МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н. Э. Баумана

КАФЕДРА ПРОЕКТИРОВАНИЕ И ТЕХНОЛОГИЯ ПРОИЗВОДСТВА ЭЛЕКТРОННОЙ АППАРАТУРЫ

Отчет о выполнении
командного проекта №1
“Реализация алгоритмов сортировки пузырьком и пирамидальной сортировки на языке программирования Си”

по курсу «Функциональная логика и теория алгоритмов»

Выполнили: студенты гр. ИУ4-33Б

1. Фомичев Павел
2. Шадрин Юрий
3. Кондратив Владимир
4. Сусликов Антон
5. Салуев Евгений
6. Яковлев Иван

Проверил:
Терехов Владимир Владимирович

Москва 2020

Цель работы: реализовать принцип работы пирамидальной и пузырьковой сортировок.

1. Исходные данные:

Пирамидальная сортировка — алгоритм сортировки, работающий в худшем, в среднем и в лучшем случае (то есть гарантированно) за $\Theta(n \log n)$ операций при сортировке n элементов. Количество применяемой служебной памяти не зависит от размера массива (то есть, O(1)).

Сортировка пузырьком — простой алгоритм сортировки. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма: $O(n^2)$.

2. Выполнение

Основная часть

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <errno.h>

#include "bool.h"
#include "swap.h"
#include "bubblesort.h"
#include "heapsort.h"
#include "sorttype.h"
#include "gnuplot.h"


void shuffle(int *array, int n, SortType type) {
    if (n <= 1) return;
    int i;
    if (type == Heap) i = 1;
    else i = 0;
    for (; i < n - 1; i++) {
        size_t j = i + rand() / (RAND_MAX / (n - i) + 1);
        if (j == 0 && type == Heap) continue;
        int t = array[j];
        array[j] = array[i];
        array[i] = t;
    }
}


int main() {
    int crit_size, laps, tmp;
    SortType type;
    printf("Select type of sort:\n1.Bubble\n2.Heap\n");
    scanf("%d", &tmp);
    if (tmp == 1) type = Bubble;
    else if (tmp == 2) type = Heap;
```

```
    else {
        printf("Wrong sort type select");
        exit(130);
    }

    printf("Print critical array size:\n");
    scanf("%d", &crit_size);
    printf("Print number of laps:\n");
    scanf("%d", &laps);

    char *output;
    if (type == Bubble) output = "bubble_time.txt";
    else output = "heap_time.txt";
    FILE *total_time = fopen(output, "w");
    if (total_time == NULL) {
        exit(EEXIST);
    }


    clock_t start, stop;
    long n = 0;
    char number[10];
    int max;
    char *name = (char *) malloc(20 * sizeof(char));
    while (n != crit_size + 1) {

        int limit = 1000000;
        tmp = n;
        for (int i = 0; i < 9; i++) {
            if (n >= limit) n += limit;
            else if (limit >= 10) limit = limit / 10;
            if (tmp != n) break;
        }
        if (tmp == n) n += 1;
        if (n > crit_size) break;

        if (n < 1000) {
            sprintf(name, "./values/%lu.txt", n);
        } else if (n < 1000000) {
            sprintf(name, "./values/%luK.txt", n / 1000);
        }

        double mean = 0;
        for (int q = 0; q < laps; q++) {
            if (type == Heap) n += 1;
            double *values = (double *) calloc(laps, sizeof(double));
            int *arr = (int *) calloc(n, sizeof(int));

            FILE *block = fopen(name, "r");
            if (block == NULL) {
                exit(ENOENT);
```

```
            }

            tmp = 0;
            if (type == Heap) {
                arr[0] = n;
                tmp = 1;
            }
            for (int i = tmp; i < n; i++) {
                fscanf(block, "%[^,],", number);
                arr[i] = atoi(number);
            }

            fclose(block);
            system("sync");
            if (type == Heap) {
                start = clock();
                HeapSort(arr);
                stop = clock();
            } else {
                start = clock();
                BubbleSort(arr, n);
                stop = clock();
            }
            shuffle(arr, n, type);
            values[q] = (1000.0 / CLOCKS_PER_SEC) * (stop - start);

            mean += values[q];

            free(values);
            free(arr);

            if (type == Heap) n -= 1;
        }
        if (mean != 0.000) mean = mean / laps;
        else mean = 0;


        printf("\nThe average time for sorting of %ld elements was: %.3f ms\n", n,
               mean);
        total_time = fopen(output, "a");
        if (total_time == NULL) {
            exit(ENOENT);
        }

        fprintf(total_time, "%ld %f\n", n, mean);
        fclose(total_time);
        max = mean;
    }
    gnuplot(crit_size, max, type);
    return 0;
}
```

Пирамидальная сортировка

