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

ОТЧЕТ

О выполнении лабораторной работы №5
«Работа с массивами.
Исследование методов сортировки массивов»

Студент: Баранов А. Т.
Группа: Б22-534
Преподаватель: Широких Т. А.

Москва — 2022

1. Формулировка индивидуального задания

Вариант №179

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 сортировки

1. Гномья сортировка (Gnome sort).
2. Сортировка Шелла (Shell sort).

2. Описание использованных типов данных

При выполнении данной лабораторной работы использовался встроенный тип данных `int`, пред- назначенный для работы с целыми числами, встроенный тип данных char, предназначенный для работы с символами, и встроенный тип данных double, предназначенный для работы с числами с плавающей точкой двойной точности.

3. Описание использованного алгоритма

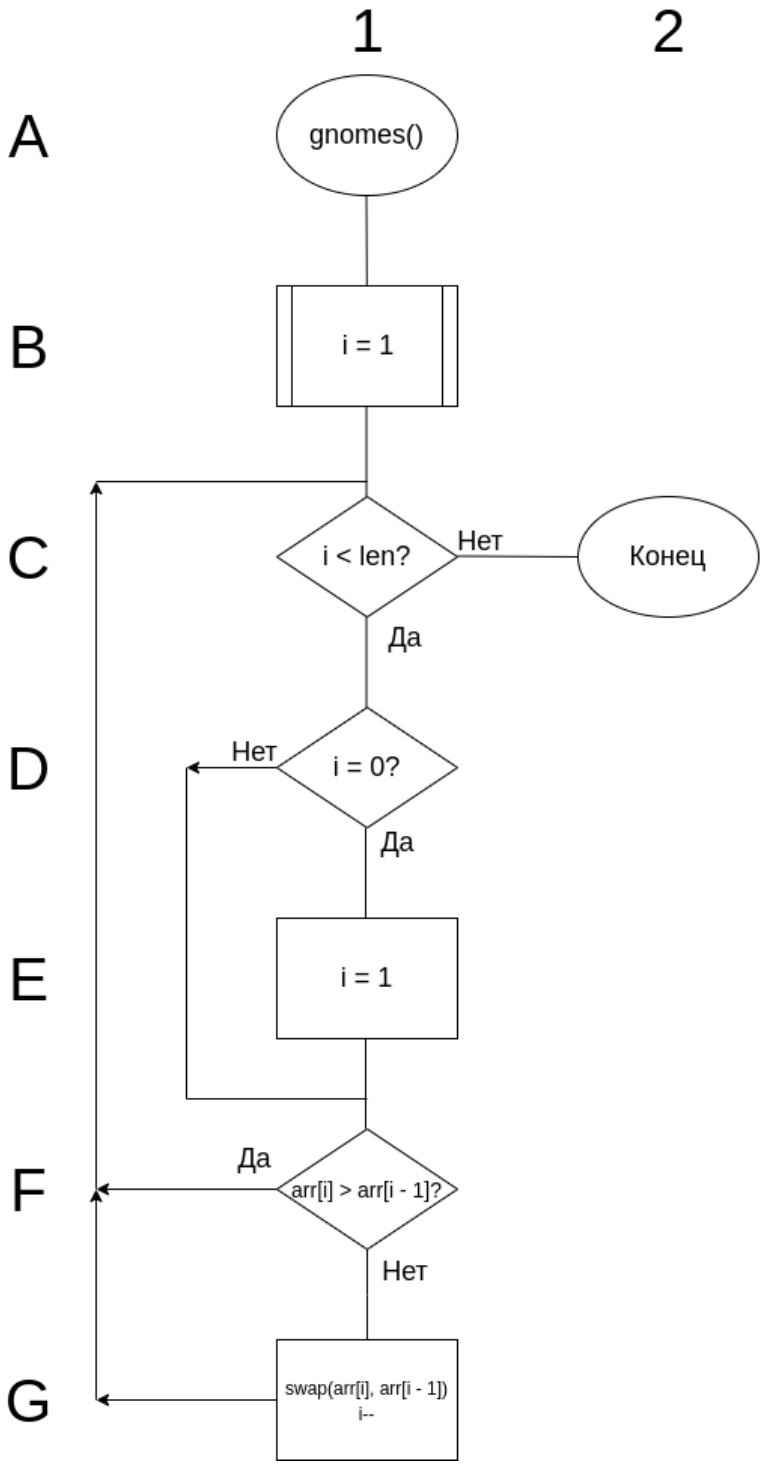

Рис. 1: Блок- схема алгоритма работы функции `gnomes()`

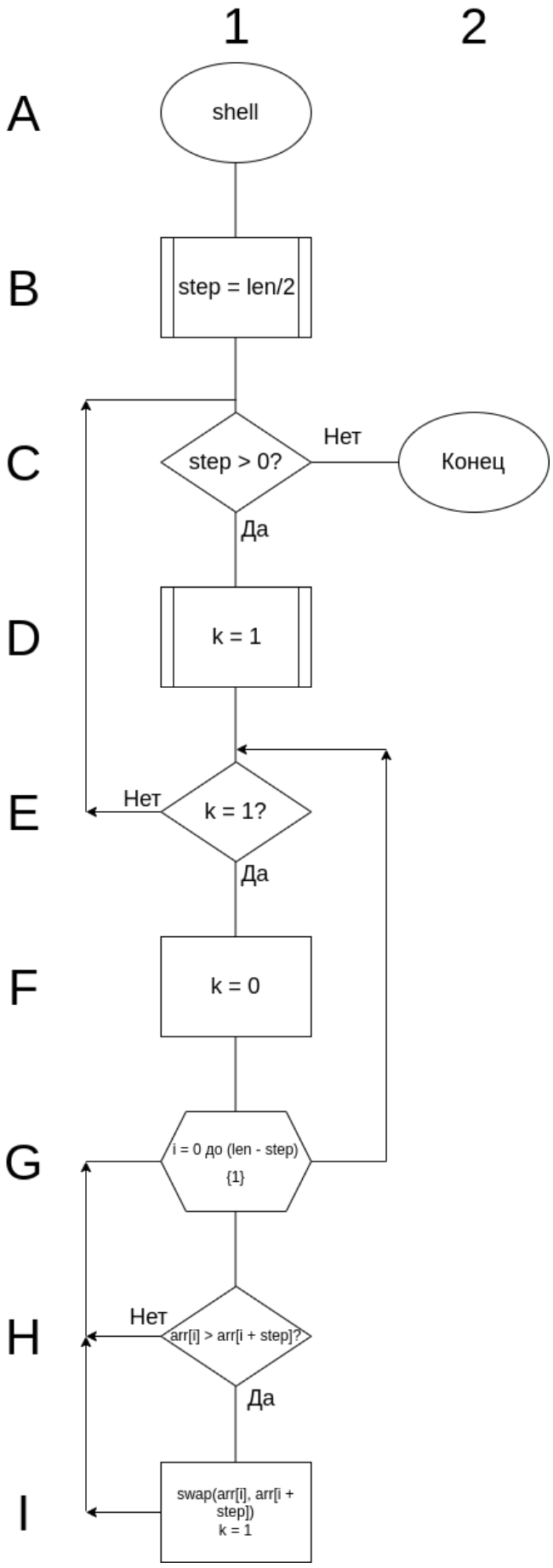

Рис. 2: Блок- схема алгоритма работы функции `shell()`

4. Исходные коды разработанных программ

Листинг 1: Исходные коды программы `prog1` (файл: `main.c`)

```
1.  #include <stdlib.h>
2.  #include "options.h"
3.  #include "met.h"
4.  #include "sorts.h"
5.
6.  int main (int argc, char **argv)
7.  {
8.      char *sort = "qsort";
9.      char *id = "score";
10.     char *in_file, *out_file;
11.     unsigned short direction = 1;
12.
13.     if (options(argc, argv, &sort, &id, &in_file, &out_file, &direction) == 1)
                goto A;
14.
15.     student *data;
16.     unsigned len = 0;
17.
18.     if(data_reading(in_file, &data, &len) == 1) goto A;
19.     if (sorting(data, len, sort, id, direction)) goto A;
20.     if(data_writing(out_file, data, len) == 1) goto A;
21.
22.     for(unsigned i = 0; i < len; i++)
23.         free(data[i].FIO);
24.     free(data);
25.
26.     return 0;
27.
28.     A:
29.     for(unsigned i; i < len; i++)
30.         free(data[i].FIO);
31.     free(data);
32.     return 1;
33. }
```

Листинг 2: Исходные коды программы `prog1` (файл: `met.h`)

```
1.  #ifndef PROG_MET_H
2.  #define PROG_MET_H
3.
4.  typedef struct
5.  {
6.      char *FIO;
7.      char group[8];
8.      double score;
9.  } student;
10.
11. int data_reading(char *in_file, student **data, unsigned *len);
12. int data_writing(char *out_file, student *data, unsigned len);
13.
14. #endif //PROG_MET_H
```

Листинг 3: Исходные коды программы `prog1` (файл: `met.c`)

```
1. #include "met.h"
2. #include <stdio.h>
3. #include <string.h>
4. #include <stdlib.h>
5. int data_reading(char *in_file, student **data, unsigned *len)
6. {
7.     FILE *in = fopen(in_file, "r");
8.     if(in == NULL)
9.     {
10.        fprintf(stderr, "Failed to open file %s\n", in_file);
11.        return 1;
12.    }
13.    char buf[8];
14.    fscanf(in, "%s%*c", buf);
15.    *len = atoi(buf);
16.    student *arr = malloc(*len * sizeof(student));
17.    if (NULL == arr)
18.    {
19.        fprintf(stderr,"Failed to allocate memory\n");
20.        return 1;
21.    }
```

```
22.     for(unsigned i = 0; i < *len; i++)
23.     {
24.         arr[i].FIO = calloc(50, sizeof(student));
25.         if (NULL == arr)
26.         {
27.             fprintf(stderr,"Failed to allocate memory\n");
28.             return 1;
29.         }
30.         fscanf(in,"%s %s %lf", arr[i].FIO, arr[i].group, &arr[i].score);
