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 _____ Куршев Е.О

Преподаватель _____ Филатов Ар.Ю

Санкт-Петербург
2020

Цель работы.

Написать свой алгоритм усовершенствованной быстрой сортировки с делением на 3 части. Оценить сложность алгоритма.

Задание

10 вариант.

Быстрая сортировка, рекурсивная реализация. Процедура трёхчастного разделения. Деление производится не на две группы, а на три: <x, =x, >x.

Основные теоретические положения.

Быстрая сортировка, сортировка Хоара, часто называемая qsort — алгоритм сортировки, разработанный английским информатиком Чарльзом Хоаром в 1960 году.

Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O(nlogn) обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена («Пузырьковая сортировка» и «Шейкерная сортировка»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы. В данной работе алгоритм усовершенствован и разделяет массив на 3 части.

Общая идея алгоритма состоит в следующем:

1. Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

2. Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».

3. Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

Выполнение работы

В ходе работы был создан следующий алгоритм:

Опорным элементом считается последний элемент массива. Далее идёт разбиение на три группы: больше, меньше или равны опорному элементу. Если найден элемент, равный опорному, то они образуют "блок" и начинают перемещаться вместе. После первой сортировки вызывается рекурсивно эта же функция, которая сначала обрабатывает ту часть, которая меньше опорного элемента, а затем ту часть, которая больше опорного элемента. Функция заканчивает свою работу, если в некоторый момент времени будет передано только одно число. Во избежание множественного копирования в функцию передаётся указатель на массив.

Выводы.

Была реализована функция my_qsort, которая производит сортировку массива с помощью метода быстрой сортировки, но разделяя элементы на 3 группы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>

template <typename T>
void my_qsort(T* arr, int start, int end){
    if(end - start > 0){
        int num_base = end;
        int cur = start;
        int quanity_base = 1;
        T base = arr[num_base];
        int size = num_base - cur;
        for(int i = 0; i < size; i++){
            if (base > arr[cur])
                cur += 1;
            else if (base == arr[cur]){
                for(int j = 0; j < num_base - cur - 1; j++){
                    T t = arr[cur + j];
                    arr[cur + j] = arr[cur + j + 1];
                    arr[cur + j + 1] = t;
                }
                num_base -= 1;
                quanity_base += 1;
            }
            else{
                T t = arr[cur];
                arr[cur] = arr[num_base - 1];
                arr[num_base - 1] = t;
                for(int k = 0; k < quanity_base; k++){
                    t = arr[num_base - 1 + k];
                    arr[num_base - 1 + k] = arr[num_base + k];
                    arr[num_base + k] = t;
                }
                num_base -= 1;
```

```
                }
            }
            std::cout << "less = ";
            if(num_base == 0)
                std::cout << "none ";
            else
                for(int i = 0; i < num_base; i++)
                    std::cout << arr[i] << ' ';
            std::cout << "base = ";
            for (int i = 0; i < quanity_base; i++)
                std::cout << base << ' ';
            std::cout << "larger = ";
            if(end + 1 - (num_base + quanity_base) == 0)
                std::cout << "none ";
            else
                for(int i = num_base + quanity_base; i < end + 1;
i++)
                    std::cout << arr[i] << ' ';
            std::cout << "\n";
            my_qsort(arr, start, num_base - 1);
            my_qsort(arr, num_base + quanity_base, end);
        }
    }

    int main(){
        int size;
        std::cout << "Enter size of sorting array: ";
        std::cin >> size;
        if (size >= 1){
            int *arr = new int[size];
            std::cout << "Enter your array: ";
            for (int i = 0; i < size; i++)
                std::cin >> arr[i];
            my_qsort(arr, 0, size - 1);
            std::cout << "Sorted array: \n";
```

```
        for (int i = 0; i < size; i++)
            std::cout << arr[i] << ' ';
    }
    else
        std::cout << "Error! Size < 1!!\n";
    return 0;
}
```