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: Сортировка

Студент гр. 9304 _____ Ковалёв П. Д.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург
2020

Цель работы.

Изучить алгоритмы различных сортировок. Реализовать один из алгоритмов на языке программирования C++.

Задание.

Вариант 21

Соломонова сортировка.

Выполнение работы.

Сначала производится считывание массива данных в вектор из стандартной библиотеки. Параллельно считыванию элементов массива в вектор, происходит их считывание в дополнительный вектор, который будет сортироваться функцией *std::sort*. Допустим ввод как символов, так и целых чисел. После считывания, первый вектор передается в функцию *solomonSort()*, которая уже осуществляет саму сортировку.

Алгоритм сортировки:

Вычисляем минимальный (*min*) и максимальный (*max*) элементы вектора, после вычисляем дельту: *(max-min)/n = delta*, где *n* — длина вектора. Отсюда вытекает логичное ограничение, что должно выполняться условие *max — min >= n*. Данное условие проверяется в функции *main()*. После вычисления *delta*, в цикле *for* происходит вычисление индекса по формуле *indx = (vec[i] - min)/d + 1*, и в новый вектор *newVec* по данному индексу записывается *i*-ый элемент первоначального вектора. Если индекс совпадает при разных итерациях, то элемент кладется после первоначального элемента по тому же индексу. Имеем несколько «куч» элементов; кучи уже стоят в отсортированном порядке, однако элементы внутри куч не отсортированы.

В дальнейшем происходит «сбор камней», перед который исходный вектор очищается. В цикле *for* осуществляется проход по вектору *newVec*, содержащему кучи элементов. Если внутри кучи лежит один элемент,

сортировать ее не требуется, если лежат 2 элемента — сортировка так же не требуется, с помощью оператора *if* происходит поиск наибольшего и наименьшего элементов. Если лежат 3 и более элементов, куча сортируется сортировкой вставками. После этого, элементы из кучи помещаются в исходный вектор. Как результат, элементы в исходном векторе лежат в отсортированном порядке.

После завершения соломоновой сортировки, копия исходного вектора сортируется при помощи *std::sort*, результат выводится в терминал.

Тестирование.

Запуск программы начинается с запуска команды *make* в терминале, что приведет к созданию исполняемого файла *lab4*. Запуск программы начинается с ввода команды *./lab4* в терминале в директории *lab4*. Тестирование же проводится с помощью скрипта *tester.py*, который запускается командой *python3 tester.py* в командной строке в директории *lab4*. В текстовых файлах лежат входные данные. Подавать на вход программе нужно строку, элементы в которой разделены пробелами, а сама строка взята в кавычки.

В тестировании в выходных данных показано сравнение результата соломоновой сортировки с работой std::sort.