31.     }
32.     if (fclose(in) == EOF)
33.     {
34.         fprintf(stderr, "Failed to close file %s\n", in_file);
35.         return 1;
36.     }
37.     *data = arr;
38.     return 0;
39. }
40.
41. int data_writing(char *out_file, student *data, unsigned len)
42. {
43.     FILE *out = fopen(out_file, "w");
44.
45.     if (NULL == out)
46.     {
47.         fprintf(stderr, "Failed to open file %s\n", out_file);
48.         return 1;
49.     }
50.
51.     for(unsigned i = 0; i < len; i++)
52.         fprintf(out, "%s %s %lf\n", data[i].FIO, data[i].group, data[i].score);
53.
54.     if (fclose(out) == EOF)
55.     {
56.         fprintf(stderr, "Failed to close file %s\n", out_file);
57.         return 1;
58.     }
59.     return 0;
60. }
```

Листинг 4: Исходные коды программы `prog1` (файл: `options.h`)

```
1. #ifndef PROG_OPTIONS_H
2. #define PROG_OPTIONS_H
3.
4. void help();
5. void err_usage();
6. short int options(int argc, char *argv[], char **sort, char **id,
7.                    char **in_file, char **out_file, unsigned short int *direction);
8.
9. #endif //PROG_OPTIONS_H
```

Листинг 5: Исходные коды программы `prog1` (файл: `options.c`)

```
1.  #include "options.h"
2.  #include <stdio.h>
3.  #include <getopt.h>
4.
5.  void help()
6.  {
7.      printf("\nOPTIONS:\n");
8.      printf("-h prints this message again.\n");
9.      printf("-s changes sorting algorithm to Gnome sort (\"gnomes\" argument)
10.         or Shell sort (\"shell\" argument).\nQuick Sort is used as default.\n");
11.     printf("-p changes sort field to by name (\"name\" argument), by
                group(\"group\" argument).
12.         \nBy average score is used as default.\n");
13.     printf("-d changes sorting direction to descending.\nAscending order is
            used as default.\n\n");
14. }
15.
16. void err_usage()
17. {
18.     fprintf(stderr, "Programme requires two arguments: input and output files.
                        Use -h to read all options.\n");
19. }
20.
```

```
21. short int options(int argc, char *argv[], char **sort, char **id,
22.                           char **in_file, char **out_file, unsigned short int *direction)
23. {
24.     int c;
25.     while ((c = getopt (argc, argv, "hs:p:d")) != -1)
26.         switch (c)
27.         {
28.             case 'h':
29.                 help();
30.                 return 0;
31.             case 's':
32.                 *sort = optarg;
33.                 break;
34.             case 'p':
35.                 *id = optarg;
36.                 break;
37.             case 'd':
38.                 *direction = -1;
39.                 break;
40.             case '?':
41.                 if (optopt != 's' && optopt != 'p')
42.                 {
43.                     fprintf(stderr, "Unknown option '-%c'.\n", optopt);
44.                     return 1;
45.                 }
46.             default:
47.                 fprintf(stderr ,"Unknown error\n");
48.                 return 1;
49.         }
50.     if (optind <= argc - 2)
51.     {
52.         *in_file = argv[optind++];
53.         *out_file = argv[optind++];
54.     }
55.     else
56.     {
57.         err_usage();
58.         return 1;
59.     }
60.
```

```
61.     while (optind < argc)
62.         printf ("Excess argument %s\n", argv[optind++]);
63.
64.     return 0;
65. }
```

