Национальный исследовательский ядерный университет «МИФИ»

Институт интеллектуальных кибернетических систем

Кафедра №12 «Компьютерные системы и технологии»

ОТЧЁТ

О выполнении лабораторной работы № 5

"Исследование методов сортировки массивов данных"

Студент: Ким В.А.

Группа: Б22-703

Преподаватель: Овчаренко Е.С.

Москва - 2023

1. Формулировка индивидуального задания

Вариант № 92.

Структура данных

Деталь:

- идентификатор (строка длиной 8 символов);
- название (строка произвольной длины);
- количество (натуральное число).

2. Описание использованных типов данных

При выполнении данной лабораторной работы был а создана структура данных , в которой использовались тип данных int, предназначенный для работы с целыми числами с плавающей запятой и тип char для работы со строкой фиксированной и не фиксированной длины. А также использовались массивы, элементами которых были созданные структуры.

3. Описание использованного алгоритма

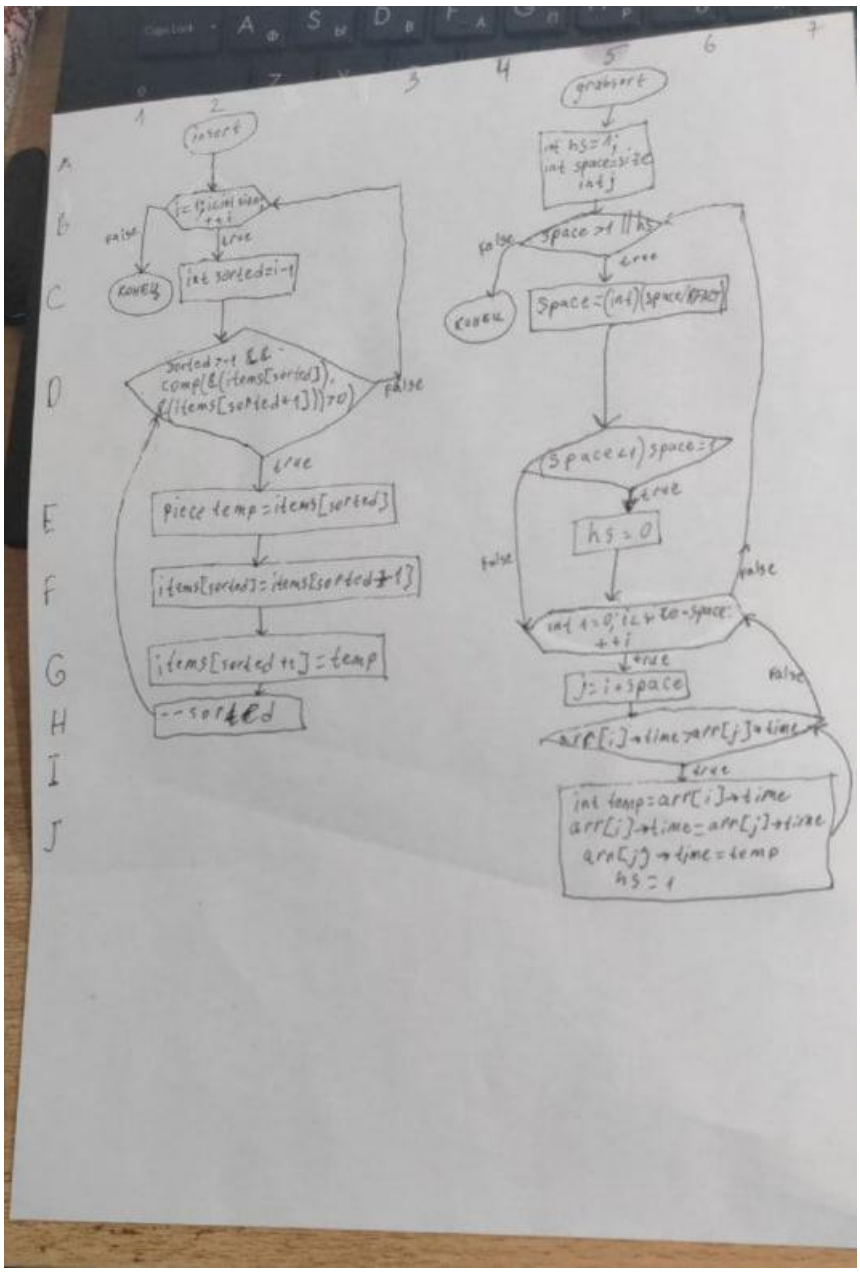

Рис. 1: Блок-схема алгоритма работы функций сортировки insort, grabsort

4. Исходные коды разработанных программ

```c
#include "./help.h"

int main(int argc, char** argv) {
    char* field = "id",
        *name_in_file="./in.t",
        *name_out_file="./result.txt";
    short way = ASC,
        algorithm = GrabSort,
        generation = 0;

    arg_get(argc, argv, &algorithm, &field, &way, &generation, &name_in_file, &name_out_file);

    if (!generation) {
        FILE* file_input = fopen(name_in_file, "r");
        FILE* file_output = fopen(name_out_file, "w");
        if (file_input == NULL) throw(open_file);

        char* temp = get_str_file(file_input);
        unsigned len = atoi(temp);
        free(temp);

        Piece* arr = malloc(len * sizeof(Piece));
        if (arr == NULL) throw(null_ptr);

        for (unsigned i = 0; i < len; ++i) {
            arr[i].id = get_str_file(file_input);
            char* temp = get_str_file(file_input);
            arr[i].count = atoi(temp);
            arr[i].name = get_str_file(file_input);

            free(temp);
        }

        int (*compare) (Piece*, Piece*) = compare_id_asc;
```

```
        if (strcmp(field, "id") == 0 && way == ASC)
            compare = compare_id_asc;
        else if (strcmp(field, "id") == 0 && way == DESC)
            compare = compare_id_desc;
        else if (strcmp(field, "count") == 0 && way == ASC)
            compare = compare_count_asc;
        else if (strcmp(field, "count") == 0 && way == DESC)
            compare = compare_count_desc;
        else if (strcmp(field, "name") == 0 && way == ASC)
            compare = compare_name_asc;
        else if (strcmp(field, "name") == 0 && way == DESC)
            compare = compare_name_desc;


        clock_t start = clock();
        if (algorithm == InsertSort)
            insort(arr, len, compare);
        else if (algorithm == GrabSort)
            grabsort(arr, len, compare);
        else qsort(arr, len, sizeof(Piece), (int (*)(const void*, const void*) )compare);
