

Национальный исследовательский ядерный университет «МИФИ»

Институт интеллектуальных кибернетических систем

Кафедра №12 «Компьютерные системы и технологии»



# ОТЧЕТ

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

Студент: Кукса И. В..

Группа: Б23-503

Преподаватель: Бабалова И. Ф.

Москва — 2023

# 1. Формулировка индивидуального задания

## 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. Гномья сортировка (Gnome sort).
2. Парная сортировка вставками (Pair insertion sort).

# 2. Описание использованных типов данных

1. Взаимодействие программ с пользователем должно быть выстроено с помощью иерархического диалогового меню.
2. Программа № 1 должна осуществлять проверку корректности данных, вводимых пользователем, и, в случае ошибок, выдавать соответствующие сообщения, после чего продолжать работу.
3. Программа № 1 должна осуществлять проверку корректности данных, считываемых из файлов. В случае ошибок формата файла — выдавать соответствующие сообщения в стандартный поток вывода ошибок и продолжать работу, считая что ввод не был выполнен успешно. В случае некорректных данных для конкретных записей — выдавать соответствующие сообщения в поток ошибок, после чего продолжать работу, игнорируя данные записи.
4. Для работы с данными, формат которых описан в индивидуальном задании, должен быть разработан собственный составной тип данных — структура.
5. Для работы с данными, структура которых описана в индивидуальном задании, должен быть разработан формат хранения в текстовом файле.
6. Для работы с данными, структура которых описана в индивидуальном задании, должен быть разработан формат хранения в бинарном файле.
7. Работа с текстовыми файлами должна осуществляться при помощи функций стандартной библиотеки `fopen()`, `fclose()`, `fprintf()`, `fscanf()`.
8. Работа с бинарными файлами должна осуществляться при помощи функций стандартной библиотеки `fopen()`, `fclose()`, `fread()`, `fwrite()`.
9. Логически законченные части алгоритма решения задачи должны быть оформлены в виде отдельных функций с параметрами. Использование глобальных переменных не допускается.
10. Исходные коды программы должны быть логичным образом разбиты на несколько файлов.
11. Программа должна корректным образом работать с памятью, для проверки необходимо использовать соответствующие программные средства, например: `valgrind` (при тестировании и отладке программы ее необходимо запускать командой вида `valgrind ./lab5`, а при анализе производительности — `./lab5`).

### 3. Описание использованного алгоритма

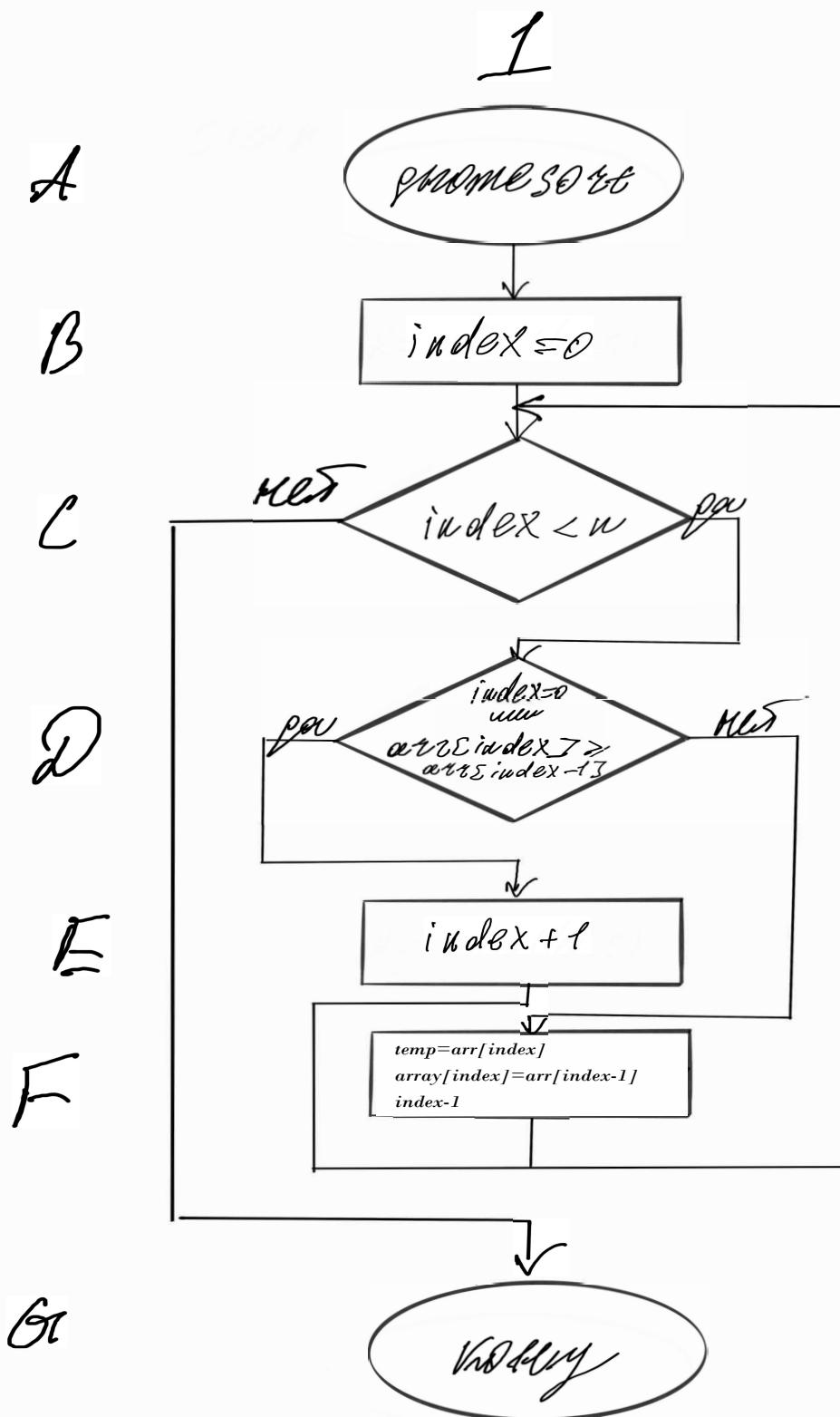


Рис. 3: Блок-схема функции  
array.h (gnomesort)

