

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:** Мартыненко Д. М.

**Группа:** С22-501

**Преподаватель:** Овчаренко Е. С.

Москва — 2022

# 1. Формулировка индивидуального задания

## Вариант 200:

### Структура данных (студент):

- ФИО (строка произвольной длины).
- Номер группы (строка, формат которой соответствует транслитерированному формату номеров групп в НИЯУ МИФИ).
- Средний балл (дробное число).

### Алгоритмы сортировок:

1. Чётно-нечётная сортировка (Odd-even sort).
2. Двухсторонняя сортировка выбором (Double selection sort).

# 2. Описание использованных типов данных

При выполнении лабораторной работы были использованны следующие типы данных:

1. Структура Student студента по заданию, имеющие следующие поля: `.name` - указатель (`char*`) на имя студента, `.group` - номер (`int`) студента, `.score` - средний балл (`float`) студента.
2. Для удобного хранения данных о массиве была создана структура Students, имеющие следующие поля: `.ptr` - указатель (`Student*`) на массив, `.size` - длина (`int`) массива.
3. Для удобного хранения данных о файлах была создана структура Files, имеющие следующие поля: `.pathIn` - указатель (`char*`) на строку исходного файла, `.pathOut` - указатель (`char*`) на строку итогового файла.
4. Для удобного хранения данных о была создана структура RandParams, имеющие следующие поля: `.tries` - количество (`int`) потворений сортировки, `.size` - размер (`int`) тестируемого массива.

# 3. Описание алгоритмов сортировок

## 3.1. Чётно-нечётная сортировка

Алгоритм сортировки построен на другой сортировке - пузырьковой. В отличие от него независимо сравниваются элементы под чётными и нечётными индексами с последующими элементами независимо. В начале итерации флаг, определяющий отсортирован ли массив, ставится как истинна, а далее каждый нечётный сравнивается с последующими, и если необходимо поменять местами, то в флаг устанавливается ложь. Аналогично для чётных. Алгоритм прекращается, когда флаг остаётся истинным. Сложность сортировки:

- худшая -  $O(n^2)$ .
- средняя -  $O(n^2)$ .
- лучшая -  $O(n)$ .

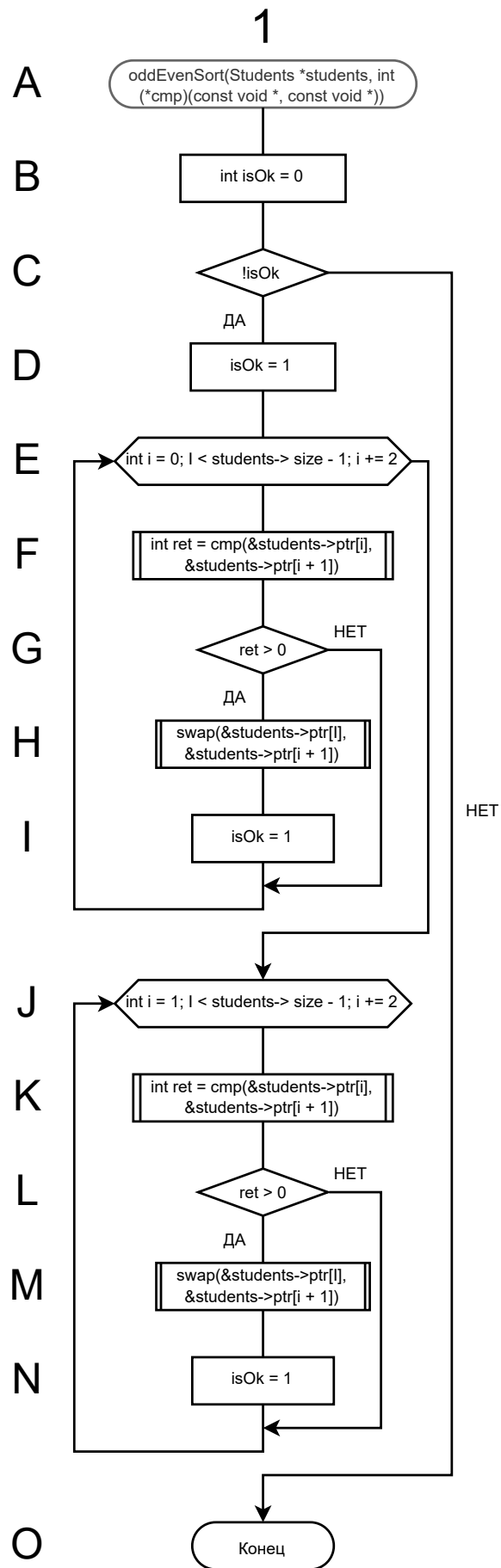


Рис. 1: Блок-схема алгоритма работы функции `oddEvenSort()`

### 3.2. Двухсторонняя сортировка выбором

Алгоритм сортировки построен на другой сортировке - сортировкой выбором. Помимо максимального элемента массива будет находиться и минимальный. Максимум будет перемещаться в конец подмассива, а минимум соответственно в начало. Сложность сортировки:

- худшая -  $O(n^2/2)$ .
- средняя -  $O(n^2/2)$ .
- лучшая -  $O(n^2/2)$ .

