Национальный исследовательский ядерный университет «МИФИ»
Институт интеллектуальных кибернетических систем
Кафедра №12 «Компьютерные системы и технологии»

ОТЧЕТ

О выполнении лабораторной работы №5

«Исследование методов сортировки массивов данных.»

Студент: Титов Д.И.
Группа: Б22-505
Преподаватель: Вавренюк А.Б.

Москва — 2022

1. Формулировка индивидуального задания

Вариант №23

Структура данных

Запись в журнале событий:

- идентификатор (целое число);
- уровень важности (целое число, обозначающее один из следующих уровней: debug, info, warn, error, fatal);
- текст (строка произвольной длины).

Алгоритмы сортировки

1. Шейкерная сортировка (Shaker sort).
2. Пирамидальная сортировка (Heap sort).

2. Описание использованных типов данных

При выполнении данной работы были использованы типы данных int, char, указатели на char, А также собственные структуры данных удовлетворяющих условию задачи и указатели на них.

3. Описание использованного алгоритма

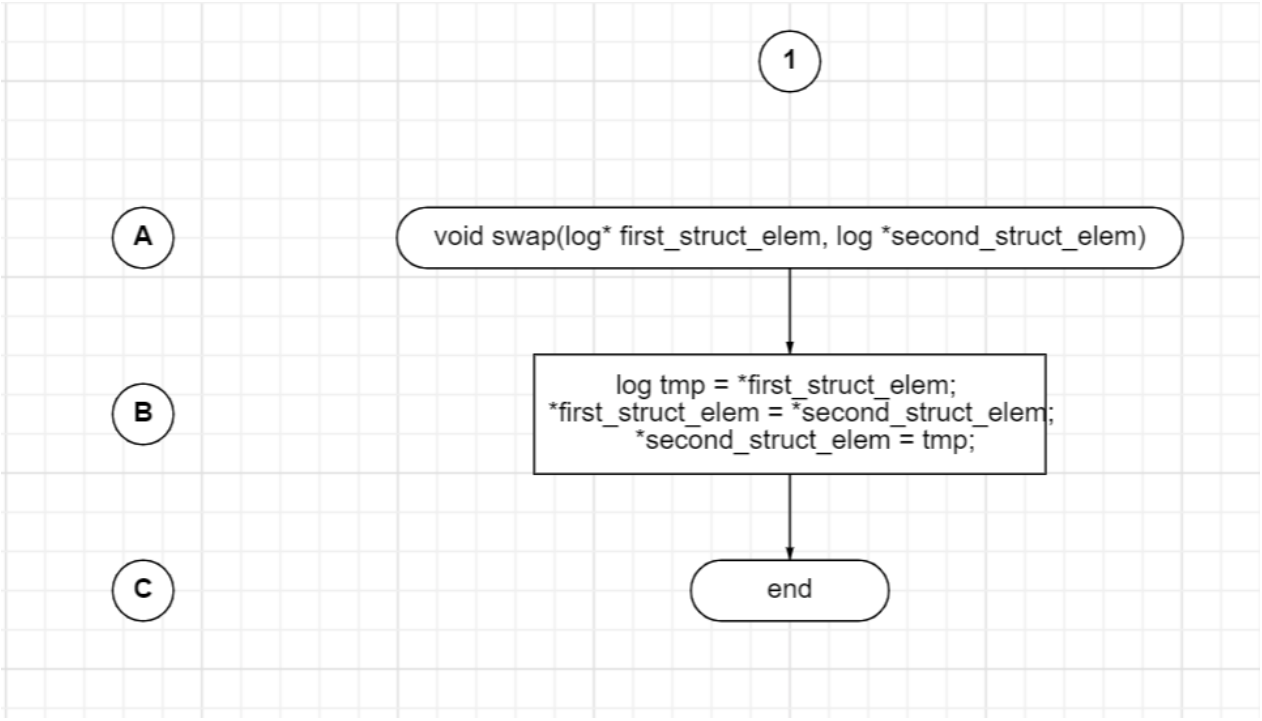

Рис. 1: Блоксхема алгоритма работы функции swap

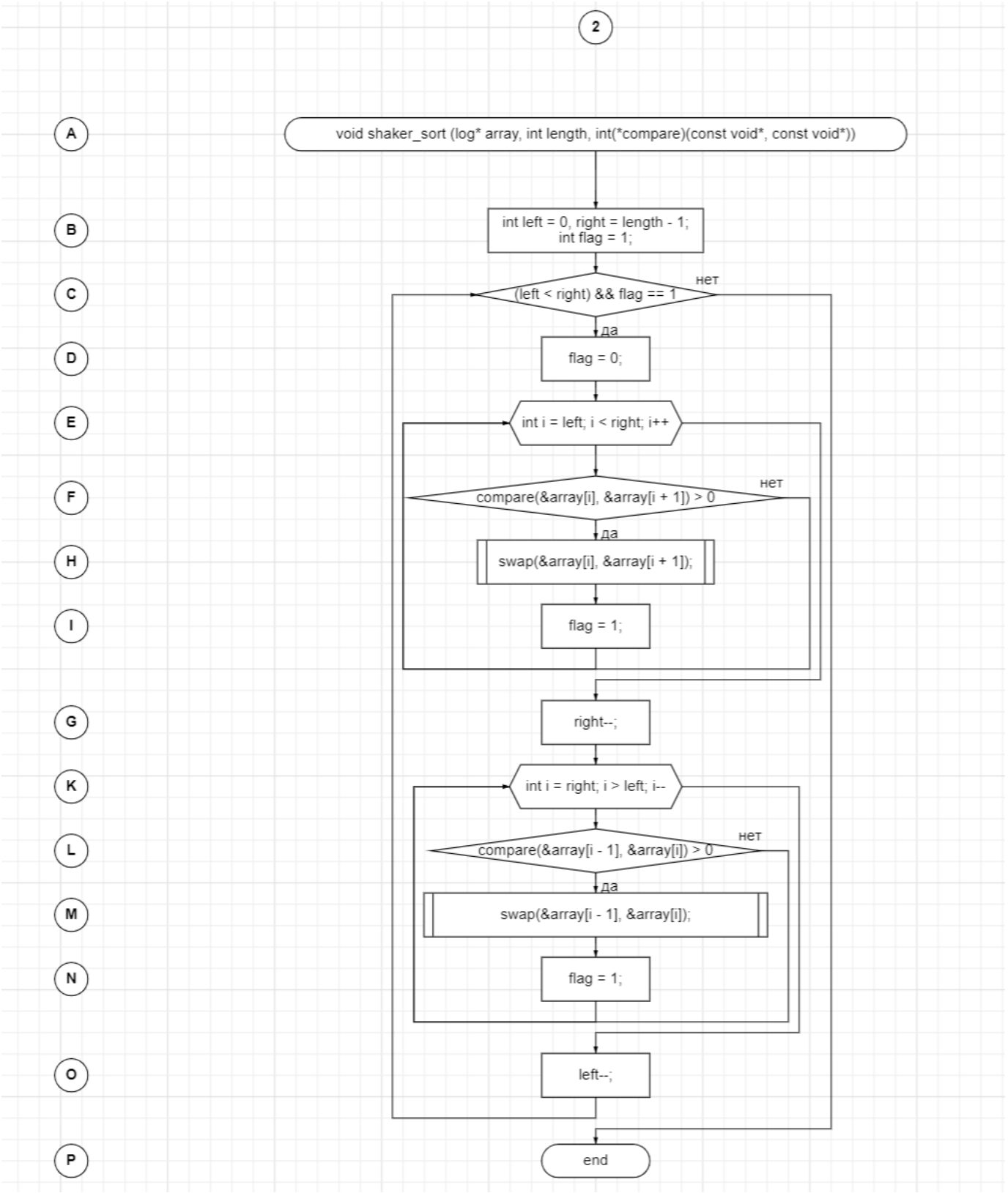