1

2

3

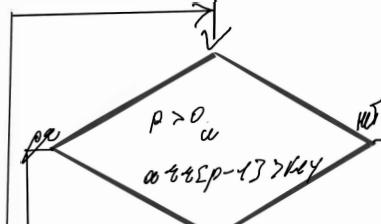
A

 $\text{array}[\text{p}] = \text{key}$ 

B

 $p = n - i$   
 $\text{key} = \text{array}[p]$ 

C



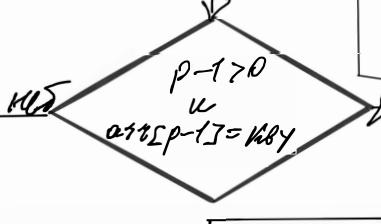
D

 $\text{array}[p] = \text{array}[p-1]$   
 $p = p - 1$ 

E

 $\text{array}[p] = \text{key}$ 

F



G

 $\text{array}[p-1] = \text{array}[p-2]$   
 $p = p - 1$ 

H

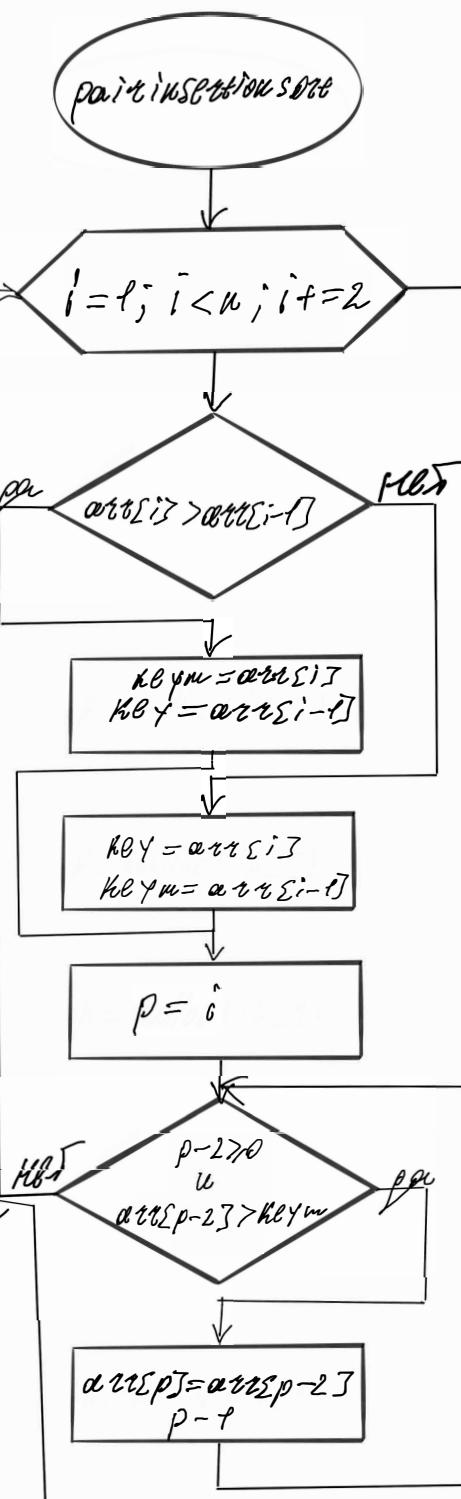


Рис. 5: Блок-схема  
функции array.h  
(pairinsertsort)

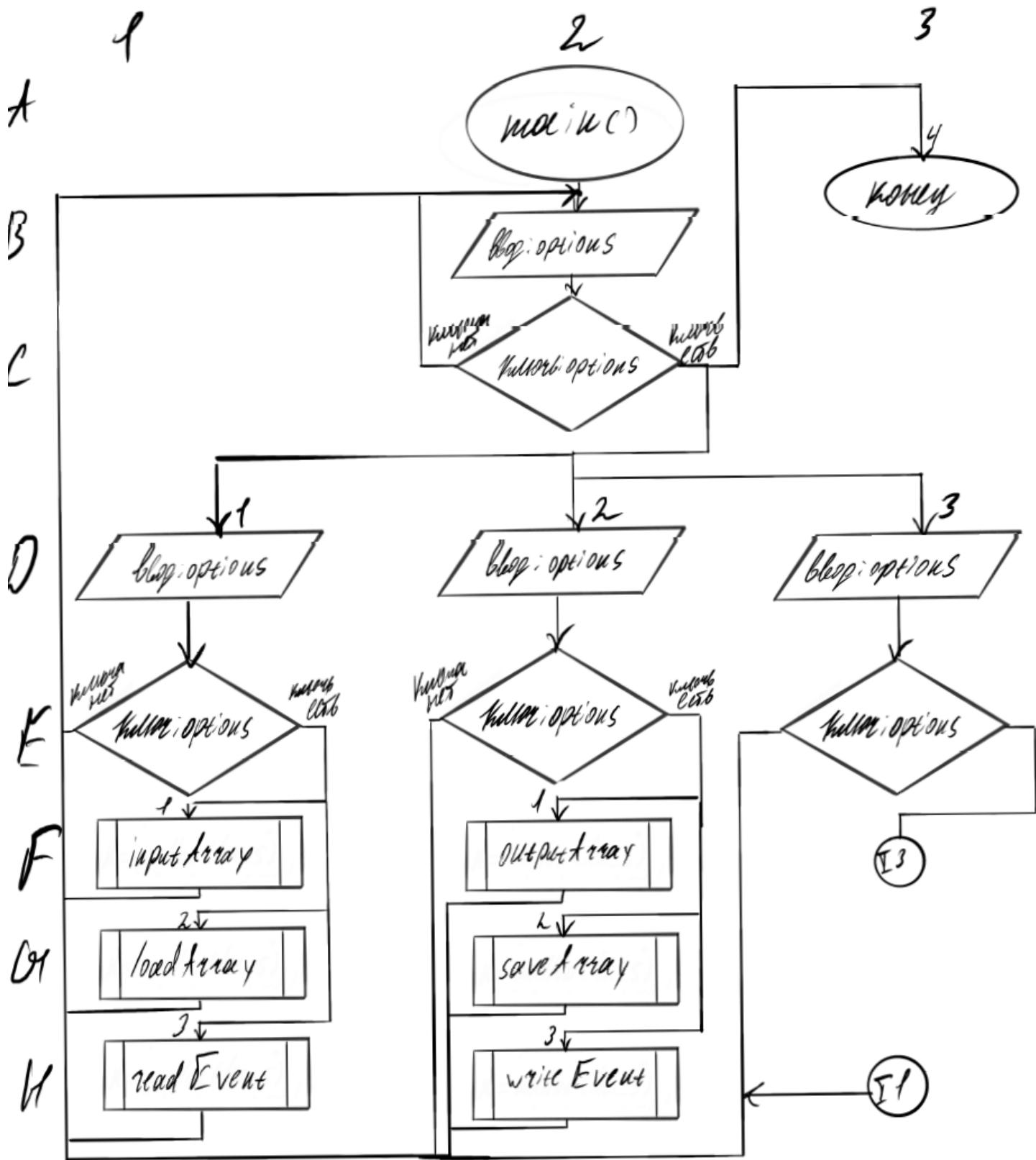
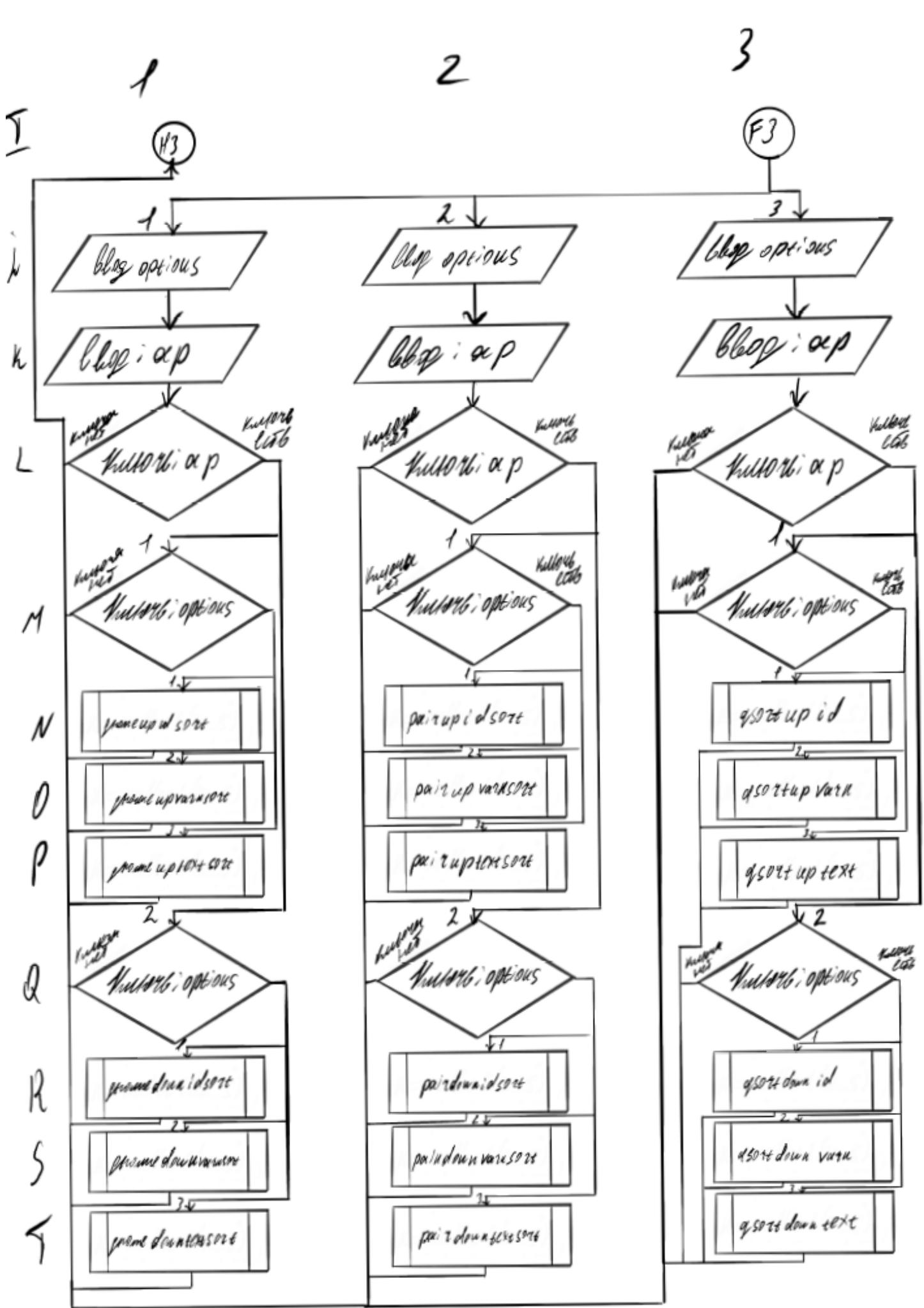