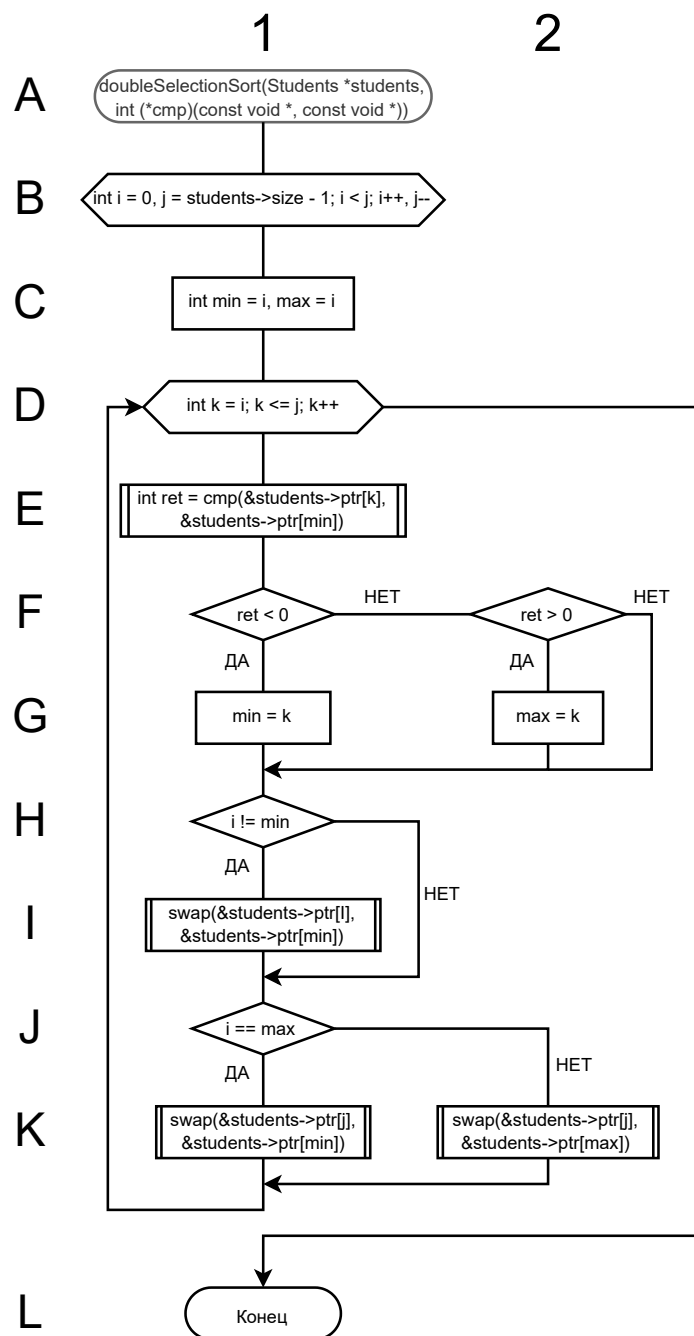


Рис. 2: Блок-схема алгоритма работы функции `doubleSelectionSort()`

## 4. Исходные коды разработанных программ

### 4.1. Исходный код программы для работы с файлами

Исходный код 1: библиотека parser.c

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <getopt.h>
4
5  #include "include/args.h"
6  #include "include/files.h"
7  #include "include/sorter.h"
8  #include "include/utils.h"
9
10 void printFileArgsHelp();
11 int parseFileArgs(int argc, char *argv[], Files *files);
12
13 int main(int argc, char *argv[]) {
14     int algorithm = QSORT, field = NAME, direction = INCREASE;
15
16     int ret = parseArgs(argc, argv, &algorithm, &field, &direction);
17     if (ret == -1) {
18         ret = 0;
19         printFileArgsHelp();
20     } else if (!ret) {
21         Files files = {.pathIn = NULL, .pathOut = NULL};
22         ret = parseFileArgs(argc, argv, &files);
23         if (!ret) {
24             FILE *fIn = openFile(files.pathIn, "r");
25             if (fIn) {
26                 Students students = {.ptr = NULL, .size = 0};
27                 scanFile(fIn, &students);
28                 if (students.size) {
29                     printSuccessMsg("Successfully scanned %d lines!\n", students.size);
30                     sortStudents(&students, algorithm, field, direction);
31                     FILE *fOut = openFile(files.pathOut, "w");
32                     if (fOut) {
33                         writeFile(fOut, &students);
34                         if (!ferror(fOut)) {
35                             printSuccessMsg("Successfully written %d lines!\n",
36                                             students.size);
37                         }
38                         freeStudents(&students);
39                     }
40                     fclose(fOut);
41                 }
42             }
43             fclose(fIn);
44         }
45         freeCharPtr(2, files.pathIn, files.pathOut);
46     }
47
48     return ret;
49 }
50
51 void printFileArgsHelp() {
52     printf("Usage: parser [arguments] [input] [output]\n");
53     printArgsHelp();
54 }
```

```

55
56 int parseFileArgs(int argc, char *argv[], Files *files) {
57     int ret = 1, args;
58
59     args = argc - optind;
60     if (args > 2) {
61         printf("Too many arguments, need two!\n");
62     } else if (args < 2) {
63         printf("Not enough arguments, need two!\n");
64     } else {
65         files->pathIn = strdup(argv[optind]);
66         files->pathOut = strdup(argv[optind + 1]);
67         if (files->pathOut && files->pathIn) ret = 0;
68     }
69     if (ret) printFileArgsHelp();
70
71     return ret;
72 }

```

## 4.2. Исходный код программы с таймированием

Исходный код 2: библиотека timing.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <getopt.h>
4  #include <time.h>
5
6  #include "include/args.h"
7  #include "include/objects.h"
8  #include "include/randomizer.h"
9  #include "include/sorter.h"
10
11 void printFileArgsHelp();
12 int parseTimingArgs(int argc, char *argv[], RandParams *params);
13
14 int main(int argc, char *argv[]) {
15     int algorithm = QSORT, field = NAME, direction = INCREASE;
16
17     int ret = parseArgs(argc, argv, &algorithm, &field, &direction);
18     if (ret == -1) {
19         ret = 0;
20         printFileArgsHelp();
21     } else if (!ret) {
22         RandParams params = {.tries = 0, .size = 0};
23         ret = parseTimingArgs(argc, argv, &params);
24         if (!ret) {
25             srand(time(NULL));
26             size_t fullTime = 0;
27             for (int i = 0; i < params.tries; i++) {
28                 Students students = getRandomStudents(params.size);
29                 fullTime += sortStudents(&students, algorithm, field, direction);
30                 freeStudents(&students);
31             }
32             printf("Average time to sort array of %d elements: %Lf\n", params.size,
33                 (long double) fullTime / CLOCKS_PER_SEC / params.tries);
34         }
35     }
36
37     return ret;
38 }