Рис.2: Блоксхема алгоритма работы функции shaker_sort

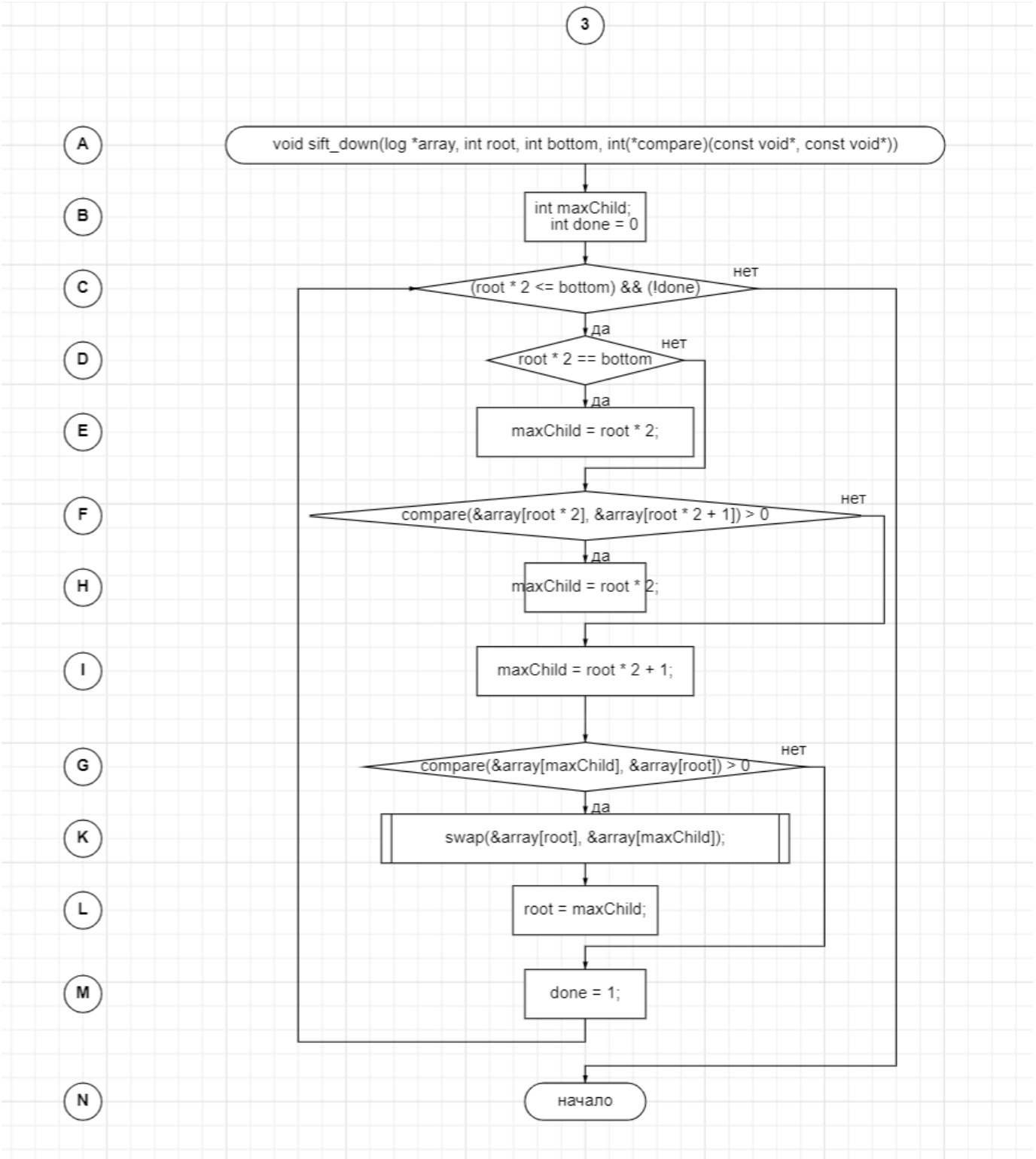

Рис.3: Блоксхема алгоритма работы функции sift_down

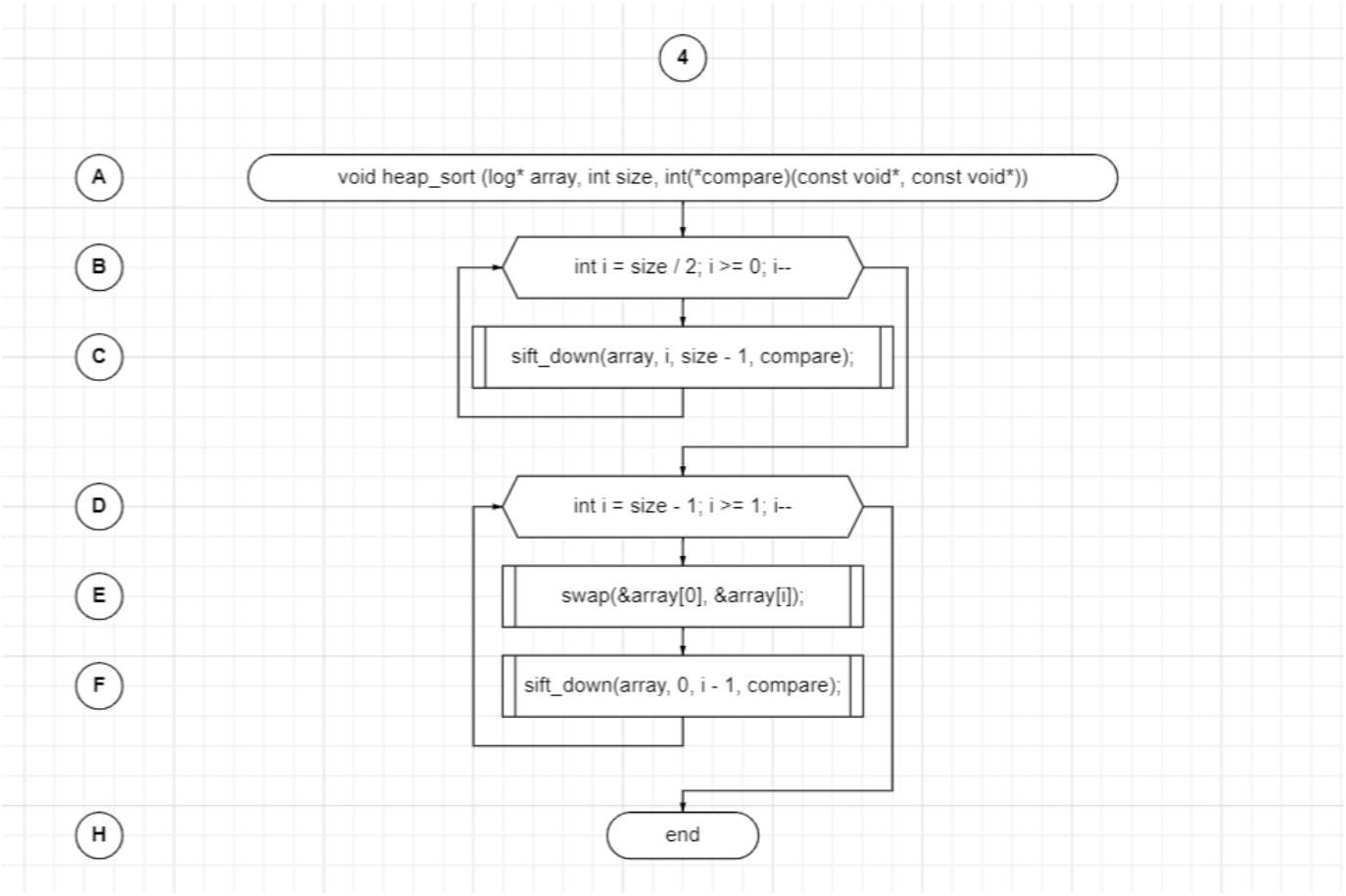

Рис.4: Блоксхема алгоритма работы функции heap_sort

4. Исходные коды разработанных программ

```
#ifndef FUNCS_H
#define FUNCS_H

#include "log.h"

void swap (log*, log*);
void shaker_sort (log*, int, int(*compare)(const void*, const void*));
void sift_down(log*, int, int, int(*compare)(const void*, const void*));
void heap_sort (log*, int, int(*compare)(const void*, const void*));

#endif
```