```
#include <stdio.h>

void heapify(int *arr, int root_id) {
    int is_completed = 0;
    int heap_bottom = arr[0];

    while (!(is_completed)) {

        int left = 2 * root_id, right = left + 1, swap_index = root_id;

        if (left <= heap_bottom && arr[left] > arr[swap_index])
            swap_index = left;
        if (right <= heap_bottom && arr[right] > arr[swap_index])
            swap_index = right;
        if (swap_index == root_id)
            is_completed = 1;
        else {
            swap(&arr[root_id], &arr[swap_index]);
            root_id = swap_index;
        }
    }
}

void HeapSort(int *arr) {
    int size = arr[0];
    for (int i = size / 2; i >= 1; i--)
        heapify(arr, i);

    while (arr[0] > 1) {
        swap(&arr[1], &arr[arr[0]]);
        arr[0]--;
        heapify(arr, 1);
    }
}
```

Сортировка пузырьком

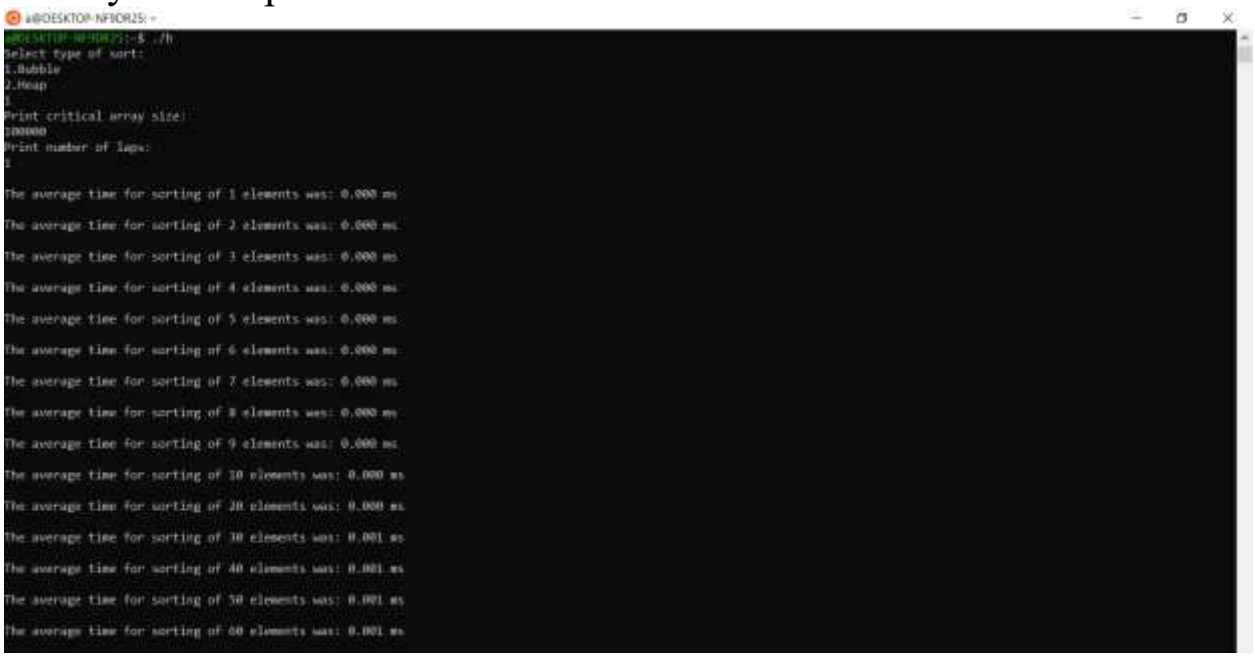
```
#include <stdio.h>