Рис. 6: Блок-схема функции main.c



## 4.1 Исходные коды разработанных программ

Листинг 1: Исходные коды программы Lab51 (файл: lab51.c)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "array.h"
5 #include <readline/readline.h>
6 int main() {
7     char textFileName[100];
8     int size;
9     int option;
10    int ss=0;
11    LogEntry* logArray = (LogEntry*) malloc(sizeof(LogEntry));
12
13    do{
14        print_1();
15        printf("Enter: ");
16        scanf("%d",&option);
17
18        switch(option){
19            case 1:
20                //in
21                print_in();
22                printf("Enter: ");
23                scanf("%d",&option);
24                switch (option){
25                    case 1:
26                        //std in
27                        inputLogArray(&logArray, &size);
28                        break;
29                    case 2:
30                        //text in
31                        printf("Enter the name of the text file: ");
32                        scanf("%s", textFileName);
33                        loadLogArrayFromFile(&logArray, &size, textFileName);
34                        break;
35                    case 3:
36                        //bin in
37                        printf("Enter the name of the text file: ");
38                        scanf("%s", textFileName);
39                        readEventFromFile(&logArray,&size,textFileName);
40                        break;
41                    default:
42                        break;
43                }
44                break;
45            case 2:
46                //out
47                print_out();
48                printf("Enter: ");
49                scanf("%d",&option);
50                switch (option){
51                    case 1:
52                        //std out
53                        outputLogArray(logArray, size);
54                        break;
55                    case 2:
56                        //text save
57                        printf("Enter the name of the text file: ");
58                        scanf("%s", textFileName);
59                        saveLogArrayToFile(logArray, size, textFileName);
60                        break;
61                    case 3:
62                        //bin save
63                        printf("Enter the name of the text file: ");
64                        scanf("%s", textFileName);
65                        writeEventToFile(logArray,size,textFileName);
66                        break;
67                    default:
68                        break;
69                }
70                break;
71            case 3:
72                //sort
73                print_sort();
74        }
```

```

74     print_sort();
75     printf("Enter: ");
76     scanf("%d",&option);
77     switch (option){
78         case 1:
79             //gnome sort
80             print_h();
81             printf("Enter: ");
82             scanf("%d",&option);
83             gnomeSort(&logArray,size,option);
84             break;
85         case 2:
86             //pair sort
87             print_h();
88             printf("Enter: ");
89             scanf("%d",&option);
90             pairInsertionSort(&logArray,size,option);
91             break;
92         case 3:
93             //qsort
94             int ap;
95             print_h();
96             printf("Enter: ");
97             scanf("%d",&option);
98             print_updown();
99             printf("Enter: ");
100            scanf("%d",&ap);
101            switch (ap){
102                case 1:
103                    //UP
104                    switch (option){
105                        case 1:
106                            //id
107                            qsort(logArray,size,sizeof(LogEntry),comupid);
108                            break;
109                        case 2:
110                            //varn
111                            qsort(logArray,size,sizeof(LogEntry),comuplevel);
112                            break;
113                        case 3:
114                            //txt
115                            qsort(logArray,size,sizeof(LogEntry),comuptext);
116                            break;
117                        default:
118                            break;
119                    }
120                    break;
121                case 2:
122                    //DOWN
123                    switch (option){
124                        case 1:
125                            //id
126                            qsort(logArray,size,sizeof(LogEntry),comdownid);
127                            break;
128                        case 2:
129                            //varn
130                            qsort(logArray,size,sizeof(LogEntry),comdownlevel);
131                            break;
132                        case 3:
133                            //txt
134                            qsort(logArray,size,sizeof(LogEntry),comdowntext);
135                            break;
136                        default:
137                            break;
138                    }
139                }
140                break;
141
142            default:
143                break;
144            }
145        break;
146    case 4:
147

```

```

147     case 4:|
148         //stop
149         ss=1;
150         break;
151     default:
152         break;
153     }
154 }while(ss==0);
155
156 free(logArray);
157
158 }
159

```

Листинг 2: Исходные коды программы Lab51 (файл: array1.h)