Листинг 1: исходные кода программы prog (файл: funcs.h)

```c
#include <stdio.h>
#include "funcs.h"
#include "log.h"

void swap (log* first_struct_elem, log* second_struct_elem) {
    log tmp = *first_struct_elem;
    *first_struct_elem = *second_struct_elem;
    *second_struct_elem = tmp;
}

void shaker_sort (log* array, int length, int(*compare)(const void*, const void*)) {
    int left = 0, right = length - 1;
    int flag = 1;
    while ((left < right) && flag == 1) {
        flag = 0;
        for (int i = left; i < right; i++) {
            if (compare(&array[i], &array[i + 1]) > 0) {
                swap(&array[i], &array[i + 1]);
                flag = 1;
            }
        }
        right--;
        for (int i = right; i > left; i--) {
            if (compare(&array[i - 1], &array[i]) > 0) {
                swap(&array[i - 1], &array[i]);
                flag = 1;
            }
        }
        left++;
    }
}

void sift_down(log *array, int root, int bottom, int(*compare)(const void*, const void*)) {
    int maxChild;
    int done = 0;
    while ((root * 2 <= bottom) && (!done)) {
        if (root * 2 == bottom) {
            maxChild = root * 2;
        } else if (compare(&array[root * 2], &array[root * 2 + 1]) > 0) {
            maxChild = root * 2;
        } else {
            maxChild = root * 2 + 1;
        }
        if (compare(&array[maxChild], &array[root]) > 0) {
            swap(&array[root], &array[maxChild]);
            root = maxChild;
        }
        else {
            done = 1;
        }
    }
}

void heap_sort (log* array, int size, int(*compare)(const void*, const void*)) {
    for (int i = size / 2; i >= 0; i--) {
        sift_down(array, i, size - 1, compare);
    }
    for (int i = size - 1; i >= 1; i--) {
        swap(&array[0], &array[i]);
        sift_down(array, 0, i - 1, compare);
    }
}
```

Листинг 2: исходные кода программы prog (файл: funcs.c)

```h
#ifndef LOG_H
#define LOG_H

typedef struct {
        int identifier;
        int importance_level;
        char* string;
} log;

int identifier_comparator (const log*, const log*);
int identifier_comparator_reverse (const log*, const log*);
int importance_level_comparator (const log*, const log*);
int importance_level_comparator_reverse (const log*, const log*);
int string_comparator (const log*, const log*);
int string_comparator_reverse(const log*, const log*);

#endif
```

Листинг 3: исходные кода программы prog (файл: log.h)

```c
#include <stdio.h>
#include <string.h>
#include "log.h"

int identifier_comparator (const log* first_struct_elem, const log* second_struct_elem) {
    return first_struct_elem->identifier - second_struct_elem->identifier;
}

int identifier_comparator_reverse (const log* first_struct_elem, const log* second_struct_elem) {
    return second_struct_elem->identifier - first_struct_elem->identifier;
}

int importance_level_comparator (const log* first_struct_elem, const log* second_struct_elem) {
    return first_struct_elem->importance_level - second_struct_elem->importance_level;
}

int importance_level_comparator_reverse (const log* first_struct_elem, const log* second_struct_elem) {
    return second_struct_elem->importance_level - first_struct_elem->importance_level;
}

int string_comparator (const log* first_struct_elem, const log* second_struct_elem) {
    if (strlen(first_struct_elem->string) == strlen(second_struct_elem->string)){
        return strcmp(first_struct_elem->string, second_struct_elem->string);
    } else {
            if (strlen(first_struct_elem->string) > strlen(second_struct_elem->string)){
                    return 1;
            } else if (strlen(first_struct_elem->string) < strlen(second_struct_elem->string)) {
                    return -1;
            }
    }
}

int string_comparator_reverse (const log* first_struct_elem, const log* second_struct_elem) {
    if (strlen(first_struct_elem->string) == strlen(second_struct_elem->string)) {
        return strcmp(second_struct_elem->string, first_struct_elem->string);
    } else {
            if (strlen(first_struct_elem->string) < strlen(second_struct_elem->string)) {
                    return 1;
            } else if (strlen(first_struct_elem->string) > strlen(second_struct_elem->string)) {
                    return -1;
            }
    }
}
```

Листинг 4: исходные кода программы prog (файл: log.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include "log.h"
#include "funcs.h"

char* my_readline(FILE* file) {
    char buffer[81] = {0};
    char *my_string = NULL;
    int length = 0;
    int element = 0;
    do {
        int flag = 0;
        element = fscanf(file, "%80[^\n]%n", buffer, &flag);
        if (element < 0) {
            if (!my_string) {
                return 0;
            }
        } else if (element > 0) {
            int chunk_len = strlen(buffer);
            int str_len = length + chunk_len;
            my_string = realloc(my_string, str_len + 1);
            memcpy(my_string + length, buffer, chunk_len);
            length = str_len;
            my_string[str_len] = '\0';
        } else {
            fscanf(file, "%*1[\n]");
        }
    } while (element > 0);
    if (length > 0) {
        my_string[length] = '\0';
        return my_string;
    } else {
        my_string = calloc(1, sizeof(char));
        return 0;
    }
}
```

Листинг 5: исходные кода программы prog (файл: main.c)

```c
int main(int argc, char **argv){
    log *stroka;
    int size = 0;
    int flag = 0;
    char* sorting = NULL, *field = NULL, *orientation = NULL, *in = NULL, *out = NULL;
    while((flag = getopt(argc, argv, "QSHLICUDi:o:")) != -1){
        switch(flag){
            case 'Q':
                sorting = "Q";
                break;
            case 'S':
                sorting = "S";
                break;
            case 'H':
                sorting = "H";
                break;
            case 'L':
                field = "L";
                break;
            case 'I':
                field = "I";
                break;
            case 'C':
                field = "C";
                break;
            case 'U':
                orientation = "U";
                break;
            case 'D':
                orientation = "D";
                break;
            case 'i':
                in = optarg;
                break;
            case 'o':
                out = optarg;
                break;
            case '?':
                printf("This Is Wrong Argument.\n");
                return 1;
        }
    }
    if (sorting == NULL || field == NULL || orientation == NULL || in == NULL || out == NULL){
        printf("Put Some Arguments\n");
        return 2;
    }
    FILE *input_data = fopen(in, "r");
    if (!input_data){
        printf("These File Can't Be Open\n");
        return 3;
    }
    FILE *output_data = fopen(out, "w");
    if (!output_data){
        fclose(input_data);
        printf("These File Can't Be Open\n");
        return 3;
    }
```

Листинг 6: исходные кода программы prog (файл: main.c)

