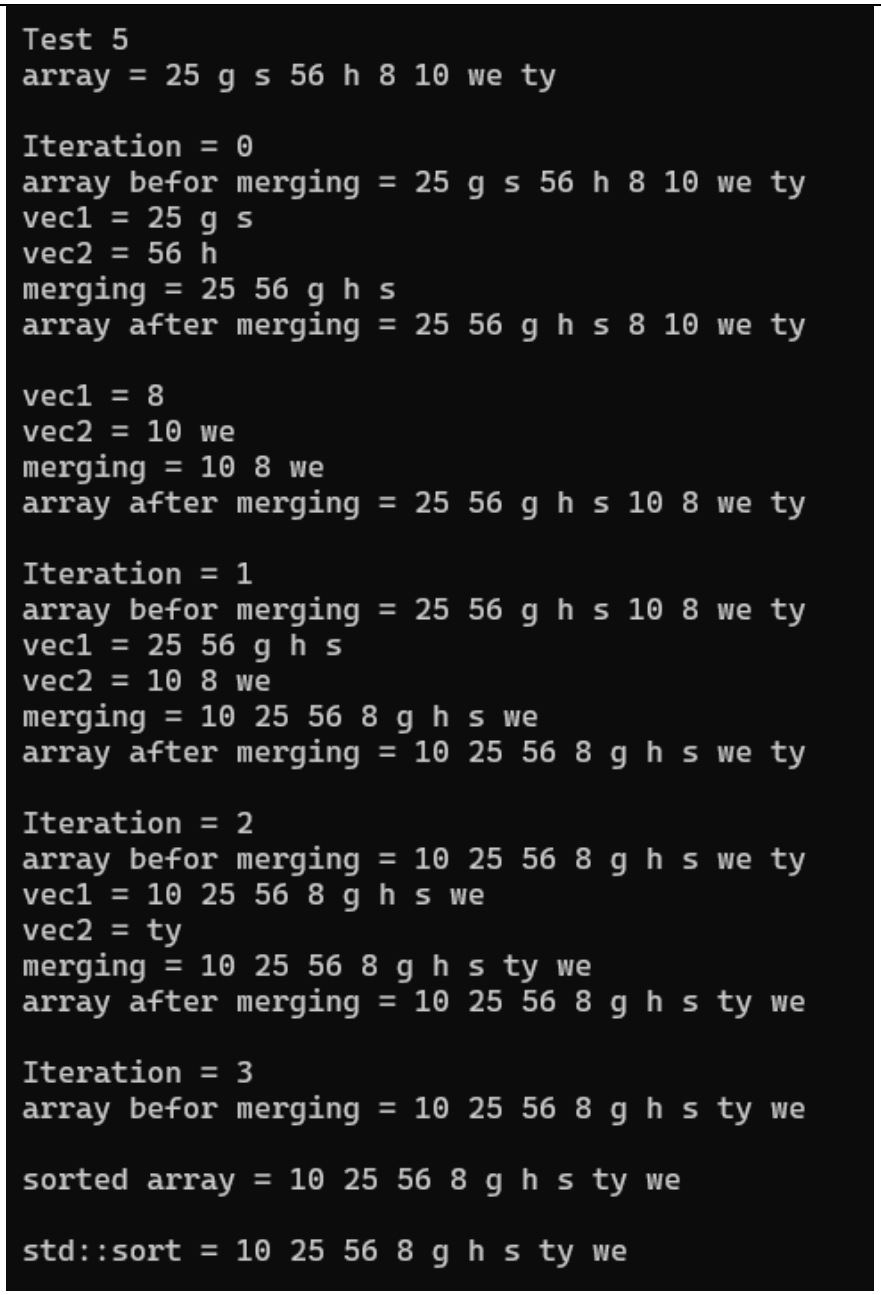
Таблица В.1 – Результаты тестирования

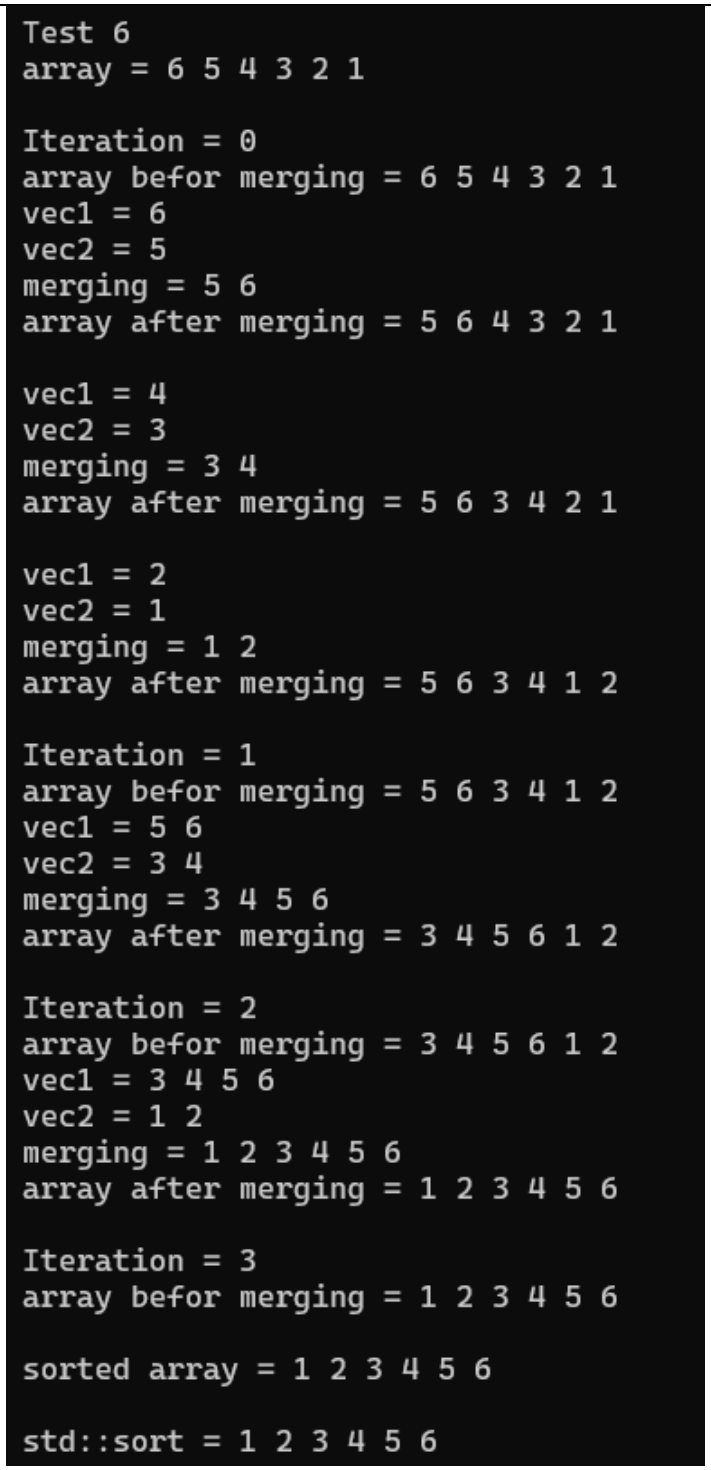
№	Тест	Комментарии
1	Test 1 array = 14 2 3 5 6 4 9 7 1 Iteration = 0 array befor merging = 14 2 3 5 6 4 9 7 1 vec1 = 14 vec2 = 2 3 5 6 merging = 2 3 5 6 14 array after merging = 2 3 5 6 14 4 9 7 1 vec1 = 4 9 vec2 = 7 merging = 4 7 9 array after merging = 2 3 5 6 14 4 7 9 1 Iteration = 1 array befor merging = 2 3 5 6 14 4 7 9 1 vec1 = 2 3 5 6 14 vec2 = 4 7 9 merging = 2 3 4 5 6 7 9 14 array after merging = 2 3 4 5 6 7 9 14 1 Iteration = 2 array befor merging = 2 3 4 5 6 7 9 14 1 vec1 = 2 3 4 5 6 7 9 14 vec2 = 1 merging = 1 2 3 4 5 6 7 9 14 array after merging = 1 2 3 4 5 6 7 9 14 Iteration = 3 array befor merging = 1 2 3 4 5 6 7 9 14 sorted array = 1 2 3 4 5 6 7 9 14 std::sort = 1 2 3 4 5 6 7 9 14	Был отсортирован массив чисел.