```

```

39
40 void printFileArgsHelp() {
41     printf("Usage: timing [arguments] [input] [output]\n");
42     printArgsHelp();
43 }
44
45 int parseTimingArgs(int argc, char *argv[], RandParams *params) {
46     int ret = 1, args;
47
48     args = argc - optind;
49     if (args > 2) {
50         printf("Too many arguments, need two!\n");
51     } else if (args < 2) {
52         printf("Not enough arguments, need two!\n");
53     } else {
54         sscanf(argv[optind], "%d", &params->tries);
55         sscanf(argv[optind + 1], "%d", &params->size);
56         if (params->tries && params->size) ret = 0;
57     }
58     if (ret) printFileArgsHelp();
59
60     return ret;
61 }

```

### 4.3. Исходный код обработки аргументов

Исходный код 3: библиотека args.c.

```

1 #include <stdio.h>
2 #include <getopt.h>
3 #include <string.h>
4
5 #include "include/args.h"
6 #include "include/utils.h"
7
8 int parseArgs(int argc, char *argv[], int *a, int *f, int *d) {
9     int arg;
10
11     while ((arg = getopt(argc, argv, ":ha:f:d:")) != -1) {
12         switch (arg) {
13             case 'a':
14                 if (!strcmp(optarg, "qsort")) *a = QSORT;
15                 else if (!strcmp(optarg, "odd-even")) *a = ODD_EVEN_SORT;
16                 else if (!strcmp(optarg, "double-selection")) {
17                     *a = DOUBLE_SELECTION_SORT;
18                 }
19                 else {
20                     printErrorMsg("Algorithm %s is not supported, supported: "
21                                   "qsort, odd-even, double-selection\n", optarg);
22                     return 1;
23                 }
24                 break;
25             case 'f':
26                 if (!strcmp(optarg, "name")) *f = NAME;
27                 else if (!strcmp(optarg, "group")) *f = GROUP;
28                 else if (!strcmp(optarg, "score")) *f = SCORE;
29                 else {
30                     printErrorMsg("Field %s doesn't exists,"
31                                   "exists: name, group, score\n", optarg);
32                     return 1;
33                 }
34             }
35         }
36     }
37 }

```

```

34         break;
35     case 'd':
36         if (!strcmp(optarg, "increase")) *d = INCREASE;
37         else if (!strcmp(optarg, "decrease")) *d = DECREASE;
38         else {
39             printErrorMsg("Option %s isn't available,"
40                           "available: increase, decrease\n", optarg);
41             return 1;
42         }
43         break;
44     case 'h':
45         return -1;
46     case '?':
47         printErrorMsg("Unknown option argument: -%c\n", optopt);
48     default:
49         break;
50 }
51 }
52
53 return 0;
54 }
55
56 void printArgsHelp() {
57     printf("Arguments:\n");
58     printf("    -a <algorithm>\t\tUse <algorithm> to sort array (default: qsort)\n");
59     printf("    -f <field>\t\t\tSort array by <field> (default: name)\n");
60     printf("    -d <direction>\t\tSort array by <direction> (default: increase)\n");
61     printf("    -h\t\t\t\tPrint help (this message) and exit\n");
62 }

```

Исходный код 4: заголовочный файл include/args.h.

```

1  #ifndef INCLUDE_ARGS_H
2  #define INCLUDE_ARGS_H
3
4  enum Algorithm {
5      QSORT = 0,
6      ODD_EVEN_SORT = 1,
7      DOUBLE_SELECTION_SORT = 2
8  };
9
10 enum Field {
11     NAME = 0,
12     GROUP = 1,
13     SCORE = 2
14 };
15
16 enum Direction {
17     INCREASE = 0,
18     DECREASE = 1
19 };
20
21 int parseArgs(int argc, char *argv[], int *a, int *f, int *d);
22 void printArgsHelp();
23
24 #endif //INCLUDE_ARGS_H