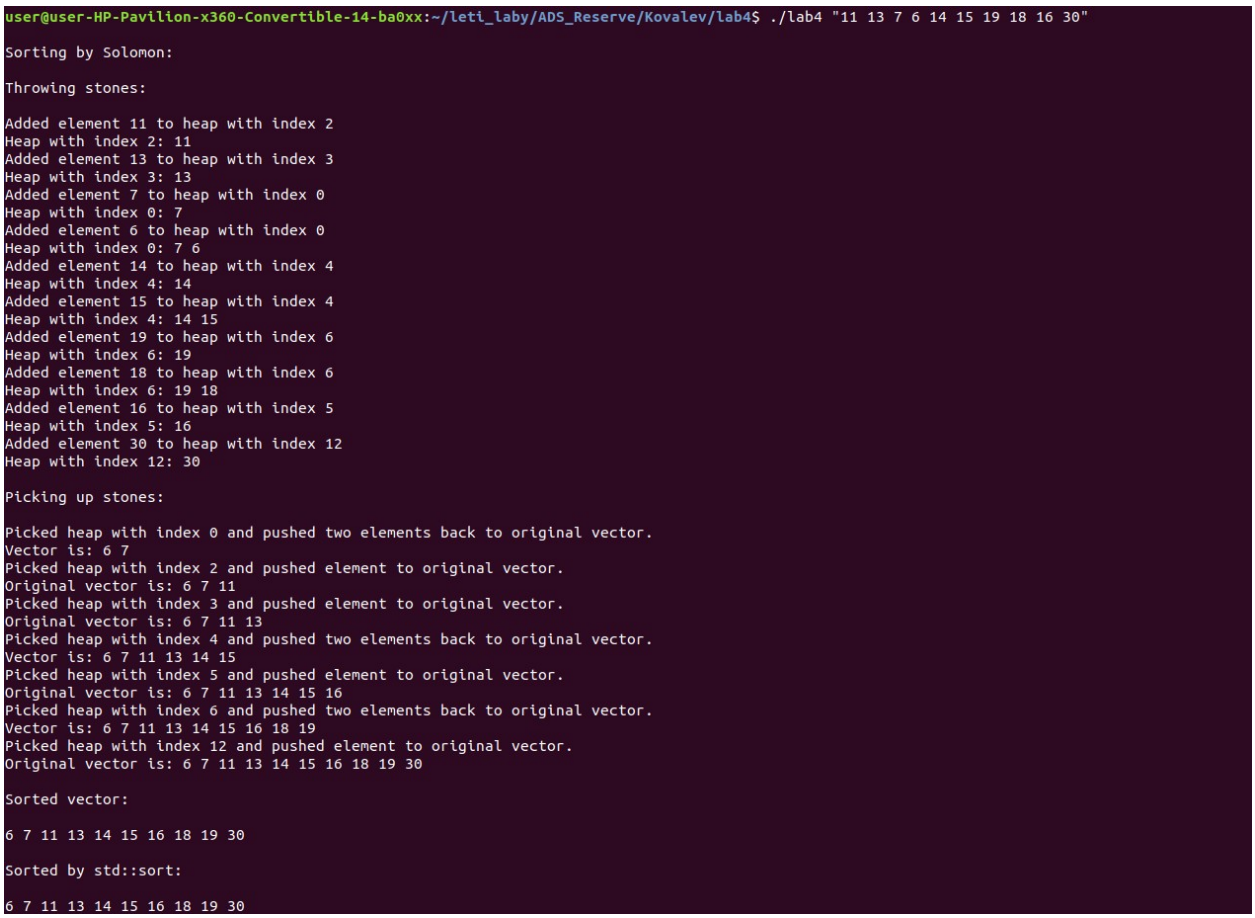

Рисунок 1 — Пример запуска программы

Результаты тестирования представлены в приложении Б.

Выводы.

Ознакомились с соломоновой сортировкой, реализовали алгоритм данной сортировки на языке программирования C++. Сложность соломоновой сортировки составляет $O(n^2)$, где n — количество элементов массива. Минусы данной сортировки — требования к входным данным (разность максимального и минимального элементов должна быть больше либо равна длине вектора исходных данных), и большое потребление памяти.

При выполнении задания использовался класс вектора из стандартной библиотеки. Вывод программы сравнивался с выводом std::sort.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

template<typename T>
void printer(std::vector<T>& vec){
    for(int i = 0; i < vec.size(); i++){
        std::cout << vec[i] << ' ';
    }
    std::cout << '\n';
}
template<typename T>
T findMin(std::vector<T>& vec){
    T num = vec[0];
    for(int i = 0; i < vec.size(); i++){
        if(vec[i] < num){
            num = vec[i];
        }
    }
    return num;
}
template<typename T>
T findMax(std::vector<T>& vec){
    T num = vec[0];
    for(int i = 0; i < vec.size(); i++){
        if(vec[i] > num){
            num = vec[i];
        }
    }
    return num;
}
template<typename T>
void insertSort(std::vector<T>& vec){
    T x;
    int i, j;
    for(i = 0; i < vec.size(); i++) {
        x = vec[i];
        for (j = i - 1; j >= 0 && vec[j] > x; j--) {
            vec[j + 1] = vec[j];
        }
        vec[j+1] = x;
    }
}
template<typename T>
void solomonSort(std::vector<T>& vec){
    T min, max;
    T n = vec.size();
    min = findMin(vec);
    max = findMax(vec);
```