void BubbleSort(int *arr, int size) {
    for (int i = 0; i < size; i++) {
        bool flag = True;
        for (int j = 0; j < size - (i + 1); j++) {
            if (arr[j] > arr[j + 1]) {
                flag = False;
                swap(&arr[j], &arr[j + 1]);
            }
        }
        if (flag == True) break;}}
```

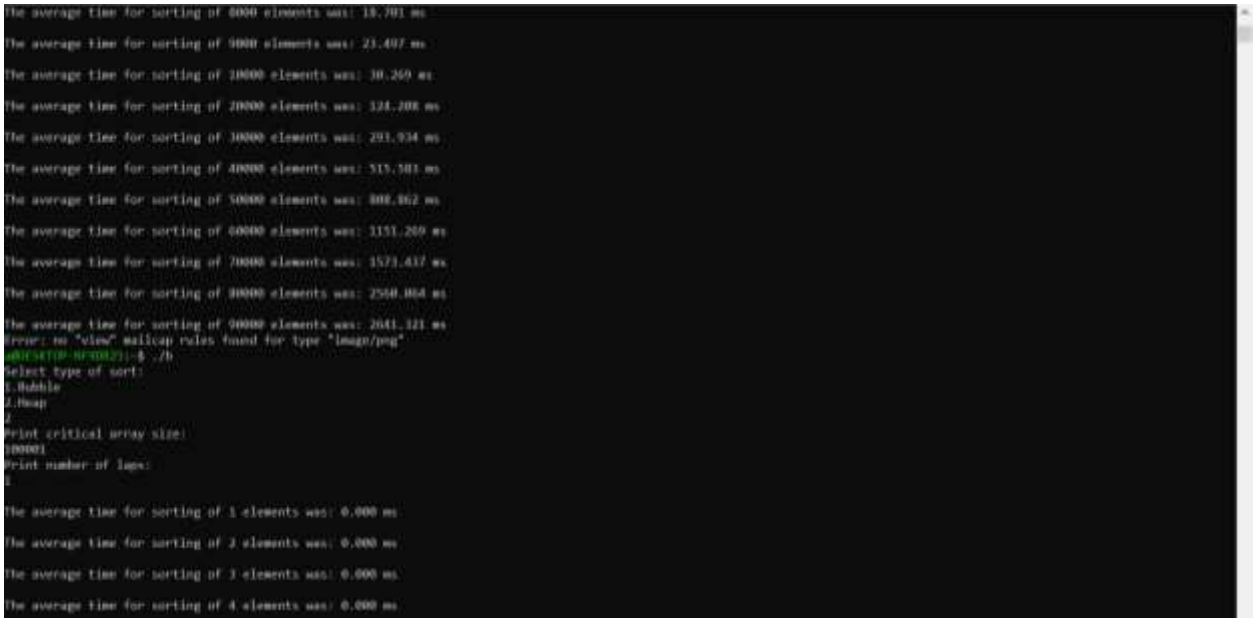
Код построения графиков

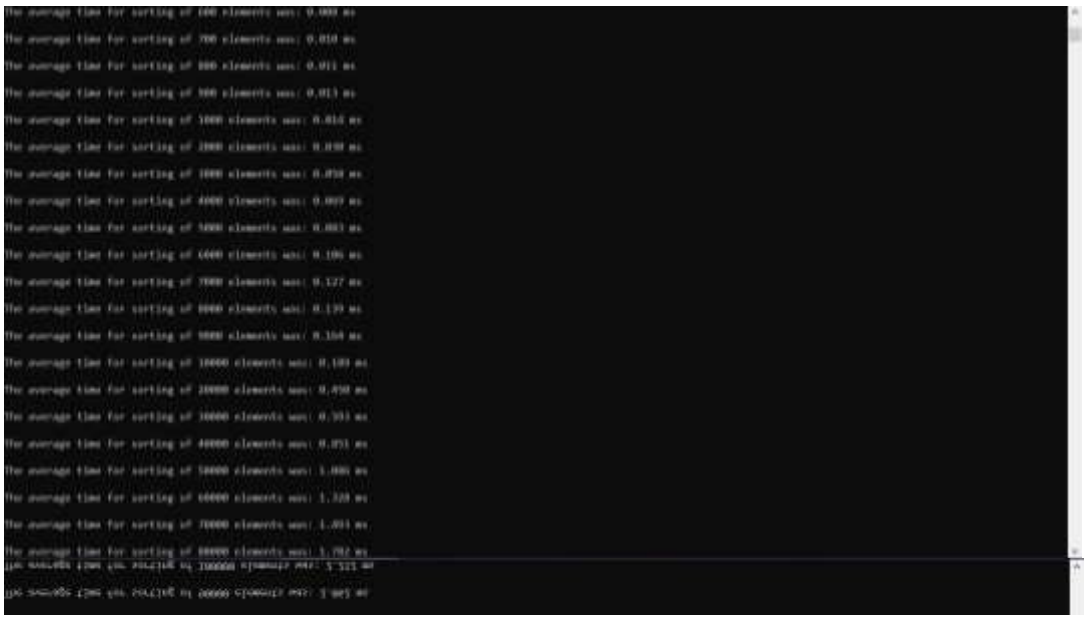
```
#include <unistd.h>