```

## 4.4. Исходный код работы с файлами

Исходный код 5: библиотека files.c.

```

1  #include <stdio.h>

```



```

2  #include <string.h>
3  #include <errno.h>
4  #include <stdlib.h>
5  #include <math.h>
6
7  #include "include/files.h"
8  #include "include/utils.h"
9
10 FILE *openFile(char *path, char *mode) {
11     FILE *f = fopen(path, mode);
12     if (!f) {
13         printErrorMsg("Failed to open file: %s\n", strerror(errno));
14     }
15     return f;
16 }
17
18 void scanFile(FILE *f, Students *students) {
19     int ret;
20     do {
21         Student student = { .name = NULL };
22         do {
23             char *buf = malloc((128 + 1) * sizeof(char));
24             ret = fscanf(f, "%128[^,]", buf);
25             if (ret < 0) {
26                 if (student.name) {
27                     printErrorMsg("File format incorrect, use csv format!\n");
28                 } else ret = 0;
29             } else if (ret > 0) {
30                 size_t oldSize = getLength(student.name);
31                 size_t newSize = oldSize + getLength(buf) + 1;
32                 student.name = realloc(student.name, newSize * sizeof(char));
33                 appendString(student.name, oldSize, buf);
34             } else {
35                 if (student.name) ret = 1;
36                 ret += fscanf(f, ", %d, %f\n", &student.group, &student.score);
37                 if (ret == 3) {
38                     appendStudents(students, student);
39                 } else {
40                     if (!feof(f)) {
41                         printErrorMsg("File format incorrect, use csv format!\n");
42                     } else if (ferror(f)) {
43                         printErrorMsg("Error occurred while reading file!\n");
44                     } else if (!student.name) {
45                         printErrorMsg("Student name cannot be empty, aborting!\n");
46                     } else {
47                         printErrorMsg("Unknown error occurred, aborting!\n");
48                     }
49                     ret = -1;
50                 }
51             }
52             free(buf);
53         } while (ret == 1);
54         if (ret == -1) freeStudents(students);
55     } while (ret == 3);
56 }
57
58 void writeFile(FILE *f, Students *students) {
59     for (int i = 0; i < students->size; i++) {
60         Student student = students->ptr[i];
61         fprintf(f, "%s, %d, %f\n", student.name, student.group, fabsf(student.score));

```

```

62         if (ferror(f)) {
63             printErrorMsg("Error occurred while writing to file!\n");
64             break;
65         }
66     }
67 }

```

Исходный код 6: заголовочный файл include/files.h.

```

1  #ifndef INCLUDE_FILES_H
2  #define INCLUDE_FILES_H
3
4  #include "objects.h"
5
6  FILE *openFile(char *path, char *mode);
7  void scanFile(FILE *f, Students *students);
8  void writeFile(FILE *f, Students *students);
9
10 #endif //INCLUDE_FILES_H

```

## 4.5. Исходный код структур

Исходный код 7: библиотека objects.c.

```

1  #include <stdlib.h>
2
3  #include "include/objects.h"
4
5  void appendStudents(Students *students, Student student) {
6      if (!students) return;
7      students->size++;
8      students->ptr = realloc(students->ptr, sizeof(Student) * students->size);
9      students->ptr[students->size - 1] = student;
10 }
11
12 void freeStudents(Students *students) {
13     if (!students) return;
14     for (int i = 0; i < students->size; i++) {
15         free(students->ptr[i].name);
16     }
17     free(students->ptr);
18     students->size = 0;
19 }

```

Исходный код 8: заголовочный файл include/objects.h.

```

1  #ifndef INCLUDE_OBJECTS_H
2  #define INCLUDE_OBJECTS_H
3
4  typedef struct Student {
5      char *name;
6      int group;
7      float score;
8  } Student;
9
10 typedef struct Students {
11     Student *ptr;
12     int size;
13 } Students;
14
15 typedef struct Files {
16     char *pathIn;

```

```

17     char *pathOut;
18 } Files;
19
20 typedef struct RandParams {
21     int tries;
22     int size;
23 } RandParams;
24
25 void appendStudents(Students *students, Student student);
26 void freeStudents(Students *students);
27
28 #endif //INCLUDE_OBJECTS_H

```

## 4.6. Исходный код сортировок

Исходный код 9: библиотека `sorter.c`.

```

1  #include <string.h>
2  #include <stdlib.h>
3  #include <printf.h>
4  #include <time.h>
5
6  #include "include/args.h"
7  #include "include/sorter.h"
8
9  size_t sortStudents(Students *students, int a, int f, int d) {
10     int (*cmp)(const void *, const void *) = NULL;
11     size_t time = clock();
12
13     switch (f) {
14         case GROUP:
15             if (d == INCREASE) cmp = groupComparator;
16             else cmp = reversedGroupComparator;
17             break;
18         case SCORE:
19             if (d == INCREASE) cmp = scoreComparator;
20             else cmp = reversedScoreComparator;
21             break;
22         default:
23             if (d == INCREASE) cmp = nameComparator;
24             else cmp = reversedNameComparator;
25     }
26     switch (a) {
27         case ODD_EVEN_SORT:
28             oddEvenSort(students, cmp);
29             break;
30         case DOUBLE_SELECTION_SORT:
31             doubleSelectionSort(students, cmp);
32             break;
33         default:
34             qsort(students->ptr, students->size, sizeof(Student), cmp);
35     }
36
37     return clock() - time;
38 }
39
40 void oddEvenSort(Students *students, int (*cmp)(const void *, const void *)) {
41     int isOk = 0;
42
43     while (!isOk) {
44         isOk = 1;