        printf(
            "%lf.15\n\n",
            (double)(clock() - start) / CLOCKS_PER_SEC
        );

        fprintf(file_output, "%d\n", len);
        for (int i = 0; i < len; ++i) {
            fprintf(file_output, "Id: %s\t\t", arr[i].id);
            fprintf(file_output, "Count: %d\t\t", arr[i].count);
            fprintf(file_output, "Name: %s\n", arr[i].name);
        }

        for (int i = 0; i < len; ++i) free(arr[i].id), free(arr[i].name);

        free(arr);
        fclose(file_input);
        fclose(file_output);
    }
    else if (generation) {
        FILE* file_output = fopen(name_out_file, "w");
        srand( time(NULL) );

        fprintf(file_output, "%d\n", generation);

        char* solution = "0123456789qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM";

        for (unsigned i = 0; i < generation; ++i) {
            for (unsigned i = 0; i < 8; ++i)
                fprintf(file_output, "%c", solution[(rand() % 62)]);
            fprintf(file_output, "\n");

            fprintf(file_output, "%u\n", rand()%1000+1);

            unsigned short len = rand() % 20 + 1;
            for (unsigned i = 0; i < len; ++i)
                fprintf(file_output, "%c", solution[(rand() % 62)]);
            fprintf(file_output, "\n");
        }

        fclose(file_output);
    }
    return 0;
}
```

5. Диаграммы сортировок

С помощью второй программы prog2.c было произведено таймирование. Перенаправив данные в Excel мы имеем следующие диаграммы сортировок.

кол-во	qsort	insort	grabsor
1000	0,00015	0,008	0,0007
5000	0,0007	0,18	0,005
10000	0,0015	0,77	0,01
15000	0,0023	1,73	0,015
20000	0,003	3,12	0,02
30000	0,005	7,5	0,025
40000	0,0075	15	0,032
50000	0,01	26,3	0,04
100000	0,02		0,09
250000	0,06		0,18
500000	0,14		0,68
1000000	0,3		1,53
5000000	1,84		7
10000000	3,75		15

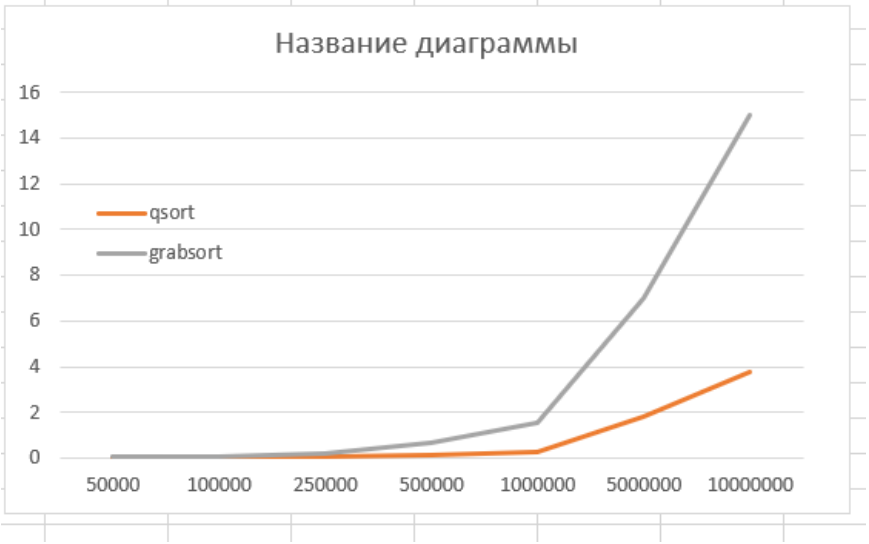

Рис: диаграмма сортировок qsort, insort, grabsort

Как видно из диаграмм, сортировка расческой имеет в среднем сложность O(n logn);

Быстрая сортировка имеет среднюю сложность: O(n logn);

Тогда как сортировка выбором имеет в среднюю сложность: O(n^2)

6. Выводы

В ходе выполнения данной работы на примере программы, которая производит сортировку массива структур были реализованы следующие этапы:

1. Организация ввода/вывода (из файла in.t в файл rezult.txt).

2. Создание структуры с тремя полями

3. Разработка функций. (Функции: compare_id_asc, compare_id_desc, compare_count_asc, compare_count_desc, compare_name_asc, compare_name_desc, insort, grabsort, arg_get)

4. Объявление и использование переменных.

5. Работа с памятью, использование функций malloc, realloc, free.

6. Сортировка структур по определенному полю

Исходя из графиков, мы узнали, что самая эффективная сортировка из реализованных – это quicksort, имеющая сложность O(NlogN)

Combsort имеет ту же сложность, но с другим коэффициентом, большим, чем NlogN

InsertionSort имеет сложность O(N^2), поэтому это самая худшая сортировка из всех предложенных