```
        T d = ((int)max - (int)min)/n;
        std::vector<std::vector<T>> newVec;
        newVec.resize((max - min)/d + 1);
        std::cout << "\nThrowing stones:\n\n";
        for(int i = 0; i < n; i++){
            int indx = ((int)vec[i] - (int)min)/d + 1;
            std::vector<T> tmp;
            tmp.push_back(vec[i]);
            if(!newVec[indx - 1].empty()){
                newVec[indx - 1].push_back(tmp[0]);
            }else {
                newVec[indx - 1] = tmp;
            }
            std::cout << "Added element " << vec[i] << " to heap with
index " << indx - 1 << "\n";
            std::cout << "Heap with index " << indx - 1 << ": ";
            printer(newVec[indx - 1]);
        }
        vec.clear();
        std::cout << "\nPicking up stones:\n\n";
        for(int i = 0; i < newVec.size(); i++){
            if(!newVec[i].empty()){
                if(newVec[i].size() == 1){
                     std::cout << "Picked heap with index " << i << "
and pushed element to original vector." << "\n";
                    std::cout << "Original vector is: ";
                    vec.push_back(newVec[i][0]);
                }else if(newVec[i].size() == 2){
                     std::cout << "Picked heap with index " << i << "
and pushed two elements back to original vector." << "\n";
                    std::cout << "Vector is: ";
                    if(newVec[i][0] < newVec[i][1]){
                        vec.push_back(newVec[i][0]);
                        vec.push_back(newVec[i][1]);
                    }else{
                        vec.push_back(newVec[i][1]);
                        vec.push_back(newVec[i][0]);
                    }
                }else if(newVec[i].size() >= 3) {
                     std::cout << "Picked heap with index " << i << "
and pushed more than two elements to original vector." << "\n";
                    std::cout << "Vector is: ";
                    insertSort(newVec[i]);
                    for(int j = 0; j < newVec[i].size(); j++){
                        vec.push_back(newVec[i][j]);
                    }
                }
                printer(vec);
            }
        }
    }
    int main() {
        std::string args;
        getline(std::cin, args);
        int flag = 1;//1 - int, 2 - char
        if(isalpha(args[0])){
```

```
        flag = 2;
    }
    if(isdigit(args[0])){
        flag = 1;
    }
    std::istringstream str(args);
    int elem1;
    char elem2;
    std::vector<int> v1;
    std::vector<char> v2;
    std::vector<int> spareV1;
    std::vector<char> spareV2;
    if(flag == 1){//for int
        while(str >> elem1){
            v1.push_back(elem1);
            spareV1.push_back(elem1);
        }
        std::cout << "\nSorting by Solomon:\n";
        int min = findMin(v1);
        int max = findMax(v1);
        if(max - min < v1.size()){
            std::cout << "Incorrect massive!\n";
            return 0;
        }
        solomonSort(v1);
        std::cout << "\nSorted vector:\n\n";
        printer(v1);
        std::cout << "\nSorted by std::sort:\n\n";
        std::sort(std::begin(spareV1), std::end(spareV1));
        printer(spareV1);
    }
    if(flag == 2){//for char
        while(str >> elem2){
            v2.push_back(elem2);
            spareV2.push_back(elem2);
        }
        std::cout << "\nSorting by Solomon:\n";
        char min = findMin(v2);
        char max = findMax(v2);
        if((int)max - (int)min < v2.size()){
            std::cout << "Incorrect massive!\n";
            return 0;
        }
        solomonSort(v2);
        std::cout << "\nSorted vector:\n\n";
        printer(v2);
        std::cout << "\nSorted by std::sort:\n\n";
        std::sort(std::begin(spareV2), std::end(spareV2));
        printer(spareV2);
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	11 13 7 6 14 15 19 18 16 30	Solomon sort: 6 7 11 13 14 15 16 18 19 30 std::sort: 6 7 11 13 14 15 16 18 19 30	Вы
2.	13 75 28 56 18 46 34 14 44 37 82 15	Solomon sort: 13 14 15 18 28 34 37 44 46 56 75 82 std::sort: 13 14 15 18 28 34 37 44 46 56 75 82	
3.	a c b r y s j w q	Solomon sort: a b c j q r s w y std::sort: a b c j q r s w y	
4.	16 8 6 11 8 0 2 7 7 10 14 6	Solomon sort: 0 2 6 6 7 7 8 8 10 11 14 16 std::sort: 0 2 6 6 7 7 8 8 10 11 14 16	
5.	1 2 3 4 5 6 7 8 9 10 11	Sorting by Solomon: Incorrect massive!	Данные не удовлетворяю условию
6.	8 5 4 14 4 10 13 0 14 5 0 12	Solomon sort: 0 0 4 4 5 5 8 10 12 13 14 14 std::sort: 0 0 4 4 5 5 8 10 12 13	

		14 14	
7.	14 1 3 12 17 2 10 1 1 17 0 23	Solomon sort: 0 1 1 1 3 10 12 12 14 17 17 23 std::sort: 0 1 1 1 3 10 12 12 14 17 17 23	
8.	g k e i l a b d w q m l d e	Solomon sort: a b d d e e g i k l l m q w std::sort: a b d d e e g i k l l m q w	
9.	h o p w e n c a s r g h y	Solomon sort: a c e g h h n o p r s w y std::sort: a c e g h h n o p r s w y	
10.	g l a v d e t h a n l d e y u i o	Solomon sort: a a d d e e g h i l l n o t u v y std::sort: a a d d e e g h i l l n o t u v y	