```
    stroka = calloc(15001, sizeof(log));
    while(!feof(input_data)){
        fscanf(input_data, "%d", &(stroka[size].identifier));
        fscanf(input_data, "%d", &(stroka[size].importance_level));
        stroka[size].string = my_readline(input_data);
        fscanf(input_data, "%*1[\n]");
        ++size;
    }
    if (sorting == "Q") {
        if (field == "I") {
            if (orientation == "U") {
                qsort(stroka, size, sizeof(log), (int (*)(const void *, const void *)) identifier_comparator);
            }
            else if (orientation == "D") {
                qsort(stroka, size, sizeof(log), (int (*)(const void *, const void *)) identifier_comparator_reverse);
            }
        }
        else if (field == "L") {
            if (orientation == "U") {
                qsort(stroka, size, sizeof(log), (int (*)(const void *, const void *)) importance_level_comparator);
            }
            else if (orientation == "D") {
                qsort(stroka, size, sizeof(log), (int (*)(const void *, const void *)) importance_level_comparator_reverse);
            }
        }
        else if (field == "C") {
            if (orientation == "U") {
                qsort(stroka, size, sizeof(log), (int (*)(const void *, const void *)) string_comparator);
            }
            else if (orientation == "D") {
                qsort(stroka, size, sizeof(log), (int (*)(const void *, const void *)) string_comparator_reverse);
            }
        }
    }
    else if (sorting == "S") {
        if (field == "I") {
            if (orientation == "U") {
                shaker_sort(stroka, size, (int (*)(const void *, const void *)) identifier_comparator);
            }
            else if (orientation == "D") {
                shaker_sort(stroka, size, (int (*)(const void *, const void *)) identifier_comparator_reverse);
            }
        }
        else if (field == "L") {
            if (orientation == "U") {
                shaker_sort(stroka, size, (int (*)(const void *, const void *)) importance_level_comparator);
            }
            else if (orientation == "D") {
                shaker_sort(stroka, size, (int (*)(const void *, const void *)) importance_level_comparator_reverse);
            }
        }
        else if (field == "C") {
            if (orientation == "U") {
                shaker_sort(stroka, size, (int (*)(const void *, const void *)) string_comparator);
            }
            else if (orientation == "D") {
                shaker_sort(stroka, size, (int (*)(const void *, const void *)) string_comparator_reverse);
            }
        }
    }
    else if (sorting == "H") {
        if (field == "I") {
            if (orientation == "U") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) identifier_comparator);
            }
            else if (orientation == "D") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) identifier_comparator_reverse);
            }
        }
        else if (field == "L") {
            if (orientation == "U") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) importance_level_comparator);
            }
            else if (orientation == "D") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) importance_level_comparator_reverse);
            }
        }
        else if (field == "C") {
            if (orientation == "U") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) string_comparator);
            }
            else if (orientation == "D") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) string_comparator_reverse);
            }
        }
```

Листинг 7: исходные кода программы prog (файл: main.c)

```
    for (int i = 0; i < size; ++i) {
        fprintf(output_data, "%d %d%s\n", stroka[i].identifier, stroka[i].importance_level, stroka[i].string);
        fflush(output_data);
        free(stroka[i].string);
    }
    free(stroka);
    fclose(input_data);
    fclose(output_data);
    return 0;
}
```

Листинг 8: исходные кода программы prog (файл: main.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <getopt.h>
#include "log.h"
#include "funcs.h"

int main(int argc, char **argv) {
    int size = 0;
    srand(time(NULL));
    int flag = 0;
    char *sorting = NULL, *field = NULL, *orientation = NULL, *size_of_log = NULL, *number_of_sorting = NULL;
    while ((flag = getopt(argc, argv, "QSHLICUDs:n:")) != -1) {
        switch (flag) {
            case 'Q':
                sorting = "Q";
                break;
            case 'S':
                sorting = "S";
                break;
            case 'H':
                sorting = "H";
                break;
            case 'L':
                field = "L";
                break;
            case 'I':
                field = "I";
                break;
            case 'C':
                field = "C";
                break;
            case 'U':
                orientation = "U";
                break;
            case 'D':
                orientation = "D";
                break;
            case 's':
                size_of_log = optarg;
                break;
            case 'n':
                number_of_sorting = optarg;
                break;
            case '?':
                printf("This Is Wrong Argument.\n");
                return 1;
        }
    }
    if (sorting == NULL || field == NULL || orientation == NULL || size_of_log == NULL || number_of_sorting == NULL) {
        printf("Put Some Arguments\n");
        return 2;
    }
    log *stroka;
    for (int i = 0; i < strlen(size_of_log); i++) {
        size = size * 10 + (size_of_log[i] - '0');
    }
    int sorting_counter = 0;
    for (int i = 0; i < strlen(number_of_sorting); i++) {
        sorting_counter = sorting_counter * 10 + (number_of_sorting[i] - '0');
    }
```