```

```

45     for (int i = 0; i < students->size - 1; i += 2) {
46         if (cmp(&students->ptr[i], &students->ptr[i + 1]) > 0) {
47             swap(&students->ptr[i], &students->ptr[i + 1]);
48             isOk = 0;
49         }
50     }
51     for (int i = 1; i < students->size - 1; i += 2) {
52         if (cmp(&students->ptr[i], &students->ptr[i + 1]) > 0) {
53             swap(&students->ptr[i], &students->ptr[i + 1]);
54             isOk = 0;
55         }
56     }
57 }
58 }
59
60 void doubleSelectionSort(Students *students, int (*cmp)(const void *, const void *)) {
61     for (int i = 0, j = students->size - 1; i < j; i++, j--) {
62         int min = i, max = i;
63         for (int k = i; k <= j; k++) {
64             if (cmp(&students->ptr[k], &students->ptr[min]) < 0) {
65                 min = k;
66             } else if (cmp(&students->ptr[k], &students->ptr[max]) > 0) {
67                 max = k;
68             }
69         }
70
71         if (i != min) {
72             swap(&students->ptr[i], &students->ptr[min]);
73         }
74
75         if (i == max) {
76             swap(&students->ptr[j], &students->ptr[min]);
77         } else {
78             swap(&students->ptr[j], &students->ptr[max]);
79         }
80     }
81 }
82
83 void swap(Student *obj1, Student *obj2) {
84     Student tmp = *obj1;
85     *obj1 = *obj2;
86     *obj2 = tmp;
87 }
88
89 int nameComparator(const void *obj1, const void *obj2) {
90     return strcmp(((Student *) obj1)->name, ((Student *) obj2)->name);
91 }
92
93 int reversedNameComparator(const void *obj1, const void *obj2) {
94     return nameComparator(obj1, obj2) * -1;
95 }
96
97 int groupComparator(const void *obj1, const void *obj2) {
98     return (((Student *) obj1)->group - ((Student *) obj2)->group);
99 }
100
101 int reversedGroupComparator(const void *obj1, const void *obj2) {
102     return groupComparator(obj1, obj2) * -1;
103 }
104

```

```

105 int scoreComparator(const void *obj1, const void *obj2) {
106     float diff = ((Student *) obj1)->score - ((Student *) obj2)->score;
107     return !diff ? 0 : diff > 0 ? 1 : -1;
108 }
109
110 int reversedScoreComparator(const void *obj1, const void *obj2) {
111     return scoreComparator(obj1, obj2) * -1;
112 }

```

Исходный код 10: заголовочный файл include/sorter.h.

```

1 #ifndef INCLUDE_SORTER_H
2 #define INCLUDE_SORTER_H
3
4 #include "objects.h"
5
6 size_t sortStudents(Students *students, int a, int f, int d);
7 void oddEvenSort(Students *students, int (*cmp)(const void *, const void *));
8 void doubleSelectionSort(Students *students, int (*cmp)(const void *, const void *));
9 void swap(Student *obj1, Student *obj2);
10 int nameComparator(const void *obj1, const void *obj2);
11 int reversedNameComparator(const void *obj1, const void *obj2);
12 int groupComparator(const void *obj1, const void *obj2);
13 int reversedGroupComparator(const void *obj1, const void *obj2);
14 int scoreComparator(const void *obj1, const void *obj2);
15 int reversedScoreComparator(const void *obj1, const void *obj2);
16
17 #endif //INCLUDE_SORTER_H

```

## 4.7. Исходный код рандомайзера

Исходный код 11: библиотека randomizer.c.

```

1 #include <stdlib.h>
2 #include <string.h>
3 #include <printf.h>
4
5 #include "include/randomizer.h"
6
7 Students getRandomStudents(int size) {
8     Students students = { .size = 0, .ptr = NULL};
9     for (int i = 0; i < size; i++) {
10         appendStudents(&students, getRandomStudent());
11     }
12     return students;
13 }
14
15 Student getRandomStudent() {
16     Student student = {
17         .name = getRandomName(),
18         .group = getRandomInt(100, 1000),
19         .score = getRandomFloat(10.0f)
20     };
21     return student;
22 }
23
24 int getRandomInt(int from, int to) {
25     return from + rand() % (to - from + 1);
26 }
27
28 float getRandomFloat(float to) {

```

```

29     return ((float) rand() / (float) (RAND_MAX)) * to;
30 }
31
32 char *getRandomName() {
33     char *ret = calloc(1, sizeof(char));
34     for (int i = 0; i < 3; i++) {
35         char *str = getRandomString(getRandomInt(6, 12), i == 2 ? 0 : 1);
36         ret = realloc(ret, strlen(ret) + strlen(str) + 1);
37         strcat(ret, str);
38         free(str);
39     }
40     return ret;
41 }
42
43 char *getRandomString(int size, int afterSpace) {
44     char *ret = malloc((size + afterSpace + 1) * sizeof(char));
45     ret[0] = (char) getRandomInt(65, 90);
46     for (int i = 1; i < size; i++) {
47         ret[i] = (char) getRandomInt(97, 122);
48     }
49     for (int i = size; i < size + afterSpace; ++i) {
50         ret[i] = ' ';
51     }
52     ret[size + afterSpace] = '\0';
53     return ret;
54 }

```

Исходный код 12: заголовочный файл include/randomizer.h.

```

1 #ifndef INCLUDE_RANDOMIZER_H
2 #define INCLUDE_RANDOMIZER_H
3
4 #include "objects.h"
5
6 Students getRandomStudents(int size);
7 Student getRandomStudent();
8 int getRandomInt(int from, int to);
9 float getRandomFloat(float to);
10 char *getRandomName();
11 char *getRandomString(int size, int afterSpace);
12
13 #endif //INCLUDE_RANDOMIZER_H

```

## 4.8. Исходный код вспомогательной библиотеки

Исходный код 13: библиотека utils.c.

```

1 #include <stdarg.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 #include "include/utils.h"
6
7 void printErrorMsg(const char *msg, ...) {
8     va_list args;
9     va_start(args, msg);
10
11     printf("\033[1;31m");
12     printMsg(msg, args);
13     printf("\033[0m");
14 }