```

1 #ifndef _LIB_ARRAY_
2 #define _LIB_ARRAY_
3
4
5
6 typedef enum {debug, info, warn, error, fatal} LogLevel;
7
8 typedef struct {
9     int id;
10    LogLevel level;
11    char text;
12 } LogEntry;
13
14 int comupid(const void *x,const void *y);
15 int comdownid(const void *x,const void *y);
16 int comuplevel(const void *x,const void *y);
17 int comdownlevel(const void *x,const void *y);
18 int comuptext(const void *x,const void *y);
19 int comdowntext(const void *x,const void *y);
20 void outputLogArray(LogEntry* array, int size) ;
21 void print_updown();
22 void gnomeSort(LogEntry* array, int size,int op) ;
23 void pairInsertionSort(LogEntry* arr, int n,int op) ;
24 void inputLogArray(LogEntry* array, int *size) ;
25 void saveLogArrayToFile(LogEntry* array, int size, const char* filename);
26 void loadLogArrayFromFile(LogEntry* array, int *size, const char* filename) ;
27 void readEventFromFile(LogEntry* array,int *size, char* filename) ;
28 void writeEventToFile(LogEntry* event,int size, const char* filename);
29 void print_1();
30 void print_in();
31 void print_out();
32 void print_sort();
33 void print_h();
34 #endif
35

```

### Листинг 3: Исходные коды программы Lab51 (файл: array1.c)

```
1 #include "array1.h"
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 int comupid(const void *x,const void *y){
7     LogEntry *xx =(LogEntry*)x;
8     LogEntry *yy =(LogEntry*)y;
9     return (xx->id-yy->id);
10 }
11
12 int comdownid(const void *x,const void *y){
13     LogEntry *xx =(LogEntry*)x;
14     LogEntry *yy =(LogEntry*)y;
15     return (yy->id-xx->id);
16 }
17
18 int comuplevel(const void *x,const void *y){
19     LogEntry *xx =(LogEntry*)x;
20     LogEntry *yy =(LogEntry*)y;
21     return (xx->level-yy->level);
22 }
23
24 int comdownlevel(const void *x,const void *y){
25     LogEntry *xx =(LogEntry*)x;
26     LogEntry *yy =(LogEntry*)y;
27     return (yy->level-xx->level);
28 }
29
30 int comuptext(const void *x,const void *y){
31     LogEntry *xx =(LogEntry*)x;
32     LogEntry *yy =(LogEntry*)y;
33     return strcmp((xx->text), (yy->text));
34 }
35
36 int comdowntext(const void *x,const void *y){
37     LogEntry *xx =(LogEntry*)x;
38     LogEntry *yy =(LogEntry*)y;
39     return strcmp((yy->text), (xx->text));
40 }
41
42 void outputLogArray(LogEntry* array, int size) {
43     printf("\n\n");
44     for (int i = 0; i < size; i++) {
45         printf("id: %d, level: %d, text: %s\n", array[i].id, array[i].level, array[i].text);
46     }
47     printf("\n");
48 }
49
50 void print_updown(){
51     printf("\n1) UP\n");
52     printf("2) Down\n");
53 }
54
55 void gnomeSort(LogEntry* *array, int size,int op) {
56     int index = 0;
57     LogEntry temp;
58     int ap;
59     print_updown();
60     printf("Enter: ");
61     scanf("%d",&ap);
62
63     switch (ap){
64         case 1:
65             //UP
66             switch (op){
67                 case 1:
68                     //id
69                     while (index < size) {
70                         if (index == 0 || (*array)[index].id >= (*array)[index - 1].id) {
71                             index++;
72                         } else {
73                             temp=(*array)[index];
74                             (*array)[index]=(*array)[index-1];
```

```

74             (*array)[index]=(*array)[index-1];
75             (*array)[index-1]=temp;
76
77         index--;
78     }
79     break;
80 case 2:
81     //varn
82     while (index < size) {
83         if (index == 0 || (*array)[index].level >= (*array)[index - 1].level) {
84             index++;
85         } else {
86             temp=(*array)[index];
87             (*array)[index]=(*array)[index-1];
88             (*array)[index-1]=temp;
89
90             index--;
91         }
92     }
93     break;
94 case 3:
95     //txt
96     while (index < size) {
97         if (index == 0 || strcmp((*array)[index].text, (*array)[index-1].text) < 0) {
98             index++;
99         } else {
100            temp=(*array)[index];
101            (*array)[index]=(*array)[index-1];
102            (*array)[index-1]=temp;
103
104            index--;
105        }
106    }
107    break;
108 default:
109     break;
110 }
111 break;
112 case 2:
113     //DOWN
114     switch (op){
115         case 1:
116             //id
117             while (index < size) {
118                 if (index == 0 || (*array)[index].id <= (*array)[index - 1].id) {
119                     index++;
120                 } else {
121                     temp=(*array)[index];
122                     (*array)[index]=(*array)[index-1];
123                     (*array)[index-1]=temp;
124
125                     index--;
126                 }
127             }
128             break;
129         case 2:
130             //varn
131             while (index < size) {
132                 if (index == 0 || (*array)[index].level <= (*array)[index - 1].level) {
133                     index++;
134                 } else {
135                     temp=(*array)[index];
136                     (*array)[index]=(*array)[index-1];
137                     (*array)[index-1]=temp;
138
139                     index--;
140                 }
141             }
142             break;
143         case 3:
144             //txt
145             while (index < size) {
146                 if (index == 0 || (strcmp((*array)[index].text, (*array)[index-1].text) > 0)) {

```

```

147         if (index == 0 || (strcmp((*array)[index].text, (*array)[index-1].text) > 0)) {
148             index++;
149         } else {
150             temp=(*array)[index];
151             (*array)[index]=(*array)[index-1];
152             (*array)[index-1]=temp;
153
154             index--;
155         }
156     }
157     break;
158   default:
159     break;
160   }
161 }
162 }
163 }
164 }
165
166
167
168
169 void pairInsertionSort(LogEntry* *arr, int n,int op) {
170 LogEntry key;
171 LogEntry keym;
172 int p;
173 int ap;
174 print_updown();
175 printf("Enter: ");
176 scanf("%d",&ap);
177
178 switch (ap){
179   case 1:
180     //UP
181     switch (op){
182       case 1:
183         //id
184         for (int i = 1; i < n; i += 2) {
185           if((*arr)[i].id>(*arr)[i-1].id){
186             key=(*arr)[i];
187             key=(*arr)[i-1];
188           }else{
189             key=(*arr)[i];
190             key=(*arr)[i-1];
191           }
192           p=i;
193           while(p-2>=0 && (*arr)[p-2].id>keym.id){
194             (*arr)[p]=(*arr)[p-2];
195             p--;
196           }
197           (*arr)[p]=keym;
198
199           while(p-1>0 && (*arr)[p-1].id>key.id){
200             (*arr)[p-1]=(*arr)[p-2];
201             p--;
202           }
203           (*arr)[p-1]=key;
204         }
205         p=n-1;
206         key=(*arr)[p];
207         while(p>0 && (*arr)[p-1].id>key.id){
208             (*arr)[p]=(*arr)[p-1];
209             p--;
210           }
211         break;
212       case 2:
213         //varn
214         for (int i = 1; i < n; i += 2) {
215           if((*arr)[i].level>(*arr)[i-1].level){
216             keym=(*arr)[i];
217             key=(*arr)[i-1];
218           }else{
219             key=(*arr)[i];
220             keym=(*arr)[i-1];

```