Листинг 9: исходные кода программы progtime (файл: maintiming.c)

```
double time_spent = 0.0;
stroka = calloc(size, sizeof(log));
for (int counter = 0; counter < sorting_counter; counter++) {
    for (int i = 0; i < size; i++) {
        stroka[i].identifier = rand() % 100;
        stroka[i].importance_level = rand() % 5;
        int length = rand() % 100;
        char arr[length + 1];
        for (int j = 0; j < length; j++) {
            arr[j] = rand() % 95 + ' ';
        }
        stroka[i].string = calloc(length + 1, sizeof(char));
        for (int j = 0; j < length; j++) {
            stroka[i].string[j] = arr[j];
        }
    }
    clock_t begin = clock();
    if (sorting == "Q") {
        if (field == "I") {
            if (orientation == "U") {
                qsort(stroka, size, sizeof(log), (int (*)(const void *, const void *)) identifier_comparator);
            } else if (orientation == "D") {
                qsort(stroka, size, sizeof(log),
                      (int (*)(const void *, const void *)) identifier_comparator_reverse);
            }
        } else if (field == "L") {
            if (orientation == "U") {
                qsort(stroka, size, sizeof(log), (int (*)(const void *, const void *)) importance_level_comparator);
            } else if (orientation == "D") {
                qsort(stroka, size, sizeof(log),
                      (int (*)(const void *, const void *)) importance_level_comparator_reverse);
            }
        } else if (field == "C") {
            if (orientation == "U") {
                qsort(stroka, size, sizeof(log), (int (*)(const void *, const void *)) string_comparator);
            } else if (orientation == "D") {
                qsort(stroka, size, sizeof(log), (int (*)(const void *, const void *)) string_comparator_reverse);
            }
        }
    } else if (sorting == "S") {
        if (field == "I") {
            if (orientation == "U") {
                shaker_sort(stroka, size, (int (*)(const void *, const void *)) identifier_comparator);
            } else if (orientation == "D") {
                shaker_sort(stroka, size, (int (*)(const void *, const void *)) identifier_comparator_reverse);
            }
        } else if (field == "L") {
            if (orientation == "U") {
                shaker_sort(stroka, size, (int (*)(const void *, const void *)) importance_level_comparator);
            } else if (orientation == "D") {
                shaker_sort(stroka, size,
                            (int (*)(const void *, const void *)) importance_level_comparator_reverse);
            }
        } else if (field == "C") {
            if (orientation == "U") {
                shaker_sort(stroka, size, (int (*)(const void *, const void *)) string_comparator);
            } else if (orientation == "D") {
                shaker_sort(stroka, size, (int (*)(const void *, const void *)) string_comparator_reverse);
            }
        }
    } else if (sorting == "H") {
        if (field == "I") {
            if (orientation == "U") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) identifier_comparator);
            } else if (orientation == "D") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) identifier_comparator_reverse);
            }
        } else if (field == "L") {
            if (orientation == "U") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) importance_level_comparator);
            } else if (orientation == "D") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) importance_level_comparator_reverse);
            }
        } else if (field == "C") {
            if (orientation == "U") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) string_comparator);
            } else if (orientation == "D") {
                heap_sort(stroka, size, (int (*)(const void *, const void *)) string_comparator_reverse);
            }
        }
    }
    clock_t end = clock();
    time_spent += (double) (end - begin) / CLOCKS_PER_SEC;
```

Листинг 10: исходные кода программы progtime (файл: maintiming.c)

```
        for (int k = 0; k < size; ++k) {
            free(stroka[k].string);
        }
    }
    time_spent /= sorting_counter;
    free(stroka);
    printf("Time Of %s Sorting Of %d structs: %lf seconds\n", sorting, size, time_spent);
    return 0;
}
```

Листинг 11: исходные кода программы progtime (файл: maintiming.c)