void gnuplot(int size, int max_time, SortType type) {

    if (type == Heap) {
        if (!access("heap_time.txt", 0))
            printf("File Present");
        else {
            printf("File heap_time.txt not found\n");
            exit(EEXIST);
        }
    } else {
        if (!access("bubble_time.txt", 0))
            printf("File Present");
        else {
            printf("File heap_time.txt not found\n");
            exit(EEXIST);
        }
    }
    FILE *fp = fopen("gnuplot.plt", "w");
    if (fp == NULL) {
        exit(EEXIST);
    }

    double to_sec = (1000.0 / CLOCKS_PER_SEC);
    fputs("set terminal png size 1600,900\n", fp);

    fputs("set output \"gnuplot.png\"\n", fp);

    fprintf(fp, "set xrange  [0:%d]\n", size + size / 5);
    fprintf(fp, "set yrange  [0:%d]\n", max_time + max_time / 5);

    fputs("set multiplot\n", fp);

    if (type == Heap) {
        fprintf(fp, "set key at %d, %d\n", size/6, max_time - 4 * max_time / 25);
        fprintf(fp, "plot x * log (x) * %f title \"x log x\" w l  lc 'red'\n", to_s
ec);
        fputs("set nokey\n", fp);
        fputs("plot \"heap_time.txt\" using 1:2 with linespoints lw 2 lt rgb '0xAAR
RGGBB'\n", fp);
        fputs("set nokey\n", fp);
    } else {
        fprintf(fp, "set key at %d, %d\n", size/6, max_time - 3 * max_time / 25);
        fprintf(fp, "plot x*x* %f title \"x^2\" w l lc 'blue'\n", to_sec);
        fputs("set nokey\n", fp);
        fputs("plot \"bubble_time.txt\" using 1:2 with linespoints lw 2 lt rgb '0x0
0000000'\n", fp);
        fputs("set nokey\n", fp);
```

```
    }
    fputs("set nomultiplot\n", fp);

    fclose(fp);