2			Был отсортирован массив символов.

3	Test 3 array = -1 -2 -3 -14 -5 -6 7 -8 -9 -10 Iteration = 0 array befor merging = -1 -2 -3 -14 -5 -6 7 -8 -9 -10 vec1 = -1 vec2 = -2 merging = -2 -1 array after merging = -2 -1 -3 -14 -5 -6 7 -8 -9 -10 vec1 = -3 vec2 = -14 -5 merging = -14 -5 -3 array after merging = -2 -1 -14 -5 -3 -6 7 -8 -9 -10 vec1 = -6 7 vec2 = -8 merging = -8 -6 7 array after merging = -2 -1 -14 -5 -3 -8 -6 7 -9 -10 vec1 = -9 vec2 = -10 merging = -10 -9 array after merging = -2 -1 -14 -5 -3 -8 -6 7 -10 -9 Iteration = 1 array befor merging = -2 -1 -14 -5 -3 -8 -6 7 -10 -9 vec1 = -2 -1 vec2 = -14 -5 -3 merging = -14 -5 -3 -2 -1 array after merging = -14 -5 -3 -2 -1 -8 -6 7 -10 -9 vec1 = -8 -6 7 vec2 = -10 -9 merging = -10 -9 -8 -6 7 array after merging = -14 -5 -3 -2 -1 -10 -9 -8 -6 7 Iteration = 2 array befor merging = -14 -5 -3 -2 -1 -10 -9 -8 -6 7 vec1 = -14 -5 -3 -2 -1 vec2 = -10 -9 -8 -6 7 merging = -14 -10 -9 -8 -6 -5 -3 -2 -1 7 array after merging = -14 -10 -9 -8 -6 -5 -3 -2 -1 7 Iteration = 3 array befor merging = -14 -10 -9 -8 -6 -5 -3 -2 -1 7 sorted array = -14 -10 -9 -8 -6 -5 -3 -2 -1 7 std::sort = -14 -10 -9 -8 -6 -5 -3 -2 -1 7	Был отсортирован массив чисел.

4		Test 4 array = summer spring winter autumn apple Iteration = 0 array befor merging = summer spring winter autumn apple vec1 = summer vec2 = spring winter merging = spring summer winter array after merging = spring summer winter autumn apple vec1 = autumn vec2 = apple merging = apple autumn array after merging = spring summer winter apple autumn Iteration = 1 array befor merging = spring summer winter apple autumn vec1 = spring summer winter vec2 = apple autumn merging = apple autumn spring summer winter array after merging = apple autumn spring summer winter Iteration = 2 array befor merging = apple autumn spring summer winter sorted array = apple autumn spring summer winter std::sort = apple autumn spring summer winter	Был отсортирован массив строк.

5		Был отсортирован массив символов.

6		Был отсортирован массив чисел.
7	Была выполнена команда ./lab4	Был введен и отсортирован массив.

```
Test 7
Веденная команда: ./lab4
array = -4 6 7 3 12 10 -6 16

Iteration = 0
array befor merging = -4 6 7 3 12 10 -6 16
vec1 = -4 6 7
vec2 = 3 12
merging = -4 3 6 7 12
array after merging = -4 3 6 7 12 10 -6 16

vec1 = 10
vec2 = -6 16
merging = -6 10 16
array after merging = -4 3 6 7 12 -6 10 16

Iteration = 1
array befor merging = -4 3 6 7 12 -6 10 16
vec1 = -4 3 6 7 12
vec2 = -6 10 16
merging = -6 -4 3 6 7 10 12 16
array after merging = -6 -4 3 6 7 10 12 16

Iteration = 2
array befor merging = -6 -4 3 6 7 10 12 16

sorted array = -6 -4 3 6 7 10 12 16

std::sort = -6 -4 3 6 7 10 12 16
```