Листинг 6: Исходные коды программы `prog1` (файл: `sorts.h`)

```
1.  #ifndef PROG_SORTS_H
2.  #define PROG_SORTS_H
3.
4.  #include "met.h"
5.
6.  int sorting(student *array, unsigned len, char *sort, char *id, unsigned short
                direction);
7.
8.  int cmp_score(const student *p1, const student *p2);
9.  int cmp_group(const student *p1, const student *p2);
10. int cmp_name(const student *p1, const student *p2);
11.
12. int cmp_score_des(const student *p1, const student *p2);
13. int cmp_group_des(const student *p1, const student *p2);
14. int cmp_name_des(const student *p1, const student *p2);
15.
16. void gnomes(student *arr, unsigned int len, int (*cmp)(const student *, const
                student *));
17. void shell(student *data, unsigned int len, int (*cmp)(const student *, const
                student *));
18.
19. #endif //PROG_SORTS_H
```

Листинг 7: Исходные коды программы `prog1` (файл: `sorts.c`)

```
1.  #include "sorts.h"
2.  #include "stddef.h"
3.  #include <string.h>
4.  #include <stdlib.h>
5.  #include <stdio.h>
6.
7.  int sorting(student *array, unsigned len, char *sort, char *id, unsigned short
                direction)
8.  {
9.      if(direction == 1)
10.     {
11.         if (!strcmp(sort, "qsort"))
12.         {
13.             if(!strcmp(id, "score"))
14.                 qsort(array, len, sizeof(student),
15.                     (int (*) (const void *, const void *)) cmp_score);
16.             else if (!strcmp(id, "name"))
17.                 qsort(array, len, sizeof(student),
18.                     (int (*) (const void *, const void *)) cmp_name);
19.             else if (!strcmp(id, "group"))
20.                 qsort(array, len, sizeof(student),
21.                     (int (*) (const void *, const void *)) cmp_group);
22.             else
23.             {
24.                 printf("Wrong sorting field\n");
25.                 return 1;
26.             }
27.         }
28.         else if(!strcmp(sort, "gnomes"))
29.         {
30.             if(!strcmp(id, "score"))
31.                 gnomes(array, len, cmp_score);
32.             else if (!strcmp(id, "name"))
33.                 gnomes(array, len, cmp_name);
34.             else if (!strcmp(id, "group"))
35.                 gnomes(array, len, cmp_group);
36.             else
37.             {
38.                 printf("Wrong sorting field\n");
```

```
            return 1;
        }
    }
    else if(!strcmp(sort, "shell"))
    {
        if(!strcmp(id, "score"))
            shell(array, len, cmp_score);
        else if (!strcmp(id, "name"))
            shell(array, len, cmp_name);
        else if (!strcmp(id, "group"))
            shell(array, len, cmp_group);
        else
        {
            printf("Wrong sorting field\n");
            return 1;
        }
    }
    else
    {
        printf("Wrong sorting algorithm");
        return 1;
    }
}
else
{
    if (!strcmp(sort, "qsort"))
    {
        if(!strcmp(id, "score"))
            qsort(array, len, sizeof(student),
                (int (*) (const void *, const void *)) cmp_score_des);
        else if (!strcmp(id, "name"))
            qsort(array, len, sizeof(student),
                (int (*) (const void *, const void *)) cmp_name_des);
        else if (!strcmp(id, "group"))
            qsort(array, len, sizeof(student),
                (int (*) (const void *, const void *)) cmp_group_des);
        else
        {
            printf("Wrong sorting field\n");
            return 1;
```

```
79.             }
80.         }
81.         else if(!strcmp(sort, "gnomes"))
82.         {
83.             if(!strcmp(id, "score"))
84.                 gnomes(array, len, cmp_score_des);
85.             else if (!strcmp(id, "name"))
86.                 gnomes(array, len, cmp_name_des);
87.             else if (!strcmp(id, "group"))
88.                 gnomes(array, len, cmp_group_des);
89.             else
90.             {
91.                 printf("Wrong sorting field\n");
92.                 return 1;
93.             }
94.         }
95.         else if(!strcmp(sort, "shell"))
96.         {
97.             if(!strcmp(id, "score"))
98.                 shell(array, len, cmp_score_des);
99.             else if (!strcmp(id, "name"))
100.                shell(array, len, cmp_name_des);
101.            else if (!strcmp(id, "group"))
102.                shell(array, len, cmp_group_des);
103.            else
104.            {
105.                printf("Wrong sorting field\n");
106.                return 1;
107.            }
108.        }
109.        else
110.        {
111.            printf("Wrong sorting algorithm");
112.            return 1;
113.        }
114.    }
115.    return 0;
116. }
117.
```

```
118. int cmp_score(const student *p1, const student *p2)
119. {
120.     return ((p1->score - p2->score) < 0)? -1:1;
121. }
122. int cmp_group(const student *p1, const student *p2)
123. {
124.     return (strcmp(p1->group, p2->group) < 0)? -1:1;
125. }
126. int cmp_name(const student *p1, const student *p2)
127. {
128.     return (strcmp(p1->FIO, p2->FIO) < 0)? -1:1;
129. }
130.
131. int cmp_score_des(const student *p1, const student *p2)
132. {
133.     return ((p1->score - p2->score) < 0)? 1:-1;
134. }
135. int cmp_group_des(const student *p1, const student *p2)
136. {
137.     return (strcmp(p1->group, p2->group) < 0)? 1:-1;
138. }
139. int cmp_name_des(const student *p1, const student *p2)
140. {
141.     return (strcmp(p1->FIO, p2->FIO) < 0)? 1:-1;
142. }
143.
144. void gnomes(student *data, unsigned int len, int (*cmp)(const student *,
                                    const student *))
145. {
146.     size_t i = 1;
147.
148.     while (i < len)
149.     {
150.         if (i == 0)
151.             i = 1;
152.         if (cmp(data + i, data + i - 1) == 1)
153.             ++i;
154.         else
155.         {
156.             student tmp = data[i];
```

```
157.            data[i] = data[i-1];
158.            data[i-1] = tmp;
159.            --i;
160.        }
161.    }
162. }
163.
164. void shell(student *data, unsigned int len, int (*cmp)(const student *, const
                        student *))
165. {
166.    unsigned step = len/2;
167.    while (step > 0)
168.    {
169.        unsigned short int k = 1;
170.        while(k)
171.        {
172.            k = 0;
173.            for(unsigned i = 0; i < len - step; i++)
174.            {
175.                if(cmp(data + i, data + i + step) == 1)
176.                {
177.                    student tmp = data[i];
178.                    data[i] = data[i + step];
179.                    data[i + step] = tmp;
180.                    k = 1;
181.                }
182.            }
183.        }
184.        step /= 2;
185.    }
186. }
```

Листинг 8: Исходные коды программы `prog2` (файл: main.c)

```
1.  #include <stdlib.h>
2.  #include <time.h>
3.  #include <stdio.h>
4.  #include "options.h"
5.  #include "gen.h"
6.  #include "sorts.h"
7.
8.  int main (int argc, char **argv)
9.  {
10.     char *sort = "qsort";
11.     unsigned num_arrays, len;
12.
13.     if (options(argc, argv, &sort, &len, &num_arrays) == 1) goto A;
14.
15.     int *data;
16.     long double time = 0;
17.     for(unsigned i = 1; i <= num_arrays; i++)
18.     {
19.         if (generating(&data, len) == 1) goto A;
20.         clock_t start = clock();
21.         if (sorting(data, len, sort) == 1) goto A;
22.         clock_t end = clock();
23.         time += (long double) (end - start) / CLOCKS_PER_SEC;
24.         free(data);
25.     }
26.     printf("%Lf", time/num_arrays);
27.     return 0;
28.
29.     A:
30.     free(data);
31.     return 1;
32. }
```