5. Описание тестовых примеров и их скриншоты

```
[titov.di@unix:~/inf/lab5]$ cat input_data.txt
1362961854 3 Dco|#q,@6SGg_#8$j5>M]tDR7,wA!IFE*6@-%LJ<|n$\n=^YR|$0nE`&QXGO~nr*%34(_\D]KH7ZU
1533536672 4 q6lsC[9!akW'8Vr+`t>u|{OAYzjm3M^&`Ky%%0&cx]ZU
1179683664 3 ]=rS\eMX5o/.WyA"XDc";e'LhjBDSTV.rFaL--"?|1MQ(lS^.4`Gvdp`L3% h[/X"m%,%0&cAZU
1914053440 2 t5vF%0&cAZU
1654336204 4 xl,V4$;"ae5fez?qS>]XHU\qKqhn-Gc&1p]BqxDP[V4BRS1&nn\7E9)m,o\94A?E/ydF%0&cAZU
1765609673 0 KL\_n/0}3~lo7/)=|5-WKAuhc"bE|SKH!)(o58jH4X5Kg?fbQs:z3-cs/F:,zeT{kYh"qTGcAZU
1530482496 0 Ony6QH'hC7uZU
196845704 2 KC.C&_<r9"q+S6/
1364194741 3 Mjq{1ubQ.XV7["WfB]G^M``?i5VuoGQ:0@7A6yrAOIU+H*ojeC.C&_<rAZU
486459208 3 w(P^:$T*K&EXFY
[titov.di@unix:~/inf/lab5]$ ./prog -S -C -U -i input_data.txt -o output_data.txt
[titov.di@unix:~/inf/lab5]$ cat output_data.txt
1530482496 0 Ony6QH'hC7uZU
1914053440 2 t5vF%0&cAZU
486459208 3 w(P^:$T*K&EXFY
196845704 2 KC.C&_<r9"q+S6/
1533536672 4 q6lsC[9!akW'8Vr+`t>u|{OAYzjm3M^&`Ky%%0&cx]ZU
1364194741 3 Mjq{1ubQ.XV7["WfB]G^M``?i5VuoGQ:0@7A6yrAOIU+H*ojeC.C&_<rAZU
1362961854 3 Dco|#q,@6SGg_#8$j5>M]tDR7,wA!IFE*6@-%LJ<|n$\n=^YR|$0nE`&QXGO~nr*%34(_\D]KH7ZU
1765609673 0 KL\_n/0}3~lo7/)=|5-WKAuhc"bE|SKH!)(o58jH4X5Kg?fbQs:z3-cs/F:,zeT{kYh"qTGcAZU
1179683664 3 ]=rS\eMX5o/.WyA"XDc";e'LhjBDSTV.rFaL--"?|1MQ(lS^.4`Gvdp`L3% h[/X"m%,%0&cAZU
1654336204 4 xl,V4$;"ae5fez?qS>]XHU\qKqhn-Gc&1p]BqxDP[V4BRS1&nn\7E9)m,o\94A?E/ydF%0&cAZU
[titov.di@unix:~/inf/lab5]$ ./prog -H -L -D -i input_data.txt -o output_data.txt
[titov.di@unix:~/inf/lab5]$ cat output_data.txt
1533536672 4 q6lsC[9!akW'8Vr+`t>u|{OAYzjm3M^&`Ky%%0&cx]ZU
1654336204 4 xl,V4$;"ae5fez?qS>]XHU\qKqhn-Gc&1p]BqxDP[V4BRS1&nn\7E9)m,o\94A?E/ydF%0&cAZU
486459208 3 w(P^:$T*K&EXFY
1179683664 3 ]=rS\eMX5o/.WyA"XDc";e'LhjBDSTV.rFaL--"?|1MQ(lS^.4`Gvdp`L3% h[/X"m%,%0&cAZU
1362961854 3 Dco|#q,@6SGg_#8$j5>M]tDR7,wA!IFE*6@-%LJ<|n$\n=^YR|$0nE`&QXGO~nr*%34(_\D]KH7ZU
1364194741 3 Mjq{1ubQ.XV7["WfB]G^M``?i5VuoGQ:0@7A6yrAOIU+H*ojeC.C&_<rAZU
1914053440 2 t5vF%0&cAZU
196845704 2 KC.C&_<r9"q+S6/
1765609673 0 KL\_n/0}3~lo7/)=|5-WKAuhc"bE|SKH!)(o58jH4X5Kg?fbQs:z3-cs/F:,zeT{kYh"qTGcAZU
1530482496 0 Ony6QH'hC7uZU
[titov.di@unix:~/inf/lab5]$ ./prog -Q -I -U -i input_data.txt -o output_data.txt
[titov.di@unix:~/inf/lab5]$ cat output_data.txt
196845704 2 KC.C&_<r9"q+S6/
486459208 3 w(P^:$T*K&EXFY
1179683664 3 ]=rS\eMX5o/.WyA"XDc";e'LhjBDSTV.rFaL--"?|1MQ(lS^.4`Gvdp`L3% h[/X"m%,%0&cAZU
1362961854 3 Dco|#q,@6SGg_#8$j5>M]tDR7,wA!IFE*6@-%LJ<|n$\n=^YR|$0nE`&QXGO~nr*%34(_\D]KH7ZU
1364194741 3 Mjq{1ubQ.XV7["WfB]G^M``?i5VuoGQ:0@7A6yrAOIU+H*ojeC.C&_<rAZU
1530482496 0 Ony6QH'hC7uZU
1533536672 4 q6lsC[9!akW'8Vr+`t>u|{OAYzjm3M^&`Ky%%0&cx]ZU
1654336204 4 xl,V4$;"ae5fez?qS>]XHU\qKqhn-Gc&1p]BqxDP[V4BRS1&nn\7E9)m,o\94A?E/ydF%0&cAZU
1765609673 0 KL\_n/0}3~lo7/)=|5-WKAuhc"bE|SKH!)(o58jH4X5Kg?fbQs:z3-cs/F:,zeT{kYh"qTGcAZU
1914053440 2 t5vF%0&cAZU
```

Рис.5: Примеры работы программы prog

```
[titov.di@unix:~/inf/lab5]$ ./progtime -Q -I -U -s 20000 -n 20
Time Of Q Sorting Of 20000 structs: 0.004307 seconds
[titov.di@unix:~/inf/lab5]$ ./progtime -H -I -U -s 20000 -n 20
Time Of H Sorting Of 20000 structs: 0.007153 seconds
[titov.di@unix:~/inf/lab5]$ ./progtime -S -I -U -s 20000 -n 20
Time Of S Sorting Of 20000 structs: 2.087309 seconds
```

Рис.6: Примеры работы программы progtime

7. Выводы и анализ работы функций

Кол-во структур	Время в секундах qsort
10 000,00	0,002013
15 000,00	0,003098
20 000,00	0,004188
25 000,00	0,005362
30 000,00	0,006553
35 000,00	0,007749
40 000,00	0,008929
45 000,00	0,010328
50 000,00	0,011595
100 000,00	0,024934
200 000,00	0,052241

