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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Сортировки

Студент гр. 9304 _____ Борисовский В.Ю.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург

2020

Цель работы.

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

Задание.

Вариант 19.

Реализовать сортировку рассческой (comb sort).

Выполнение работы.

1) Сперва реализовали функцию std::vector<T> create_rand_vect(int size), которая принимается размерность массива и генирирует массив рандомных элементов типа T заданной размерности, при помощи функций srand() и rand().

2) Далее была реализована функция void comb_sort(std::vector<T> &data), которая выполняет алгоритм сортировки рассческой, а также вывод промежуточных шагов на экран и в заключение производит сравнение полученного отсортированного вектора с вектором полученным библиотечной сортировкой sort(), если векторы равны, то значит сортировку можно считать успешной и будет напечатано за сколько перестановок была выполнена данная сортировка.

Алгоритм сортировки рассческой основывается на том, что расстояние между сравневыемыми элементами можем быть гораздо больше чем 1, как в классической сортировке пузырьком. Основная идея заключается в том, чтобы оставить как можно меньше маленьких значений в конце списка (черепах), которые замедляют алгоритм сортировки.

Уменьшение расстояния просиходит по специально высчитанной формуле, то есть подбирается оптимально.

Изначально сравниваются элементы, расстояние между которым равно размерности массива - 1, далее этот шаг уменьшается. То есть сравниваются все элементы между которым возможно такое расстояние. К примеру если расстояние равно размерности массива — 1, то сравнить получится только первый и последний элементы.

3) После была разработана функци bool string_to_int_checker(char* str), которая пригодилась в дальнейшем и которая определяет можно ли строку конвертировать в число.

4) Затем была реализована функци void key(int argc, char** argv, int size), в которой определяются ключи программы, и вызывается функция сортировки для опредленного типа шаблона T в зависимости от переданного ключа.

5) В функции мейн проверяется лишь количество переданных аргументов при запуске программы, оно должно равняться трем - ключ и размерность массива (ну и ./lab4 само собой). Также выполняется проверка на то, что один из аргументов является типом int с помощью функции bool string_to_int(char* str).

Тестирование.

Запуск программы начинается с ввода команды “make”, что приведёт к компиляции программы и созданию исполняемого файла lab4. Запуск программы производится командой ./lab4 и последующим вводом валидных аргументов. Валидными аргументами является размерность массива - натуральное число и один из ключей -c, -i, -d и их длинные версии соответственно --char, --int, --double. Ключи нужны для установки типа вектора, который будет соритроваться. Тестирование производится с помощью скрипта test_skript.py. Запуск скрипта производится командой «python3 test_skript.py» в директории lab4. Результаты тестирования представлены в приложении Б.

Выводы.

Ознакомились с сортировкой рассческой. Сложность данной сортировки может занимать от O(n^2) это худшее время до O(nlogn) это лучшее время. При выполнении задания использовался класс вектора из стандартной библиотеки. Результат работы программы сравнивался с рузльтатом std::sort.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <getopt.h>
#include <cstdlib>

template <typename T>
void comb_sort(std::vector<T> &data){
    const double factor = 1.2473309; // добавлен const
    int step = data.size() - 1;
    int permutation_counter = 0;
    std::vector<T> check_vect = data;
    std::sort(check_vect.begin(), check_vect.end());



    while (step >= 1){
        std::cout << "distance between elements: " << (int)step << '\n';
        for (int i = 0; i + step < data.size(); i++){
            if (data[i] > data[i + step]){
                permutation_counter++;
                std::cout << permutation_counter << " permutation\n";
                std::cout << "the unmodified vector: ";
                for (int j = 0; j < data.size(); j++){
                    if (j == i){
                        std::cout << "\033[1;31m" << data[j]  << " \033[0m";
```