```

220             keym=(*arr)[i-1];
221         }
222         p=i;
223         while(p-2>=0 && (*arr)[p-2].level>keym.level){
224             (*arr)[p]=(*arr)[p-2];
225             p--;
226         }
227         (*arr)[p]=keym;
228
229         while(p-1>0 && (*arr)[p-1].level>key.level){
230             (*arr)[p-1]=(*arr)[p-2];
231             p--;
232         }
233         (*arr)[p-1]=key;
234     }
235     p=n-1;
236     key=(*arr)[p];
237     while(p>0 && (*arr)[p-1].level>key.level){
238         (*arr)[p]=(*arr)[p-1];
239         p--;
240     }
241     break;
242 case 3:
243     //txt
244     for (int i = 1; i < n; i += 2) {
245         if(strcmp((*arr)[i].text, (*arr)[i-1].text) < 0){
246             keym=(*arr)[i];
247             key=(*arr)[i-1];
248         }else{
249             key=(*arr)[i];
250             keym=(*arr)[i-1];
251         }
252         p=i;
253         while(p-2>=0 && strcmp((*arr)[p-2].text, keym.text) < 0 ){
254             (*arr)[p]=(*arr)[p-2];
255             p--;
256         }
257         (*arr)[p]=keym;
258
259         while(p-1>0 && strcmp((*arr)[p-1].text, key.text) < 0 ){
260             (*arr)[p-1]=(*arr)[p-2];
261             p--;
262         }
263         (*arr)[p-1]=key;
264     }
265     p=n-1;
266     key=(*arr)[p];
267     while(p>0 && strcmp((*arr)[p-1].text, key.text) < 0 ){
268         (*arr)[p]=(*arr)[p-1];
269         p--;
270     }
271     break;
272 default:
273     break;
274 }
275 break;
276 case 2:
277     //DOWN
278     switch (op){
279         case 1:
280             //id
281             for (int i = 1; i < n; i += 2) {
282                 if((*arr)[i].id<(*arr)[i-1].id){
283                     keym=(*arr)[i];
284                     key=(*arr)[i-1];
285                 }else{
286                     key=(*arr)[i];
287                     keym=(*arr)[i-1];
288                 }
289                 p=i;
290                 while(p-2>=0 && (*arr)[p-2].id<keym.id){
291                     (*arr)[p]=(*arr)[p-2];
292                     p--;
293                 }

```

```

289     p=i;
290     while(p-2>=0 && (*arr)[p-2].id<keym.id){
291         (*arr)[p]=(*arr)[p-2];
292         p--;
293     }
294     (*arr)[p]=keym;
295
296     while(p-1>0 && (*arr)[p-1].id<key.id){
297         (*arr)[p-1]=(*arr)[p-2];
298         p--;
299     }
300     (*arr)[p-1]=key;
301     }
302     p=n-1;
303     key=(*arr)[p];
304     while(p>0 && (*arr)[p-1].id<key.id){
305         (*arr)[p]=(*arr)[p-1];
306         p--;
307     }
308     break;
309 case 2:
310     //varn
311     for (int i = 1; i < n; i += 2) {
312         if((*arr)[i].level<(*arr)[i-1].level){
313             keym=(*arr)[i];
314             key=(*arr)[i-1];
315         }else{
316             key=(*arr)[i];
317             keym=(*arr)[i-1];
318         }
319         p=i;
320         while(p-2>=0 && (*arr)[p-2].level<keym.level){
321             (*arr)[p]=(*arr)[p-2];
322             p--;
323         }
324         (*arr)[p]=keym;
325
326         while(p-1>0 && (*arr)[p-1].level<key.level){
327             (*arr)[p-1]=(*arr)[p-2];
328             p--;
329         }
330         (*arr)[p-1]=key;
331     }
332     p=n-1;
333     key=(*arr)[p];
334     while(p>0 && (*arr)[p-1].level<key.level){
335         (*arr)[p]=(*arr)[p-1];
336         p--;
337     }
338     break;
339 case 3:
340     //txt
341     for (int i = 1; i < n; i += 2) {
342         if(strcmp((*arr)[i].text, (*arr)[i-1].text) > 0){
343             keym=(*arr)[i];
344             key=(*arr)[i-1];
345         }else{
346             key=(*arr)[i];
347             keym=(*arr)[i-1];
348         }
349         p=i;
350         while(p-2>=0 && strcmp((*arr)[p-2].text, keym.text) > 0 ){
351
352             (*arr)[p]=(*arr)[p-2];
353             p--;
354         }
355     }
356     (*arr)[p]=keym;
357
358     while(p-1>0 && strcmp((*arr)[p-1].text, key.text) > 0 ){
359
360
361

```

```

361                     while(p-1>0 && strcmp((*arr)[p-1].text, key.text) > 0 ){
362                         (*arr)[p-1]=(*arr)[p-2];
363                         p--;
364                     }
365                     (*arr)[p-1]=key;
366                     }
367                     p=n-1;
368                     key=(*arr)[p];
369                     while(p>0 && strcmp((*arr)[p-1].text, key.text) > 0 ){
370                         (*arr)[p]=(*arr)[p-1];
371                         p--;
372                     }
373                     break;
374                 default:
375                     break;
376             }
377             break;
378         default:
379             break;
380     }
381     (*arr)[p]=key;
382
383 }
384
385 }
386
387
388 void inputLogArray(LogEntry* *array, int *size) {
389     printf("Enter the size of the log array: ");
390     scanf("%d", &(*size));
391
392     *array = realloc(*array, (*size)*sizeof(LogEntry));
393
394
395     for (int i = 0; i < *size; i++) {
396         printf("Enter id: ");
397         scanf("%d", &(*array)[i].id);
398
399         printf("Enter level (0-debug, 1-info, 2-warn, 3-error, 4-fatal): ");
400         int level;
401         scanf("%d", &level);
402         (*array)[i].level = (LogLevel) level;
403
404         printf("Enter text: ");
405         (*array)[i].text=readline("");
406     }
407 }
408
409
410 void saveLogArrayToFile(LogEntry* array, int size, const char* filename) {
411     FILE* file = fopen(filename, "w");
412     if (file == NULL) {
413         printf("Failed to open file for writing.\n");
414         return;
415     }
416     for (int i=0;i<size;i++){
417         fprintf(file, "%d\n", array[i].id);
418         fprintf(file, "%d\n", array[i].level);
419         fprintf(file, "%s\n", array[i].text);
420     }
421     fclose(file);
422 }
423
424
425 void loadLogArrayFromFile(LogEntry* *array, int *size, const char* filename) {
426     FILE* file = fopen(filename, "r");
427     if (file == NULL) {
428         printf("Failed to open file for reading.\n");
429         return;
430     }
431     int count=0;
432     char line[256];
433     int linelen;
434     *size=0;

```

```

433     int linelen;
434     *size=0;
435
436     while (fgets(line, sizeof(line), file)) {
437         if (count%3==0){
438             *size+=1;
439             count=0;
440             *array = realloc(*array, (*size)*sizeof(LogEntry));
441         }
442         switch (count) {
443             case 0:
444                 (*array)[*size-1].id = atoi(line);
445                 break;
446             case 1:
447                 (*array)[*size-1].level = atoi(line);
448                 break;
449             case 2:
450                 linelen=strcspn(line, "\n");
451
452                 strncpy((*array)[*size-1].text, line, linelen);
453                 (*array)[*size-1].text[linelen]='\0';
454                 break;
455         }
456     }
457
458     count++;
459 }
460
461
462     fclose(file);
463 }
464 void readEventFromFile(LogEntry* *array, int *size, char* filename) {
465     FILE* file = fopen(filename, "rb");
466     if (file == NULL) {
467         printf("Не удалось открыть файл.\n");
468         return;
469     }
470     *size=0;
471     LogEntry event;
472     while (fread(&event, sizeof(LogEntry), 1, file) == 1) {
473         *size+=1;
474         *array = realloc(*array, (*size)*sizeof(LogEntry));
475         (*array)[*size-1]=event;
476     }
477     fclose(file);
478 }
479 void writeEventToFile(LogEntry* event, int size, const char* filename) {
480     FILE* file = fopen(filename, "wb");
481     if (file == NULL) {
482         printf("Не удалось открыть файл.\n");
483         return;
484     }
485     for(int i=0 ;i<size;i++){
486         fwrite(&event[i], sizeof(LogEntry), 1, file);
487         fclose(file);
488     }
489 }
```