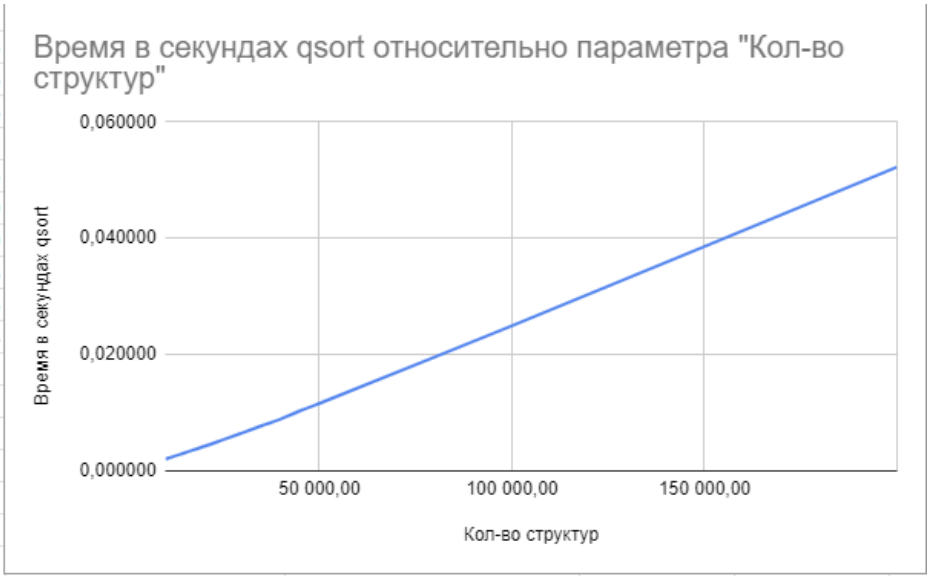

Рис.7: График времени работы qsort от количества сген. Структур усреднение 20 раз

Кол-во структур	Время в секундах shaker_sort
10 000,00	0,523621
15 000,00	1,200288
20 000,00	2,108047
25 000,00	3,309177
30 000,00	4,787485
35 000,00	6,479777
40 000,00	8,521602
45 000,00	10,522466
50 000,00	13,394047

Рис.8: График времени работы shaker_sort от количества сген. Структур усреднение 20 раз

Кол-во структур	Время в секундах heap_sort
10 000,00	0,003515
15 000,00	0,005493
20 000,00	0,007539
25 000,00	0,009626
30 000,00	0,011928
35 000,00	0,013816
40 000,00	0,016414
45 000,00	0,018048
50 000,00	0,021001
100 000,00	0,043105
200 000,00	0,091232

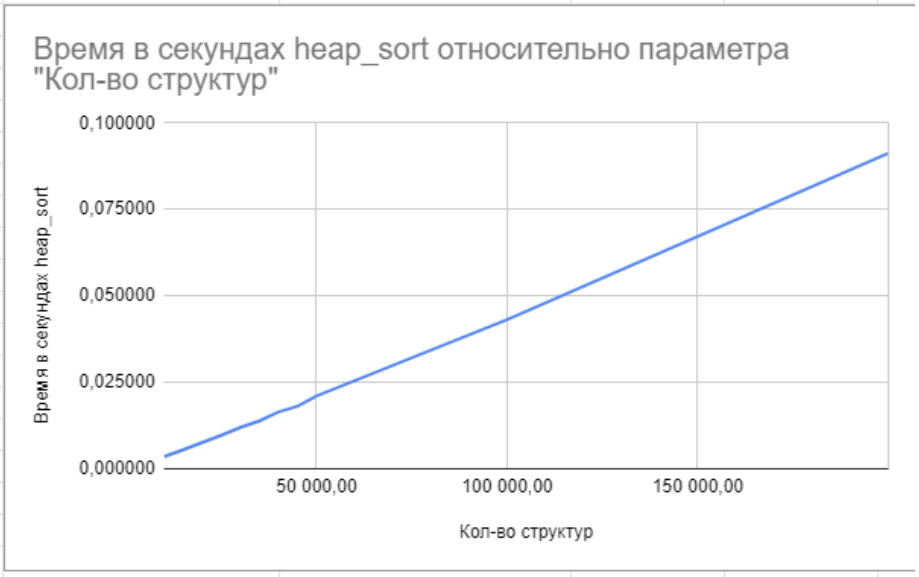

Рис.9: График времени работы heap_sort от количества сген. Структур усреднение 20 раз

Кол-во структур	Время в секундах qsort	Время в секундах heap_sort	Время в секундах shaker_sort
10 000,00	0,002013	0,003515	0,523621
15 000,00	0,003098	0,005493	1,200288
20 000,00	0,004188	0,007539	2,108047
25 000,00	0,005362	0,009626	3,309177
30 000,00	0,006553	0,011928	4,787485
35 000,00	0,007749	0,013816	6,479777
40 000,00	0,008929	0,016414	8,521602
45 000,00	0,010328	0,018048	10,522466
50 000,00	0,011595	0,021001	13,394047
100 000,00	0,024934	0,043105	
200 000,00	0,052241	0,091232	

Рис.10: Графики времени работы всех сортировок от количества сген. Структур усреднение 20 раз

В ходе выполнения данной лабораторной работы были освоены методы сортировок массивов данных. Сортировка shaktr_sort, quick_sort (встроенная в стандартный набор библиотек языка Си, а также heap_sort. С помощью программы progtime удалось выяснить как зависят сложности и времена работ данных сортировок от разного количества обрабатываемых данных, по одному полю с одним направлением. На графиках отчетливо видно квадратичную зависимость shaker_sort, что совпадает с предполагаемыми выводами для этой сортировки, что ее сложность O(n^2), для quick_sort и heap_sort в точках примерно в 50000 видно, что происходит перегиб графика, что свидетельствует о смене темпа роста, такая зависимость свойственна функциям вида n*log(n), что также совпадает с теоретическим предположением. На общем графике видно, как сильно quick_sort и heap_sort выигрывают у квадратичной shaker_sort, до такой степени что даже для 200000 строк данных они кажутся линейными в таком масштабе.