Листинг 9: Исходные коды программы `prog2` (файл: `g e n .h`)

```
1. #ifndef PROG_MET_H
2. #define PROG_MET_H
3.
4. int generating(int **array, unsigned len);
5.
6. #endif //PROG_MET_H
```

Листинг 10: Исходные коды программы `prog2` (файл: `g e n .c`)

```
1.  #include "gen.h"
2.  #include <time.h>
3.  #include <stdio.h>
4.  #include <string.h>
5.  #include <stdlib.h>
6.
7.  int generating(int **array, unsigned len)
8.  {
9.      srand(time(NULL));
10.     int *tmp = malloc(len * sizeof (int));
11.     if (NULL == tmp)
12.     {
13.         fprintf(stderr, "Failed to allocate memory\n");
14.         return 1;
15.     }
16.     for(unsigned i = 0; i < len; i++)
17.         tmp[i] = rand();
18.     *array = tmp;
19.     return 0;
20. }
```

Листинг 11: Исходные коды программы `prog2` (файл: `o p t i o n .h`)

```
1. #ifndef PROG_OPTIONS_H
2. #define PROG_OPTIONS_H
3. void help();
4. void err_usage();
5. short int options(int argc, char *argv[], char **sort, unsigned *num1,
                     unsigned *num2);
6. #endif //PROG_OPTIONS_H
```

Листинг 12: Исходные коды программы `prog2` (файл: `option.c`)

```
1.  #include "options.h"
2.  #include <stdio.h>
3.  #include <getopt.h>
4.  #include <stdlib.h>
5.
6.  void help()
7.  {
8.      printf("\nOPTIONS:\n");
9.      printf("-h prints this message again.\n");
10.     printf("-s changes sorting algorithm to Gnome sort (\"gnomes\" argument)
11.             or Shell sort (\"shell\" argument).\nQuick Sort is used as default.\n");
12. }
13.
14. void err_usage()
15. {
16.     fprintf(stderr, "Programme requires two arguments: number of arrays and
of elements.\nUse -h to read all options.\n");
17. }
18.
19. short int options(int argc, char *argv[], char **sort, unsigned *num1,
unsigned *num2)
20. {
21.     int c;
22.     while ((c = getopt (argc, argv, "hs:")) != -1)
23.         switch (c)
24.         {
25.             case 'h':
26.                 help();
27.                 return 0;
28.             case 's':
29.                 *sort = optarg;
30.                 break;
31.             case '?':
32.                 if (optopt != 's')
33.                 {
34.                     fprintf(stderr, "Unknown option '-%c'.\n", optopt);
35.                     return 1;
36.                 }
```

```
37.             default:
38.                 fprintf(stderr ,"Unknown error\n");
39.                 return 1;
40.         }
41.     if (optind <= argc - 2)
42.     {
43.         *num1 = atoi(argv[optind++]);
44.         *num2 = atoi(argv[optind++]);
45.         if (*num1 == 0 || *num2 == 0)
46.         {
47.             printf("Invalid required parametrs\n");
48.             return 1;
49.         }
50.     }
51.     else
52.     {
53.         err_usage();
54.         return 1;
55.     }
56.
57.     while (optind < argc)
58.         printf ("Excess argument %s\n", argv[optind++]);
59.
60.     return 0;
61. }
```

Листинг 13: Исходные коды программы `prog2` (файл: `sorts.h`)

```
1.  #ifndef PROG_SORTS_H
2.  #define PROG_SORTS_H
3.
4.  #include "gen.h"
5.
6.  int sorting(int *array, unsigned len, char *sort);
7.
8.  int cmp_(const int *p1, const int *p2);
9.
10. void gnomes(int *arr, unsigned int len, int (*cmp)(const int *, const int *));
11. void shell(int *data, unsigned int len, int (*cmp)(const int *, const int *));
12.
13. #endif //PROG_SORTS_H
```

Листинг 14: Исходные коды программы `prog2` (файл: `sorts.c`)

```
1.  #include "sorts.h"
2.  #include "stddef.h"
3.  #include <string.h>
4.  #include <stdlib.h>
5.  #include <stdio.h>
6.
7.  int sorting(int *array, unsigned len, char *sort)
8.  {
9.      if (!strcmp(sort, "qsort"))
10.         qsort(array, len, sizeof(int),
11.                 (int (*) (const void *, const void *)) cmp_);
12.     else if(!strcmp(sort, "gnomes"))
13.         gnomes(array, len, cmp_);
14.     else if(!strcmp(sort, "shell"))
15.         shell(array, len, cmp_);
16.     else
17.     {
18.         printf("Wrong sorting algorithm");
19.         return 1;
20.     }
21.
22.     return 0;
23. }
24.
25. int cmp_(const int *p1, const int *p2)
26. {
27.     return ((*p1 - *p2) < 0)? -1:1;
28. }
29.
30. void gnomes(int *data, unsigned int len, int (*cmp)(const int *, const int *))
31. {
32.     size_t i = 1;
33.     while (i < len)
34.     {
35.         if (i == 0)
36.             i = 1;
37.         if (cmp(data + i, data + i - 1) == 1)
```

```
38.             ++i;
39.         else
40.         {
41.             int tmp = data[i];
42.             data[i] = data[i-1];
43.             data[i-1] = tmp;
44.             --i;
45.         }
46.     }
47. }
48.
49. void shell(int *data, unsigned int len, int (*cmp)(const int *, const int *))
50. {
51.     unsigned step = len/2;
52.     while (step > 0)
53.     {
54.         unsigned short int k = 1;
55.         while(k)
56.         {
57.             k = 0;
58.             for(unsigned i = 0; i < len - step; i++)
59.             {
60.                 if(cmp(data + i, data + i + step) == 1)
61.                 {
62.                     int tmp = data[i];
63.                     data[i] = data[i + step];
64.                     data[i + step] = tmp;
65.                     k = 1;
66.                 }
67.             }
68.         }
69.         step /= 2;
70.     }
71. }
```

5. Описание тестовых наборов

Таблица 1: Тестовые наборы для программы prog1

№ теста	Необязательные входные опции
1	*без необязательных опций*
2	-s gnomes –p name
3	-s gnomes –p group -d
4	-s shell
5	-s shell -d
6	-p name -d