```

489 void print_1(){
490     printf("\n1) Input\n");
491     printf("2) Output\n");
492     printf("3) Sort\n");
493     printf("4) Stop\n");
494 }
495 void print_in(){
496     printf("\n1) From std key\n");
497     printf("2) file\n");
498     printf("3) bin file\n");
499 }
500 }
501 void print_out(){
502     printf("\n1) From std key\n");
503     printf("2) file\n");
504     printf("3) bin file\n");
505 }
506 }
507 void print_sort(){
508     printf("\n1) Gnome\n");
509     printf("2) Two pair\n");
510     printf("3) qsort\n");
511 }
512 }
513 void print_h(){
514     printf("\n1) id\n");
515     printf("2) varn\n");
516     printf("3) txt\n");
517 }
518 }
519 }
520 }
521 
```

## 4.2 Исходные коды разработанных программ

Листинг 1: Исходные коды программы Lab52 (файл: lab52.c)

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <time.h>
5 #include "array2.h"
6
7 int main() {
8     double time;
9     clock_t start, fin;
10    int km, size, ss=0, option, ilt;
11    int ap;
12
13    LogEntry* logArray = (LogEntry*) malloc(sizeof(LogEntry));
14    do{
15
16        printf("%f\n", time);
17        printf("0) Начать сортировать\n");
18        printf("1) Стоп\n");
19        printf("ENTER: ");
20        scanf("%d", &ss);
21        if (ss==0){
22
23            printf("Введите количество массивов: ");
24            scanf("%d", &km);
25            printf("Введите количество структур в массивах: ");
26            scanf("%d", &size);
27
28            print_sort();
29            printf("Enter: ");
30            scanf("%d", &option);
31            printf("\n\n"); 
```

```

35 logArray = realloc(logArray,(size)*sizeof(LogEntry));
36     if (logArray == NULL) {
37         printf("Ошибка выделения памяти\n");
38     }
39     print_h();
40     printf("Enter: ");
41     scanf("%d",&ilt);
42     print_updown();
43     printf("Enter: ");
44
45     scanf("%d",&ap);
46     printf("\n\n");
47     start=clock();
48
49
50 for (int t=0;t<km;t++){
51     for (int i = 0; i < size; i++) {
52         logArray[i].id = getRandomNumber(1, 100);
53         logArray[i].level = getRandomLevel();
54         getRandomMessage(logArray[i].text);
55     }
56     switch (option){
57         case 1:
58             //gnome sort
59
60             gnomeSort(&logArray,size,ilt,ap);
61             break;
62         case 2:
63             //pair sort
64
65             pairInsertionSort(&logArray,size,ilt,ap);
66             break;
67         case 3:
68             //qsort
69
70
71         switch (ap){
72             case 1:
73                 //UP
74                 switch (ilt){
75                     case 1:
76                         //id
77                         qsort(logArray,size,sizeof(LogEntry),comupid);
78                         break;
79                     case 2:
80                         //varn
81                         qsort(logArray,size,sizeof(LogEntry),comuplevel);
82                         break;
83                     case 3:
84                         //txt
85                         qsort(logArray,size,sizeof(LogEntry),comuptext);
86                         break;
87                     default:
88                         break;
89                 }
90                 break;
91             case 2:
92                 //DOWN
93                 switch (ilt){
94                     case 1:
95                         //id
96                         qsort(logArray,size,sizeof(LogEntry),comdownid);
97                         break;
98                     case 2:
99                         //varn
100                        qsort(logArray,size,sizeof(LogEntry),comdownlevel);
101                        break;
102                     case 3:
103                         //txt
104                         qsort(logArray,size,sizeof(LogEntry),comdowntext);
105                         break;
106                     default:
107                         break;
108                 }
109             }
110         }
111     }
112 }

```

```

106                     default:
107                         break;
108                     }
109                 }
110             break;
111         default:
112             break;
113     }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122     fin=clock();
123     time=(double)(fin-start);
124 }
125 }while(ss==0);
126
127
128
129
130
131 free(logArray);
132
133 return 0;
134 }
135

```

Листинг 2: Исходные коды программы Lab51 (файл: array2.h)

```

1 #ifndef _LIB_ARRAY2_
2 #define _LIB_ARRAY2_
3
4
5
6 typedef enum {debug, info, warn, error, fatal} LogLevel;
7
8 typedef struct {
9     int id;
10    LogLevel level;
11    char text;
12 } LogEntry;
13
14
15 int comupid(const void *x,const void *y);
16 int comdownid(const void *x,const void *y);
17 int comuplevel(const void *x,const void *y);
18 int comdownlevel(const void *x,const void *y);
19 int comuptext(const void *x,const void *y);
20 int comdowntext(const void *x,const void *y);
21 void outputLogArray(LogEntry* array, int size) ;
22 void print_updown();
23 void gnomeSort(LogEntry* array, int size,int op,int ap);
24 void pairInsertionSort(LogEntry* arr, int n,int op,int ap) ;
25 int getRandomNumber(int min, int max) ;
26 int getRandomLevel() ;
27 void getRandomMessage(char *message) ;
28 void print_sort();
29 void print_h();
30
31#endif

```

### Листинг 3: Исходные коды программы Lab51 (файл: array2.c)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <time.h>
5 #include "array2.h"
6
7
8 int comupid(const void *x,const void *y){
9     LogEntry *xx =(LogEntry*)x;
10    LogEntry *yy =(LogEntry*)y;
11    return (xx->id-yy->id);
12
13 }
14 int comdownid(const void *x,const void *y){
15     LogEntry *xx =(LogEntry*)x;
16     LogEntry *yy =(LogEntry*)y;
17     return (yy->id-xx->id);
18
19 }
20 int comulevel(const void *x,const void *y){
21     LogEntry *xx =(LogEntry*)x;
22     LogEntry *yy =(LogEntry*)y;
23     return (xx->level-yy->level);
24
25 }
26 int comdownlevel(const void *x,const void *y){
27     LogEntry *xx =(LogEntry*)x;
28     LogEntry *yy =(LogEntry*)y;
29     return (yy->level-xx->level);
30
31 }
32 int comuptext(const void *x,const void *y){
33     LogEntry *xx =(LogEntry*)x;
34     LogEntry *yy =(LogEntry*)y;
35     return strcmp((xx)->text, (yy)->text);
36
37 }
38 int comdowntext(const void *x,const void *y){
39     LogEntry *xx =(LogEntry*)x;
40     LogEntry *yy =(LogEntry*)y;
41     return strcmp((yy)->text, (xx)->text);
42
43 }
44
45 void outputLogArray(LogEntry* array, int size) {
46     printf("\n\n");
47     for (int i = 0; i < size; i++) {
48         printf("id: %d, level: %d, text: %s\n", array[i].id, array[i].level, array[i].text);
49     }
50     printf("\n");
51 }
52 void print_updown(){
53     printf("1) UP\n");
54     printf("2) Down\n");
55
56 }
57 void gnomeSort(LogEntry* *array, int size,int op,int ap) {
58     int index = 0;
59     LogEntry temp;
60
61     switch (ap){
```