    system("gnuplot gnuplot.plt ");
}
```

3. Результаты работы



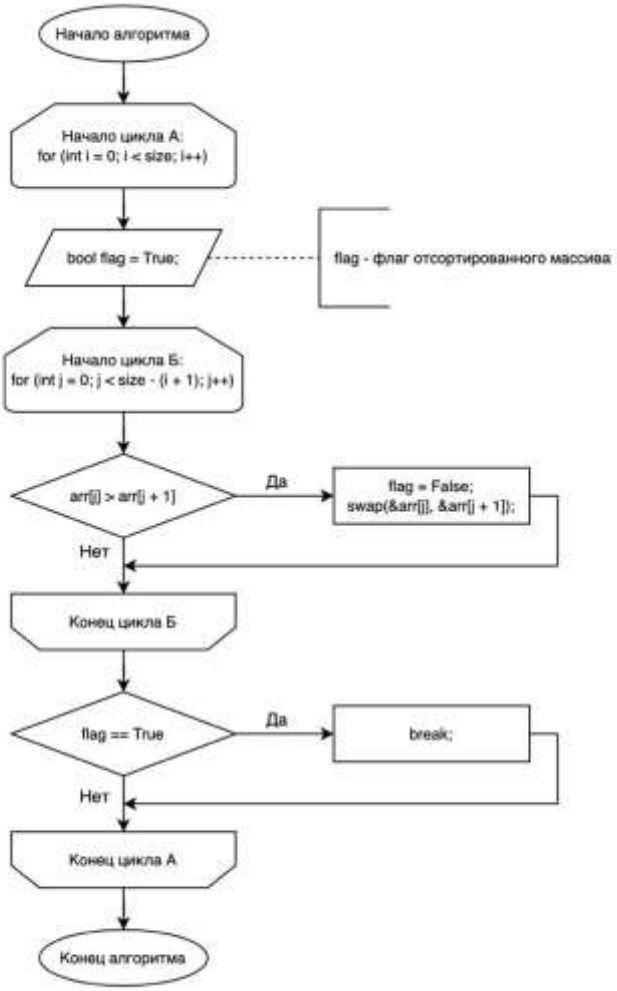




Блок схемы

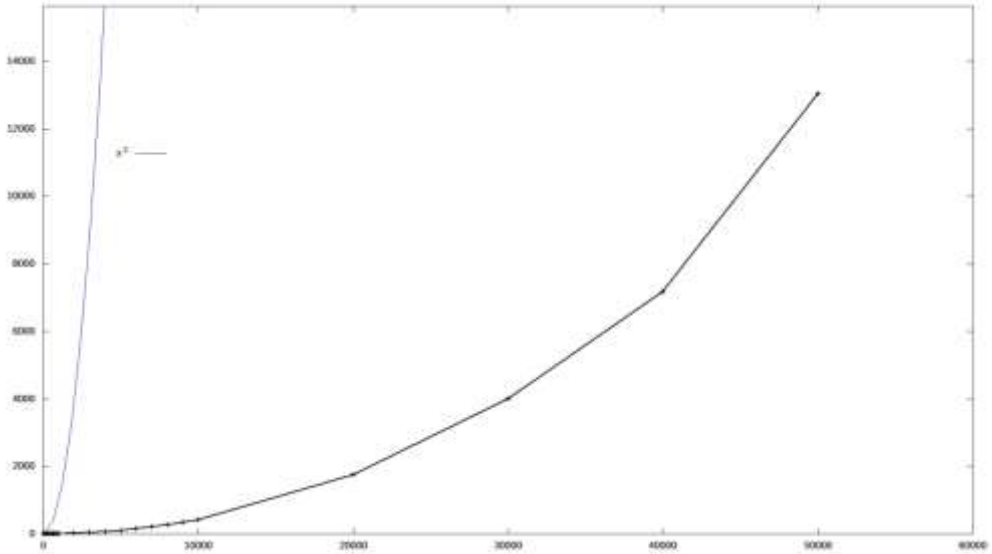
Пирамидальная сортировка

Сортировка пузырьком

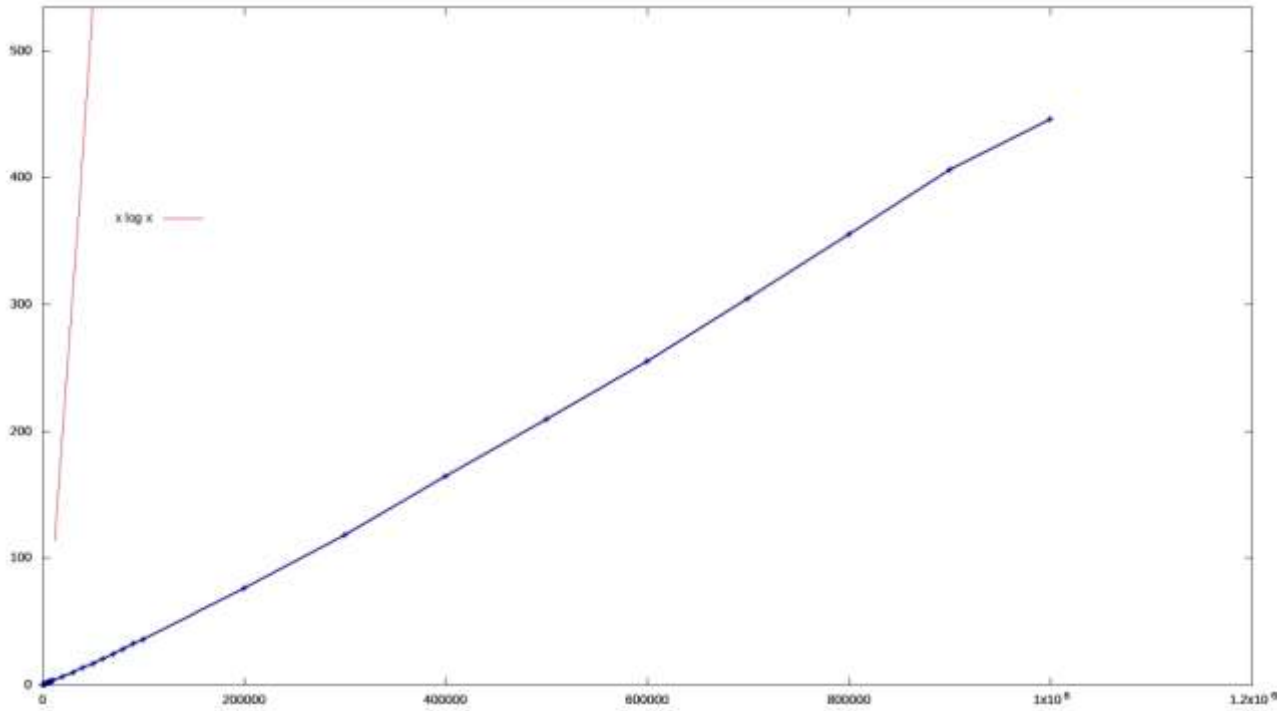


Графики

Сортировка пузырьком



Пирамидальная сортировка



4. Вывод:

В этой работе мы изучили принципы работы таких типов сортировок, как пирамидальная и пузырьковая, а также реализовали их в виде кода, который наглядно дает увидеть этот принцип в действии.