Таблица 2: Тестовые наборы для программы prog2

№ теста	Входные опции
1	10000 15
2	7500 20 –s gnomes
3	5000 30 –s shell

6. Скриншоты с результатами тестов

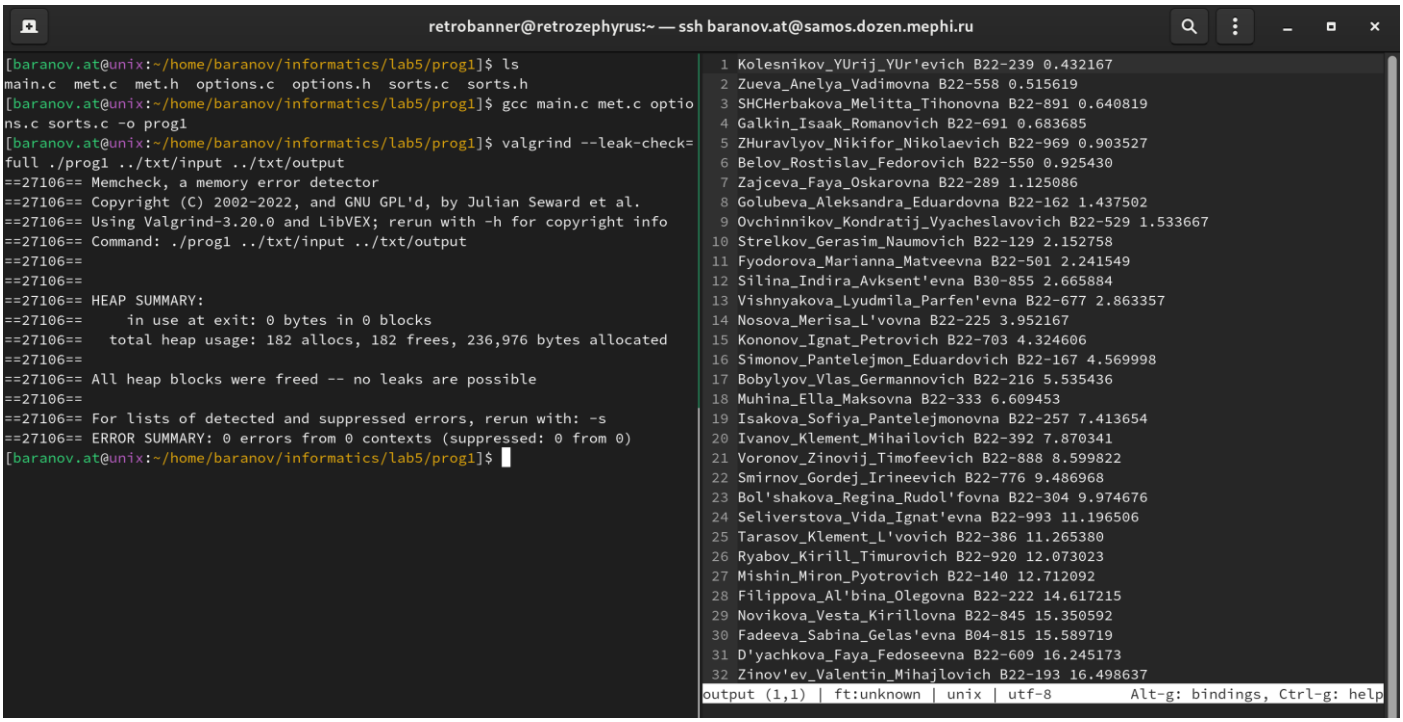

Рис. 3: Сборка запуск программы `prog1`, пример одной из сортировок по среднему баллу.

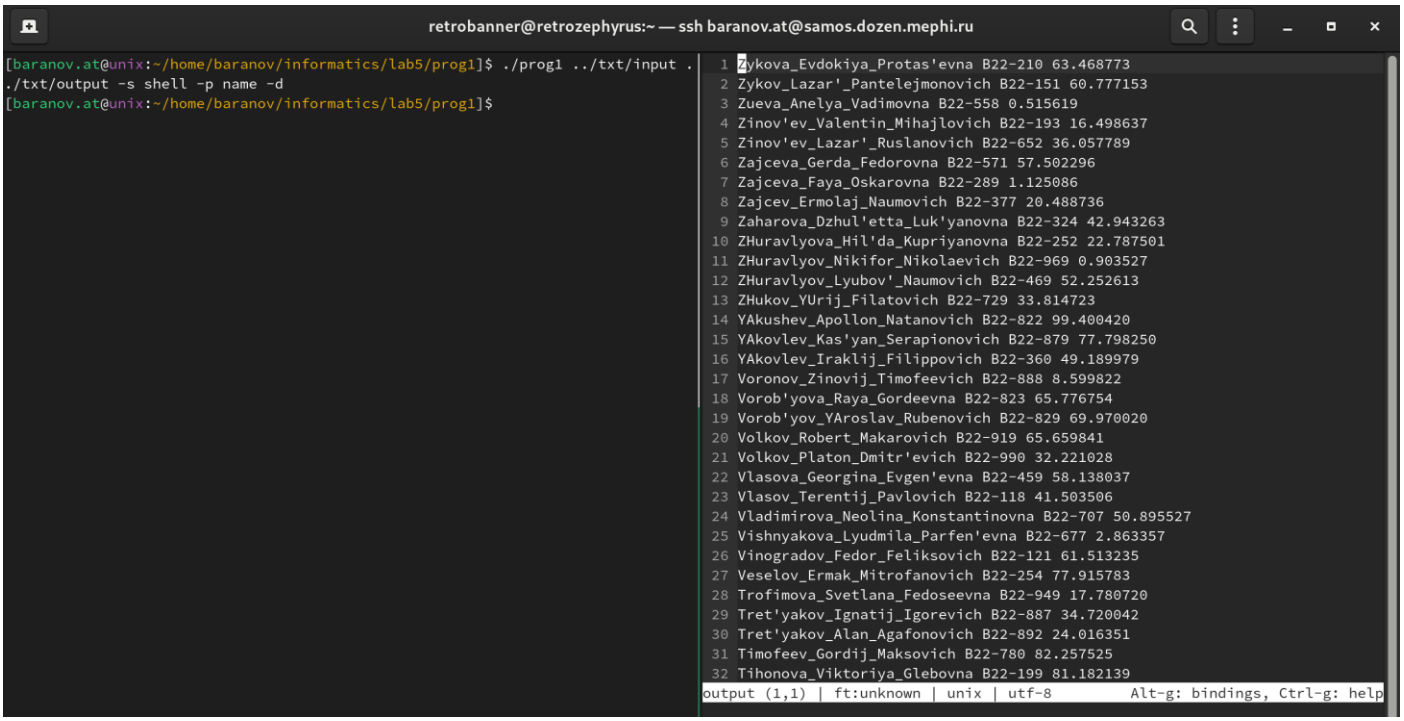

Рис. 4: Сборка запуск программы `prog1`, пример сортировки по имени по убыванию.