```

62     switch (ap){
63         case 1:
64             //UP
65             switch (op){
66                 case 1:
67                     //id
68                     while (index < size) {
69                         if (index == 0 || (*array)[index].id >= (*array)[index - 1].id) {
70                             index++;
71                         } else {
72                             temp=(*array)[index];
73                             (*array)[index]=(*array)[index-1];
74                             (*array)[index-1]=temp;
75
76                             index--;
77                         }
78                     }
79                     break;
80                 case 2:
81                     //varn
82                     while (index < size) {
83                         if (index == 0 || (*array)[index].level >= (*array)[index - 1].level) {
84                             index++;
85                         } else {
86                             temp=(*array)[index];
87                             (*array)[index]=(*array)[index-1];
88                             (*array)[index-1]=temp;
89
90                             index--;
91                         }
92                     }
93                     break;
94                 case 3:
95                     //txt
96                     while (index < size) {
97                         if (index == 0 || strcmp((*array)[index].text, (*array)[index-1].text) < 0) {
98                             index++;
99                         } else {
100                             temp=(*array)[index];
101                             (*array)[index]=(*array)[index-1];
102                             (*array)[index-1]=temp;
103
104                             index--;
105                         }
106                     }
107                     break;
108                 default:
109                     break;
110             }
111             break;
112         case 2:
113             //DOWN
114             switch (op){
115                 case 1:
116                     //id
117                     while (index < size) {
118                         if (index == 0 || (*array)[index].id <= (*array)[index - 1].id) {
119                             index++;
120                         } else {
121                             temp=(*array)[index];
122                             (*array)[index]=(*array)[index-1];

```

```

122                         (*array)[index]=(*array)[index-1];
123                         (*array)[index-1]=temp;
124
125                         index--;
126                     }
127                 }
128             break;
129         case 2:
130             //varn
131             while (index < size) {
132                 if (index == 0 || (*array)[index].level <= (*array)[index - 1].level) {
133                     index++;
134                 } else {
135                     temp=(*array)[index];
136                     (*array)[index]=(*array)[index-1];
137                     (*array)[index-1]=temp;
138
139                     index--;
140                 }
141             }
142             break;
143         case 3:
144             //txt
145             while (index < size) {
146                 if (index == 0 || (strcmp((*array)[index].text, (*array)[index-1].text) > 0)) {
147                     index++;
148                 } else {
149                     temp=(*array)[index];
150                     (*array)[index]=(*array)[index-1];
151                     (*array)[index-1]=temp;
152
153                     index--;
154                 }
155             }
156         }
157     break;
158     default:
159         break;
160     }
161 }
162 }
163 }
164
165
166
167
168 void pairInsertionSort(LogEntry* *arr, int n,int op,int ap) {
169     LogEntry key;
170     LogEntry keym;
171     int p;
172
173
174     switch (ap){
175         case 1:
176             //UP
177             switch (op){
178                 case 1:
179                     //id
180                     for (int i = 1; i < n; i += 2) {

```

```

180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
    for (int i = 1; i < n; i += 2) {
        if((*arr)[i].id>(*arr)[i-1].id){
            keym=(*arr)[i];
            key=(*arr)[i-1];
        }else{
            key=(*arr)[i];
            keym=(*arr)[i-1];
        }
        p=i;
        while(p-2>=0 && (*arr)[p-2].id>keym.id){
            (*arr)[p]=(*arr)[p-2];
            p--;
        }
        (*arr)[p]=keym;
        while(p-1>0 && (*arr)[p-1].id>key.id){
            (*arr)[p-1]=(*arr)[p-2];
            p--;
        }
        (*arr)[p-1]=key;
    }
    p=n-1;
    key=(*arr)[p];
    while(p>0 && (*arr)[p-1].id>key.id){
        (*arr)[p]=(*arr)[p-1];
        p--;
    }
}
break;
case 2:
//varn
    for (int i = 1; i < n; i += 2) {
        if((*arr)[i].level>(*arr)[i-1].level){
            keym=(*arr)[i];
            key=(*arr)[i-1];
        }else{
            key=(*arr)[i];
            keym=(*arr)[i-1];
        }
        p=i;
        while(p-2>=0 && (*arr)[p-2].level>keym.level){
            (*arr)[p]=(*arr)[p-2];
            p--;
        }
        (*arr)[p]=keym;
        while(p-1>0 && (*arr)[p-1].level>key.level){
            (*arr)[p-1]=(*arr)[p-2];
            p--;
        }
        (*arr)[p-1]=key;
    }
    p=n-1;
    key=(*arr)[p];
    while(p>0 && (*arr)[p-1].level>key.level){
        (*arr)[p]=(*arr)[p-1];
        p--;
    }
}
break;
case 3:
//txt
    for (int i = 1; i < n; i += 2) {
        if(strcmp((*arr)[i].text, (*arr)[i-1].text) < 0){
            keym=(*arr)[i];
            key=(*arr)[i-1];
        }else{
            key=(*arr)[i];
            keym=(*arr)[i-1];
        }
        p=i;
        while(p-2>=0 && strcmp((*arr)[p-2].text, keym.text) < 0 ){

```

```

249         while(p-2>=0 && strcmp((*arr)[p-2].text, keym.text) < 0 ){
250             (*arr)[p]=(*arr)[p-2];
251             p--;
252         }
253         (*arr)[p]=keym;
254
255         while(p-1>0 && strcmp((*arr)[p-1].text, key.text) < 0 ){
256             (*arr)[p-1]=(*arr)[p-2];
257             p--;
258         }
259         (*arr)[p-1]=key;
260     }
261     p=n-1;
262     key=(*arr)[p];
263     while(p>0 && strcmp((*arr)[p-1].text, key.text) < 0 ){
264         (*arr)[p]=(*arr)[p-1];
265         p--;
266     }
267     break;
268     default:
269         break;
270     }
271     break;
272 case 2:
273     //DOWN
274     switch (op){
275         case 1:
276             //id
277             for (int i = 1; i < n; i += 2) {
278                 if((*arr)[i].id<(*arr)[i-1].id){
279                     keym=(*arr)[i];
280                     key=(*arr)[i-1];
281                 }else{
282                     key=(*arr)[i];
283                     keym=(*arr)[i-1];
284                 }
285                 p=i;
286                 while(p-2>=0 && (*arr)[p-2].id<keym.id){
287                     (*arr)[p]=(*arr)[p-2];
288                     p--;
289                 }
290                 (*arr)[p]=keym;
291
292                 while(p-1>0 && (*arr)[p-1].id<key.id){
293                     (*arr)[p-1]=(*arr)[p-2];
294                     p--;
295                 }
296                 (*arr)[p-1]=key;
297             }
298             p=n-1;
299             key=(*arr)[p];
300             while(p>0 && (*arr)[p-1].id<key.id){
301                 (*arr)[p]=(*arr)[p-1];
302                 p--;
303             }
304             break;
305         case 2:
306             //varn
307             for (int i = 1; i < n; i += 2) {
308                 if((*arr)[i].level<(*arr)[i-1].level){
309                     keym=(*arr)[i];
310                     key=(*arr)[i-1];
311                 }else{
312                     key=(*arr)[i];
313                     keym=(*arr)[i-1];
314                 }
315                 p=i;
316                 while(p-2>=0 && (*arr)[p-2].level<keym.level){
317                     (*arr)[p]=(*arr)[p-2];
318                     p--;
319                 }

```