```

```

15
16 void printSuccessMsg(const char *msg, ...) {
17     va_list args;
18     va_start(args, msg);
19
20     printf("\033[1;32m");
21     printMsg(msg, args);
22     printf("\033[0m");
23 }
24
25 void printMsg(const char *msg, va_list args) {
26     while (*msg != '\0') {
27         if (*msg != '%') {
28             putchar(*msg);
29         } else {
30             msg++;
31             if (*msg != '\0') {
32                 switch (*msg) {
33                     case 's':
34                         printf("%s", va_arg(args, char*));
35                         break;
36                     case 'c':
37                         printf("%c", va_arg(args, int));
38                         break;
39                     case 'd':
40                         printf("%d", va_arg(args, int));
41                     default:
42                         break;
43                 }
44             }
45             msg++;
46         }
47     }
48 }
49
50 void freeCharPtr(int count, ...) {
51     va_list args;
52     va_start(args, count);
53
54     for (int i = 0; i < count; i++) {
55         free(va_arg(args, char*));
56     }
57 }
58
59 int getLength(char *str) {
60     int count = 0;
61     if (str) {
62         while (str[count] != '\0') count++;
63     }
64     return count;
65 }
66
67 void appendString(char *src, int srcSize, const char *dest) {
68     int end = 0, j = -1;
69     for (int i = 0; i < srcSize; i++) {
70         if (src[i] == '\0') {
71             end = i;
72             break;
73         }
74     }

```

```

75     do {
76         j++;
77         src[end + j] = dest[j];
78     } while (dest[j] != '\0');
79 }

```

Исходный код 14: заголовочный файл include/Utils.h.

```

1  #ifndef INCLUDE_UTILS_H
2  #define INCLUDE_UTILS_H
3
4  void printErrorMsg(const char *, ...);
5  void printSuccessMsg(const char *msg, ...);
6  void printMsg(const char *msg, va_list args);
7  void freeCharPtr(int count, ...);
8  int getLength(char *str);
9  void appendString(char *src, int srcSize, const char *dest);
10
11 #endif //INCLUDE_UTILS_H

```

## 5. Анализ результатов таймирования

На графике представлено среднее время сортировок в зависимости от количества сортируемых элементов. Для наиболее точного подсчёта среднего времени количество повторений сортировок было 30 раз. Такое среднее время находилось для каждого поля отдельно, а затем вычислялось среднее время. Так было подсчитано среднее время для всех сортировок от 50 до 10000 элементов с шагом в 10 элементов.

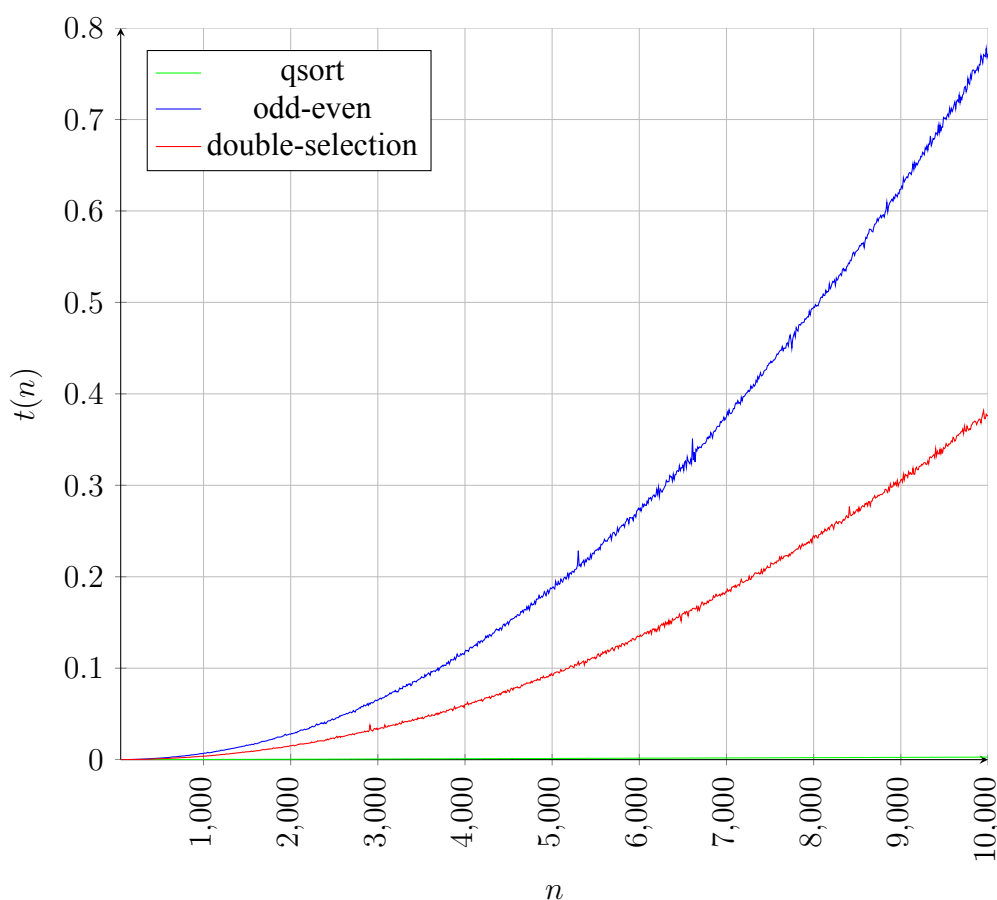


Рис. 3: Общий график сортировок.



Для анализа результата графика приведём среднюю сложность алгоритмов сортировок:

- odd-even -  $O(n^2)$ .
- double-selection -  $O(n^2/2)$ .
- qsort -  $O(n * \log n)$ .