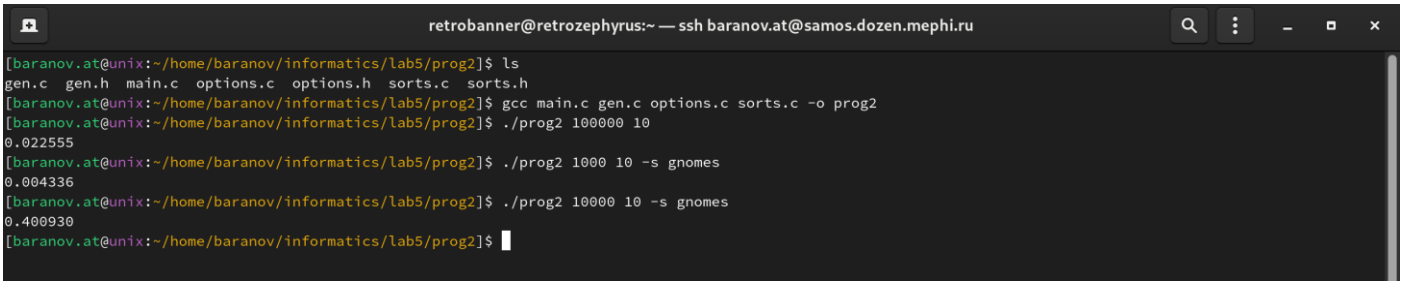

Рис. 5: Сборка и запуск программы `prog2`.

7. Исследование сортировок

`Quick sort`

Быстрая сортировка, или, по названию функции в Си, Qsort — это алгоритм сортировки, сложность которого в среднем составляет O(n*log(n)). При достаточно больших n график функции $n*\log_a n$ очень похож на прямую. Поэтому теоретически ожидаем, что график зависимости времени сортировки от количества элементов будет схожа с прямой. Построив график по точкам и задав логарифмическую линию тренда, получим:

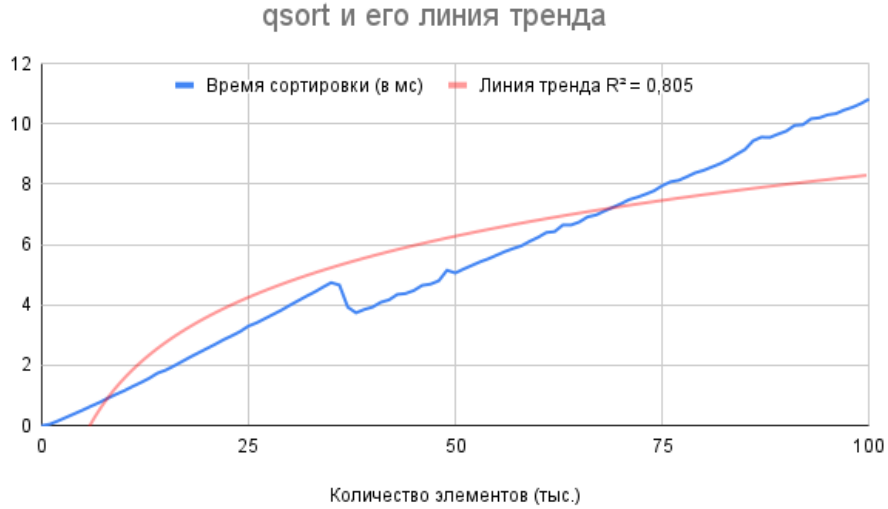

Рис. 6: Зависимость времени выполнения сортировки quick sort от количества сортируемых элементов.

Видно, что наш график, начиная с некоторого момента, растет быстрее, чем логарифмическая функция. Коэффициент корреляции, при этом, не столь высок. Сейчас попробуем сравнить с линейной линией тренда:

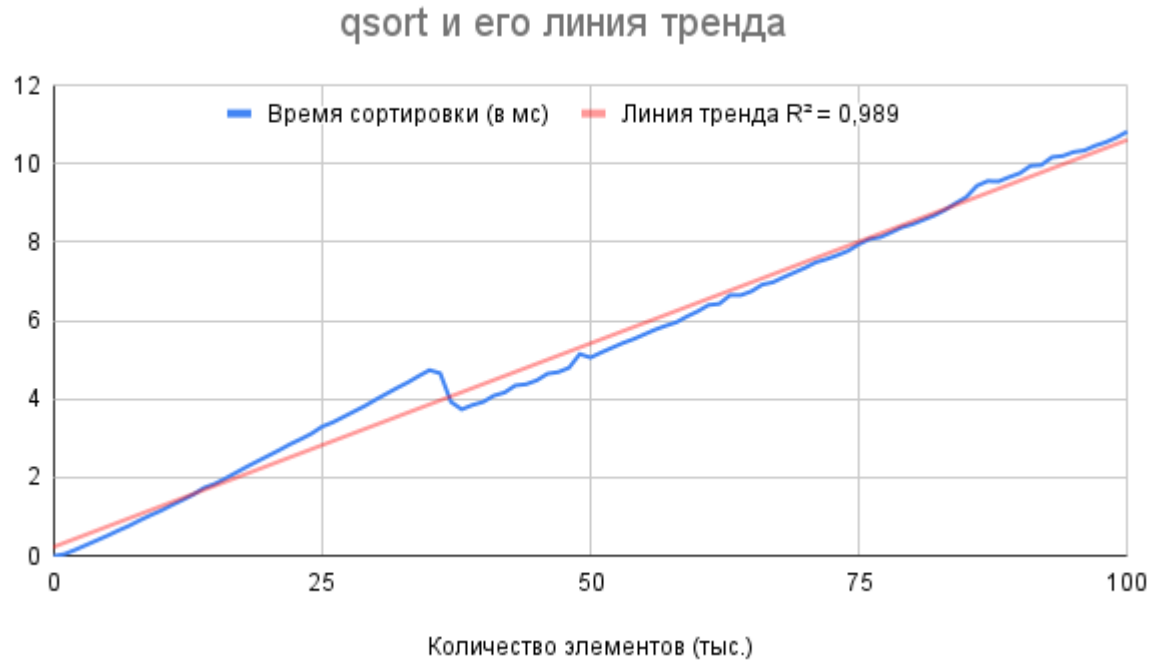

Рис. 7: линейная линия тренда qsort.

Здесь ситуация намного лучше, видно, что коэффициент корреляции очень близок к единице, поэтому функция ведет себя очень похоже на прямую, поэтому теоретическое ожидание подтвердилось.

Gnome sort

Гномья сортировка очень похожа на сортировку вставками, поэтому и её время оценивается как $O(n^2)$. Теоритическое ожидание – медленно возрастающая, похожая на параболу функция. Строим график и полиномиальную линию тренда со старшей степенью 2.