```
                    } else if (j == i + step){
                        std::cout << "\033[1;34m" << data[j]  << " \033[0m";
                    } else {
                        std::cout << data[j] << " ";
                    }
                }
                std::cout << '\n';
                std::swap(data[i], data[i + step]);
                std::cout << "the modified vector: ";
                for (int k = 0; k < data.size(); k++){
                    if (k == i){
                        std::cout << "\033[1;34m" << data[k]  << " \033[0m";
                    } else if (k == i + step){
                        std::cout << "\033[1;31m" << data[k]  << " \033[0m";
                    } else {
                        std::cout << data[k] << " ";
                    }
                }
                std::cout << '\n';
            }
        }
        std::cout << "\n\n";
        step /= factor;
    }
    if (data == check_vect){
            std::cout << "sorting spent " << permutation_counter << "
permutations\n";
    } else {
        std::cout << "sorting failed";
```

```
    }
}



// использована специализация шаблона функции
template <typename T>
std::vector<T> create_random_vector(int size);

template <>
std::vector<int> create_random_vector(int size){
    std::vector<int> vect;
    srand(time(0));
    for (int i = 0; i < size; i++){
        vect.push_back(rand() % 5000);
    }
    return vect;
}

template <>
std::vector<double> create_random_vector(int size){
    std::vector<double> vect;
    srand(time(0));
    for (int i = 0; i < size; i++){
        vect.push_back((rand() % 5000) * 0.1);
    }
    return vect;
}
```

```
template <>
std::vector<char> create_random_vector(int size){
    std::vector<char> vect;
    srand(time(0));
    for (int i = 0; i < size; i++){
        vect.push_back((char)(rand() % 95 + 32));
    }
    return vect;
}


bool string_to_int_checker(char* str){
    char* endptr;
    strtol(str, &endptr, 10);
    if (*endptr) {
        return false;
    } else {
        return true;
    }
}

void key(int argc, char** argv, int size){
    int opt;
    const char *opts = "cdi";
    struct option long_opts[] = {
            {"char", no_argument, NULL, 'c'},
            {"double", no_argument, NULL, 'd'},
            {"int", no_argument, NULL, 'i'},
            {0, 0, 0, 0}
```

```
    };
    int long_index;
    opt = getopt_long(argc, argv, opts, long_opts, &long_index);
    if (opt == 'c'){
        std::vector<char> arr = create_random_vector<char>(size);
        comb_sort(arr);
    } else if (opt == 'i'){
        std::vector<int> arr = create_random_vector<int>(size);
        comb_sort(arr);
    } else if (opt == 'd'){
        std::vector<double> arr = create_random_vector<double>(size);
        comb_sort(arr);
    } else {
        std::cout << "no such key";
    }
};

int main(int argc, char** argv){
    int size = 0;
    char *endptr;
    if (argc == 3){
        if (string_to_int_checker(argv[1])){
            size = strtol(argv[1], &endptr, 10);
            key(argc, argv, size);
        } else if (string_to_int_checker(argv[2])){
            size = strtol(argv[2], &endptr, 10);
            key(argc, argv, size);
        } else {
            std::cout << "you did not specify the size of the vector\n";
```

```
        }
    } else {
        std::cout << "you specified an invalid number of arguments\n";
    }

    return 0;

}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Результаты тестирования представлены в таблице Б.1

Таблица Б.1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные	Результат проверки
1.	-c 12 K c ; ~ u ' D ' Z G & Q	& ' ' ; D G K Q Z c u ~	sorting spent 11 permutations
2.	-i 11 3863 4437 917 4549 1820 1042 701 2687 763 4599 1636	701 763 917 1042 1636 1820 2687 3863 4437 4549 4599	sorting spent 15 permutations
3.	--double 5 386.3 443.7 91.7 454.9 182	91.7 182 386.3 443.7 454.9	sorting spent 5 permutations
4.	-f 11	no such key	Error
5.	--doubble 4	no such key	Error
6.	-q 2	no such key	Error
7.	-c q	you did not specify the size of the vector	Error
8.	--int 12.3	you did not specify the size of the vector	Error
9.	-d qwe	you did not specify the size of the vector	Error

Вывод промежуточного состояния массива на этапе каждой перестановки можно увидеть запустив тестирующий скрипт, либо же

запустив программу в ручную и передав ей валидные аргументы. (см. **Тестирование**).