Как видно, сортировки odd-even и double-selection имеют параболический вид, а double-selection имеет результаты времени в два раза меньшие чем у odd-even, что и понятно из приведённого списка средней сложности выше. Из-за низкой сложности алгоритма qsort на графике в сравнении с остальными сортировки его результаты стремятся к нулю.

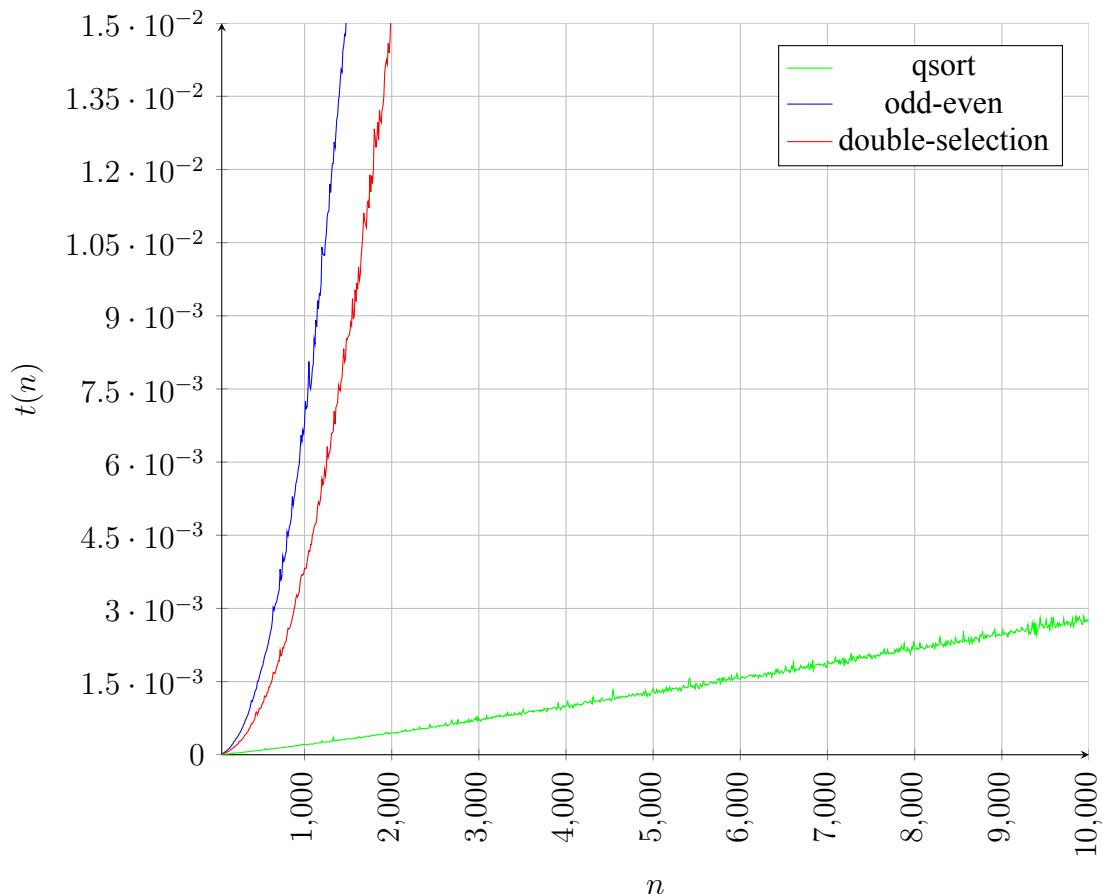


Рис. 4: Увеличенный график сортировок.

На увеличенном графике можно увидеть насколько скорость qsort отличается от скорости остальных сортировок. Таким образом можно сделать вывод, что из тестируемых сортировок самая быстрая - qsort, а медленная - odd-even.

## 6. Описание тестовых примеров

Все тесты будут производить на примере данного входного файла:

Исходный код 15: файл `input.csv`.

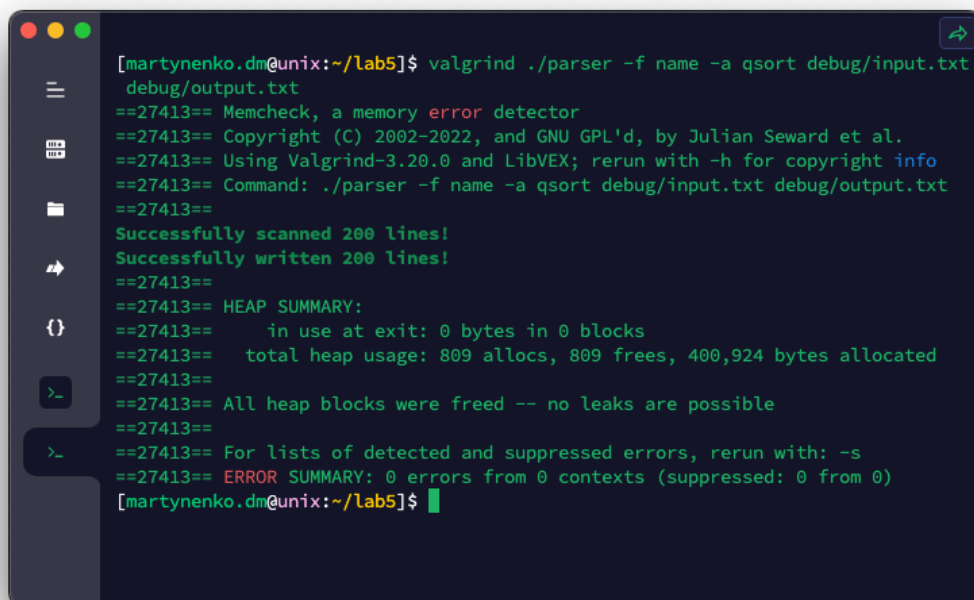
```
1 Oles Artur Konstantinov, 10, 1.2347
2 Yevgeni Rustik Shvets, 83, 5.3836
3 Maxim Onufriy Shwetz, 54, 8.2346
4 Avhust Evgeni Kovalchuk, 85, 3.2834
5 Maksim Filat Sokolovsky, 42, 5.2384
```