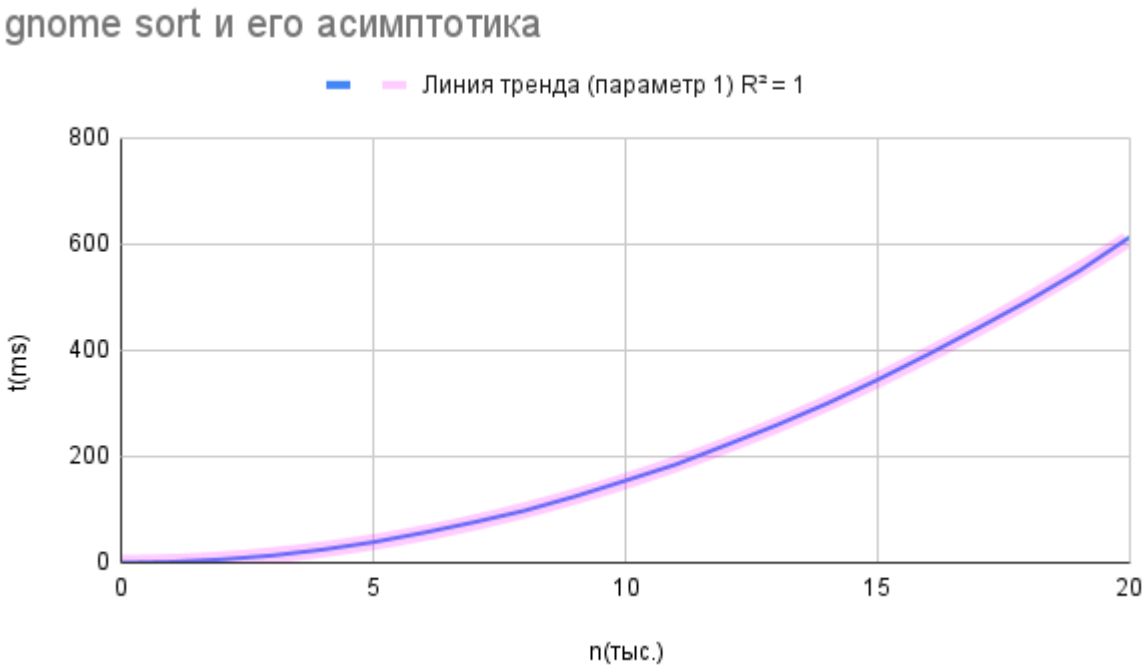

Рис. 8: Идеальная ассимптотика gnome sort.

Здесь без комментариев, теоретическое ожидание в точности повторилось.

Shell sort

Математики доказали, что максимальное время сортировки Шелла не превосходит $O(n^{1.5})$, причем уменьшить показатель степени 1.5 нельзя. Но для таких улучшений нужны сложные методы подбора шага d, используемого в сортировке, да и наилучший результат при таком раскладе показывается при строгой длине массива. В работе я использовал стандартный $d_n = 1/2\ d_{n-1}$. Он должен показывать результат $O(n^2)$. Давайте это и проверим, задав линию тренда полиномом со старшей степенью 2:

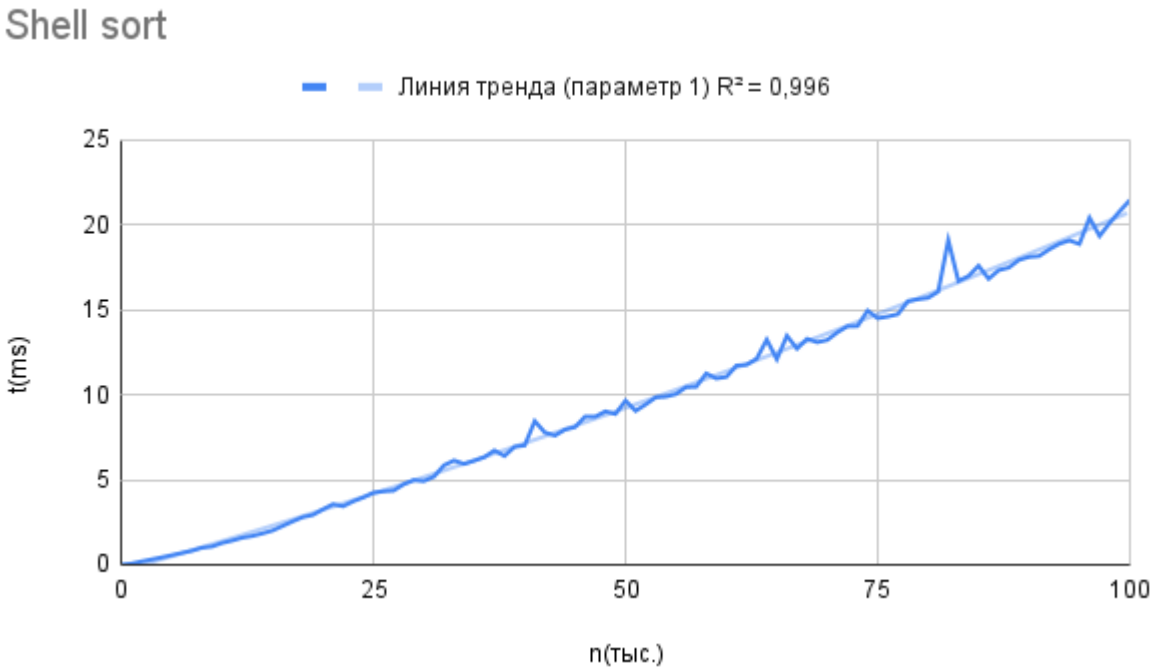

Рис. 9: Shell sort в сравнении с полиномом.

График получился «дерганым», но ничего не поделать. Как видим его также можно сравнить с линейным графиком:

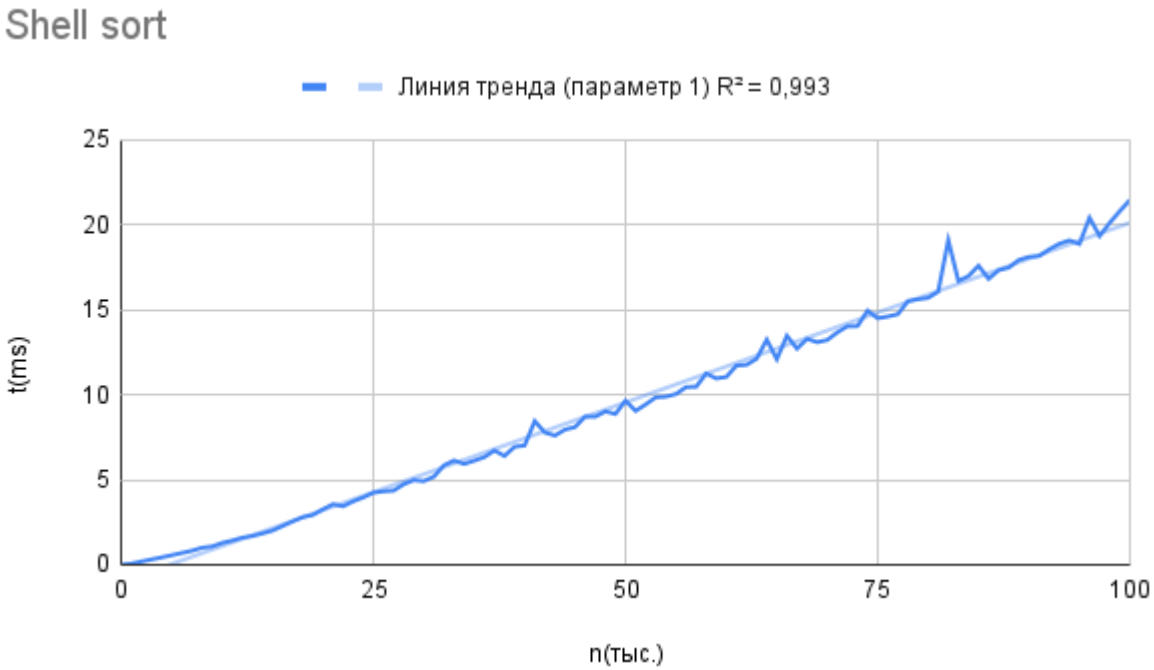

Рис. 10: Shell sort в сравнении с линией.

