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

Студентка гр. 9303 _____ Хафаева Н. Л.

Преподаватель _____ Филатов Ар. Ю.

Санкт-Петербург
2020

Цель работы.

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

Задание.

Быстрая сортировка, рекурсивная реализация. Во время сортировки массив должен быть в состоянии:

элементы < x, элементы <= x, неотсортированные элементы.

Основные теоретические положения.

Быстрая сортировка, сортировка Хоара , часто называемая qsort— один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O(n\log n) обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

Описание алгоритма.

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена. 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.

Общая идея алгоритма состоит в следующем:

1. Выбрать из массива элемент, называемый опорным. В данной лабораторной работе опорный элемент всегда является первым элементом массива.
2. Сравнить все остальные элементы с опорным, если нашелся элемент больше опорного, то идет поиск элемента, который меньше. В случае, если нашелся элемент, который меньше опорного, его индекс записывается в переменную *success.* И элемент переставляется с опорным. В итоге получается массив из трех отрезков, следующих друг

за другом: «элементы меньшие опорного», «большие» и неотсортированные элементы.

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

Операция разделения массива на две части относительно опорного элемента занимает время $O(\log_2 n)$. Поскольку все операции разделения, проделываемые на одной глубине рекурсии, обрабатывают разные части исходного массива, размер которого постоянен, суммарно на каждом уровне рекурсии потребуется также $O(n)$ операций. Следовательно, общая сложность алгоритма определяется лишь количеством разделений, то есть глубиной рекурсии. В лучшем случае общая сложность алгоритма равна $O(n*\log_2 n)$. В худшем случае сложность алгоритма будет равна $O(n^2)$.

Выводы.

В ходе выполнения лабораторной работы я ознакомилась с алгоритмом сортировки QuickSort, изучила его особенности и реализовала программу, решающую поставленную задачу с помощью данного алгоритма.

Была реализована программа, которая содержит функцию my_qsort для сортировки элементов массива.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <time.h>
#include <stdlib.h>
#include <string>

using namespace std;

void my_qsort(int*& arr, int start, int end, int size) {
    if (end - start <= 0) {
        return;
    }

    int pivot = start;
    int success = start;
    int temp;
    int i = start + 1;

    while (i <= end) {
        if (arr[i] < arr[pivot]) {
            success = i;
        } else {
            temp = i + 1;
            while (temp <= end) {
                if (arr[temp] < arr[pivot]) {
                    success = i;
                    int cur = arr[temp];
                    arr[temp] = arr[i];
                    arr[i] = cur;
                    break;
                }
                temp++;
            }
            if (arr[i] >= arr[pivot]) {
                break;
            }
        }
        i++;
    }
    if (success != start) {
        int cur = arr[success];
        arr[success] = arr[pivot];
        arr[pivot] = cur;
    }
```

```
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << "Check from " << start << " to " << end << endl;
    my_qsort(arr, start, success - 1, size);
    my_qsort(arr, success + 1, end, size);
}

int main() {
    srand(time(NULL));
    int size = rand() % 10 + 5;
    int* arr = new int[size];
    for (int i = 0; i < size; i++) {
        arr[i] = rand() % 100;
    }

    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl << endl;
    my_qsort(arr, 0, size - 1, size);
    cout << endl;
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    delete[] arr;
    return 0;
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

	Входные данные	Результат	Комментарий
1	50 2 34 4 26 50 65 39 67	39 2 34 45 26 50 65 50 67 Check from 0 to 8 26 2 34 39 45 50 65 50 67 Check from 0 to 4 2 26 34 39 45 50 65 50 67 Check from 0 to 2 2 26 34 39 45 50 50 65 67 Check from 6 to 8 2 26 34 39 45 50 50 65 67	Программа работает корректно
2	90 56 46 22 27 28 90	28 56 46 22 27 90 90 Check from 0 to 6 27 22 28 56 46 90 90 Check from 0 to 4 22 27 28 56 46 90 90 Check from 0 to 1 22 27 28 46 56 90 90 Check from 3 to 4 22 27 28 46 56 90 90	Программа работает корректно
3	53 99 33 44 73 16	16 33 44 53 73 99 Check from 0 to 5 16 33 44 53 73 99 Check from 0 to 2 16 33 44 53 73 99 Check from 1 to 2 16 33 44 53 73 99 Check from 4 to 5 16 33 44 53 73 99	Программа работает корректно
4	43 62 94 3 13 92 57 22 73 97 97 4	4 3 13 22 43 92 57 62 73 97 97 94 Check from 0 to 11 3 4 13 22 43 92 57 62 73 97 97 94 Check from 0 to 3 3 4 13 22 43 92 57 62 73 97 97 94 Check from 2 to 3 3 4 13 22 43 73 57 62 92 97 97 94 Check from 5 to 11 3 4 13 22 43 62 57 73 92 97 97 94 Check from 5 to 7 3 4 13 22 43 57 62 73 92 97 97 94 Check from 5 to 6	Программа работает корректно

		3 4 13 22 43 57 62 73 92 94 97 97 Check from 9 to 11 3 4 13 22 43 57 62 73 92 94 97 97	
5	12 22 94 62 23 43 65	12 22 94 62 23 43 65 Check from 0 to 6 12 22 94 62 23 43 65 Check from 1 to 6 12 22 65 62 23 43 94 Check from 2 to 6 12 22 43 62 23 65 94 Check from 2 to 5 12 22 23 43 62 65 94 Check from 2 to 4 12 22 23 43 62 65 94	Программа работает корректно