Таблица 1: Тестовые примеры

Сортировка	Поле	Реверс	Итоговый файл
qsort	name	Нет	Avhust Evgeni Kovalchuk, 85, 3.283400 Maksim Filat Sokolovsky, 42, 5.238400 Maxim Onufriy Shwetz, 54, 8.234600 Oles Artur Konstantinov, 10, 1.234700 Yevgeni Rustik Shvets, 83, 5.383600
odd-even	score	Да	Maxim Onufriy Shwetz, 54, 8.234600 Yevgeni Rustik Shvets, 83, 5.383600 Maksim Filat Sokolovsky, 42, 5.238400 Avhust Evgeni Kovalchuk, 85, 3.283400 Oles Artur Konstantinov, 10, 1.234700
double-selection	group	Нет	Oles Artur Konstantinov, 10, 1.234700 Maksim Filat Sokolovsky, 42, 5.238400 Maxim Onufriy Shwetz, 54, 8.234600 Yevgeni Rustik Shvets, 83, 5.383600 Avhust Evgeni Kovalchuk, 85, 3.283400

## 7. Скриншоты

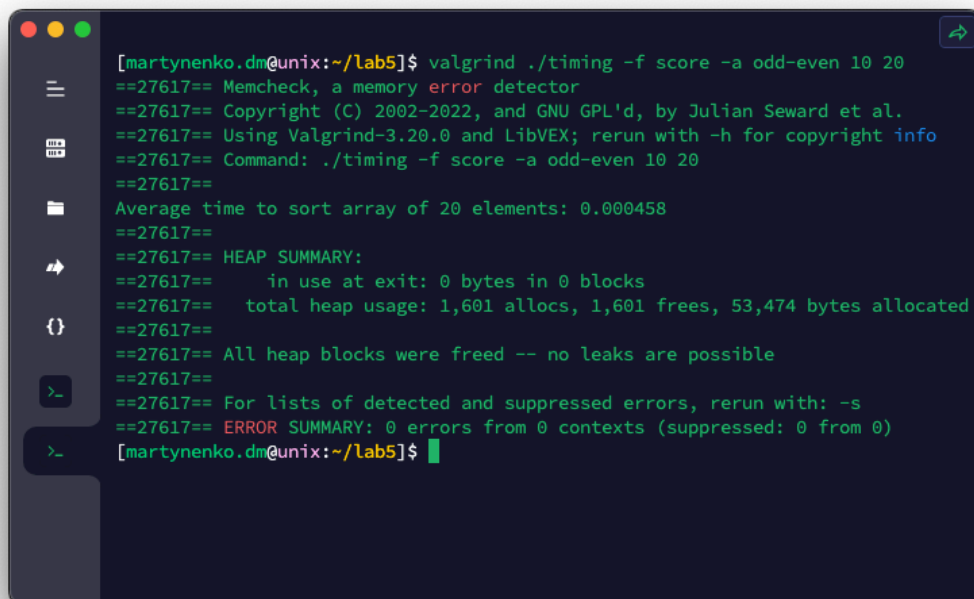
### 7.1. Программа работающая с файлами



```
[martynenko.dm@unix:~/lab5]$ valgrind ./parser -f name -a qsort debug/input.txt
debug/output.txt
==27413== Memcheck, a memory error detector
==27413== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==27413== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==27413== Command: ./parser -f name -a qsort debug/input.txt debug/output.txt
==27413==
Successfully scanned 200 lines!
Successfully written 200 lines!
==27413==
==27413== HEAP SUMMARY:
==27413==    in use at exit: 0 bytes in 0 blocks
==27413==   total heap usage: 809 allocs, 809 frees, 400,924 bytes allocated
==27413==
==27413== All heap blocks were freed -- no leaks are possible
==27413==
==27413== For lists of detected and suppressed errors, rerun with: -s
==27413== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[martynenko.dm@unix:~/lab5]$
```

Рис. 5: Успешная проверка использования памяти при помощи valgrind.

## 7.2. Программа для таймирования



```
[martynenko.dm@unix:~/lab5]$ valgrind ./timing -f score -a odd-even 10 20
==27617== Memcheck, a memory error detector
==27617== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==27617== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==27617== Command: ./timing -f score -a odd-even 10 20
==27617==
Average time to sort array of 20 elements: 0.000458
==27617==
==27617== HEAP SUMMARY:
==27617==     in use at exit: 0 bytes in 0 blocks
==27617==   total heap usage: 1,601 allocs, 1,601 frees, 53,474 bytes allocated
==27617==
==27617== All heap blocks were freed -- no leaks are possible
==27617==
==27617== For lists of detected and suppressed errors, rerun with: -s
==27617== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[martynenko.dm@unix:~/lab5]$
```

Рис. 6: Успешная проверка использования памяти при помощи valgrind.

## 8. Выводы

В ходе выполнения данной работы были:

1. Изучена и организована работа с файлами при помощи библиотеки `stdio.h`.
2. Изучены и реализованы некоторые виды сортировок.
3. Организован генерация случайных строк и чисел при помощи функции `rand()`.