Но коэффициент корреляции показывает, что полиномная и линейные линии тренда отличаются, значил одночлен со степенью 2 тоже вносит свой вклад, откуда можно сделать вывод об $O(n^2)$.

Таблица таймирования:

Таблица 3: Таймирование

n(тыс.)	qsort(мс)	gnome sort(мс)	shell(мс)
0	0	0,00	0
1	0,05	1,54	0,06
2	0,16	6,11	0,19
3	0,28	13,50	0,31
4	0,4	24,65	0,43
5	0,52	38,64	0,56
6	0,65	56,48	0,69
7	0,77	76,22	0,82
8	0,91	97,93	1
9	1,04	124,80	1,07
10	1,16	154,52	1,28
11	1,3	184,96	1,42
12	1,43	221,62	1,59
13	1,57	259,12	1,69
14	1,74	299,86	1,84
15	1,84	344,43	2,01
16	1,98	392,07	2,28
17	2,13	442,25	2,55
18	2,28	493,90	2,8
19	2,42	549,52	2,93
20	2,56	613,01	3,26
21	2,7		3,56
22	2,85		3,44
23	2,98		3,73
24	3,12		3,96
25	3,3		4,24
26	3,41		4,32
27	3,55		4,36
28	3,69		4,73
29	3,83		4,98
30	3,99		4,92
31	4,14		5,18
32	4,29		5,85
33	4,43		6,13
34	4,59		5,93
35	4,74		6,12
36	4,66		6,33
37	3,93		6,72
38	3,74		6,4
39	3,85		6,94
40	3,93		7,02
41	4,09		8,45
42	4,17		7,78
43	4,35		7,6
44	4,38		7,95
45	4,48		8,1
46	4,65		8,72
47	4,69		8,71
48	4,8		9,02
49	5,15		8,87
50	5,06		9,65
51	5,19		9,05
52	5,31		9,43
53	5,43		9,85
54	5,53		9,89
55	5,65		10,04
56	5,77		10,45
57	5,87		10,47
58	5,96		11,25
59	6,11		10,96
60	6,24		11,05
61	6,4		11,72

62	6,43		11,76
63	6,65		12,12
64	6,65		13,24
65	6,75		12,09
66	6,92		13,47
67	6,98		12,71
68	7,11		13,29
69	7,23		13,09
70	7,35		13,22
71	7,49		13,67
72	7,57		14,03
73	7,67		14,04
74	7,78		14,95
75	7,95		14,5
76	8,08		14,6
77	8,13		14,74
78	8,25		15,51
79	8,38		15,62
80	8,46		15,7
81	8,57		16,07
82	8,68		19,09
83	8,82		16,69
84	8,99		16,98
85	9,15		17,59
86	9,44		16,82
87	9,56		17,33
88	9,55		17,48
89	9,66		17,93
90	9,76		18,11
91	9,95		18,16
92	9,97		18,55
93	10,17		18,88
94	10,2		19,08
95	10,3		18,87
96	10,34		20,41
97	10,46		19,34
98	10,55		20,11
99	10,67		20,8
100	10,82		21,45

Общий график:

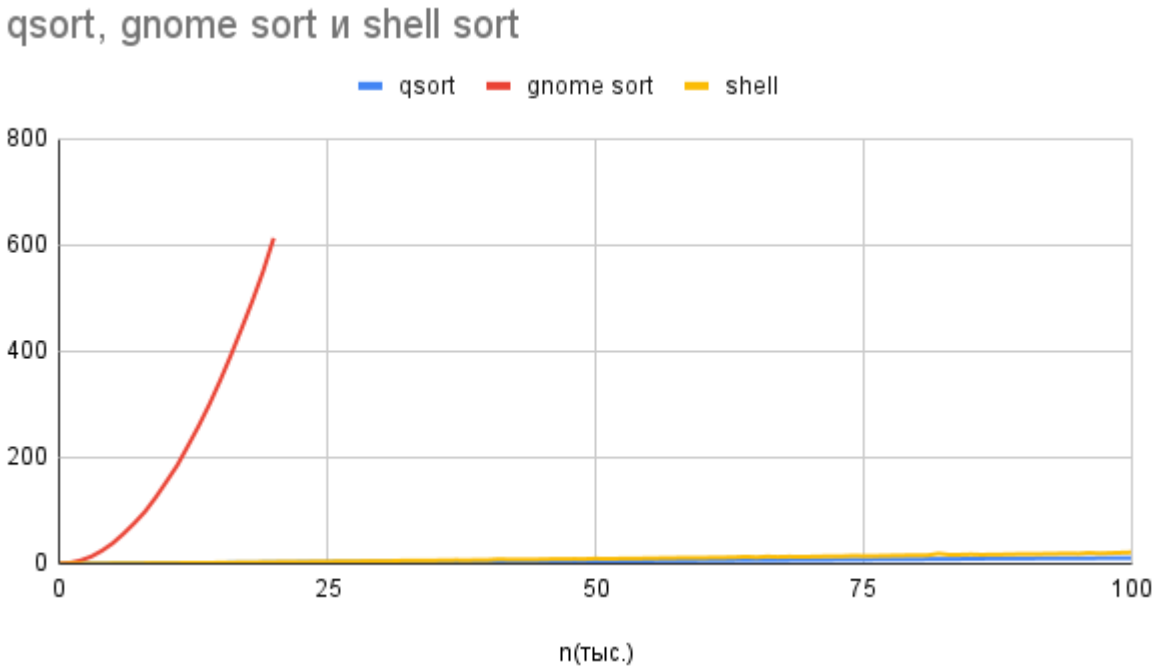

Рис. 11: Общий график.

8. Выводы

В ходе выполнения данной работы на примере программы, обрабатывающей данные, были рассмотрены базовые принципы работы построения программ на языке С и сортировки массивов:

1. Разработка функций для работы с файлами.
2. Объявление и использование переменных.
3. Работа с динамическим выделением и освобождением памяти.
4. Замеры времени работы программы.
5. Разработка функций сортировки.
6. Запуск программы с обязательными и необязательными опциями с помощью getopt.