```

319 }
320 (*arr)[p]=keym;
321
322     while(p-1>0 && (*arr)[p-1].level<key.level){
323         (*arr)[p-1]=(*arr)[p-2];
324         p--;
325     }
326     (*arr)[p-1]=key;
327 }
328 p=n-1;
329 key=(*arr)[p];
330 while(p>0 && (*arr)[p-1].level<key.level){
331     (*arr)[p]=(*arr)[p-1];
332     p--;
333 }
334 break;
335 case 3:
336     //txt
337     for (int i = 1; i < n; i += 2) {
338         if(strcmp((*arr)[i].text, (*arr)[i-1].text) > 0){
339             keym=(*arr)[i];
340             key=(*arr)[i-1];
341         }else{
342             key=(*arr)[i];
343             keym=(*arr)[i-1];
344         }
345         p=i;
346         while(p-2>=0 && strcmp((*arr)[p-2].text, keym.text) > 0 ){
347
348             (*arr)[p]=(*arr)[p-2];
349             p--;
350
351         }
352     }
353     (*arr)[p]=keym;
354
355
356
357     while(p-1>0 && strcmp((*arr)[p-1].text, key.text) > 0 ){
358         (*arr)[p-1]=(*arr)[p-2];
359         p--;
360     }
361     (*arr)[p-1]=key;
362 }
363 p=n-1;
364 key=(*arr)[p];
365 while(p>0 && strcmp((*arr)[p-1].text, key.text) > 0 ){
366     (*arr)[p]=(*arr)[p-1];
367     p--;
368 }
369 break;
370 default:
371     break;
372 }
373 break;
374 default:
375     break;
376 }
377
378 (*arr)[p]=key;
379 }
380
381
382 int getRandomNumber(int min, int max) {
383     return min + rand() % (max - min + 1);
384 }
385
386 int getRandomLevel() {
387     int level = getRandomNumber(1, 5);
388     return level;
389 }

```

```
390
391 void getRandomMessage(char *message) {
392     char letters[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
393     int length = getRandomNumber(1, 50);
394     int i;
395     for (i = 0; i < length; i++) {
396         message[i] = letters[getRandomNumber(0, strlen(letters) - 1)];
397     }
398     message[i] = '\0';
399 }
400 void print_sort(){
401     printf("\n1) Gnome\n");
402     printf("2) Two pair\n");
403     printf("3) qsort\n");
404 }
405 void print_h(){
406     printf("\n1) id\n");
407     printf("2) varn\n");
408     printf("3) txt\n");
409 }
410
411
412
```

```
[kuksa.iv@unix lab5]$ gcc lab51.c array1.c -o lab51
```

Запуск и сборка программы  
lab51

```
1) Input
2) Output
3) Sort
4) Stop
Enter: 1

1) From std key
2) file
3) bin file
Enter: 1
Enter the size of the log array: 5
Enter id: 2
Enter level (0-debug, 1-info, 2-warn, 3-error, 4-fatal): 3
Enter text: aa
Enter id: 2
Enter level (0-debug, 1-info, 2-warn, 3-error, 4-fatal): 4
Enter text: bb
Enter id: 55
Enter level (0-debug, 1-info, 2-warn, 3-error, 4-fatal): 0
Enter text: aaa
Enter id: 34
Enter level (0-debug, 1-info, 2-warn, 3-error, 4-fatal): 2
Enter text: ww
Enter id: 12
Enter level (0-debug, 1-info, 2-warn, 3-error, 4-fatal): 2
Enter text: text

1) Input
2) Output
3) Sort
4) Stop
Enter: 3

1) Gnome
2) Two pair
3) qsort
Enter: 1

1) id
2) varn
3) txt
Enter: 1

1) UP
2) Down
Enter: 2

1) Input
2) Output
3) Sort
4) Stop
Enter: 2

1) From std key
2) file
3) bin file
Enter: 1
```

```
id: 55, level: 0, text: aaa
id: 34, level: 2, text: ww
id: 12, level: 2, text: text
id: 2, level: 3, text: aa
id: 2, level: 4, text: bb
```

```
1) Input
2) Output
3) Sort
4) Stop
Enter: 3
```

```
1) Gnome
2) Two pair
3) qsort
Enter: 2
```

```
1) id
2) varn
3) txt
Enter: 3
```

```
1) UP
2) Down
Enter: 2
```

```
1) Input
2) Output
3) Sort
4) Stop
Enter: 2
```

```
1) From std key
2) file
3) bin file
Enter: 1
```

```
id: 2, level: 3, text: aa
id: 55, level: 0, text: aaa
id: 2, level: 4, text: bb
id: 12, level: 2, text: text
id: 34, level: 2, text: ww
```

```
==4081== HEAP SUMMARY:
==4081==     in use at exit: 0 bytes in 0 blocks
==4081==   total heap usage: 5 allocs, 5 frees, 2,804 bytes allocated
==4081==
==4081== All heap blocks were freed -- no leaks are possible
==4081==
==4081== For lists of detected and suppressed errors, rerun with: -s
==4081== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
[kuksa.iv@unix lab5]$ gcc lab52.c array2.c -o lab52
```

```
0.000000
0) Начать сортировать
1) Стоп
ENTER: 0
Введите количество массивов: 100
Введите количество структур в массивах: 23
```

```
1) Gnome
2) Two pair
3) qsort
Enter: 1
```

```
1) id
2) varn
3) txt
Enter: 2
```

```
1) UP
2) Down
Enter: 1
```

```
3.000000
0) Начать сортировать
1) Стоп
ENTER: 0
Введите количество массивов: 1000
Введите количество структур в массивах: 234
```

```
1) Gnome
2) Two pair
3) qsort
Enter: 1
```

```
1) id
2) varn
3) txt
Enter: 1
```

```
1) UP
2) Down
Enter: 1
```

```
397.000000
0) Начать сортировать
1) Стоп
ENTER: 0
Введите количество массивов: 1000
Введите количество структур в массивах: 2345
```

```
==9875==  
==9875== HEAP SUMMARY:  
==9875==     in use at exit: 0 bytes in 0 blocks  
==9875==   total heap usage: 4 allocs, 4 frees, 3,236 bytes allocated  
==9875==  
==9875== All heap blocks were freed -- no leaks are possible  
==9875==  
==9875== Use --track-origins=yes to see where uninitialized values come from  
==9875== For lists of detected and suppressed errors, rerun with: -s
```

Запуск и сборка программы  
lab52

```
0) Начать сортировать
1) Стоп
ENTER: 0
Введите количество массивов: 1000
Введите количество структур в массивах: 2345
```

```
1) Gnome
2) Two pair
3) qsort
Enter: 3
```

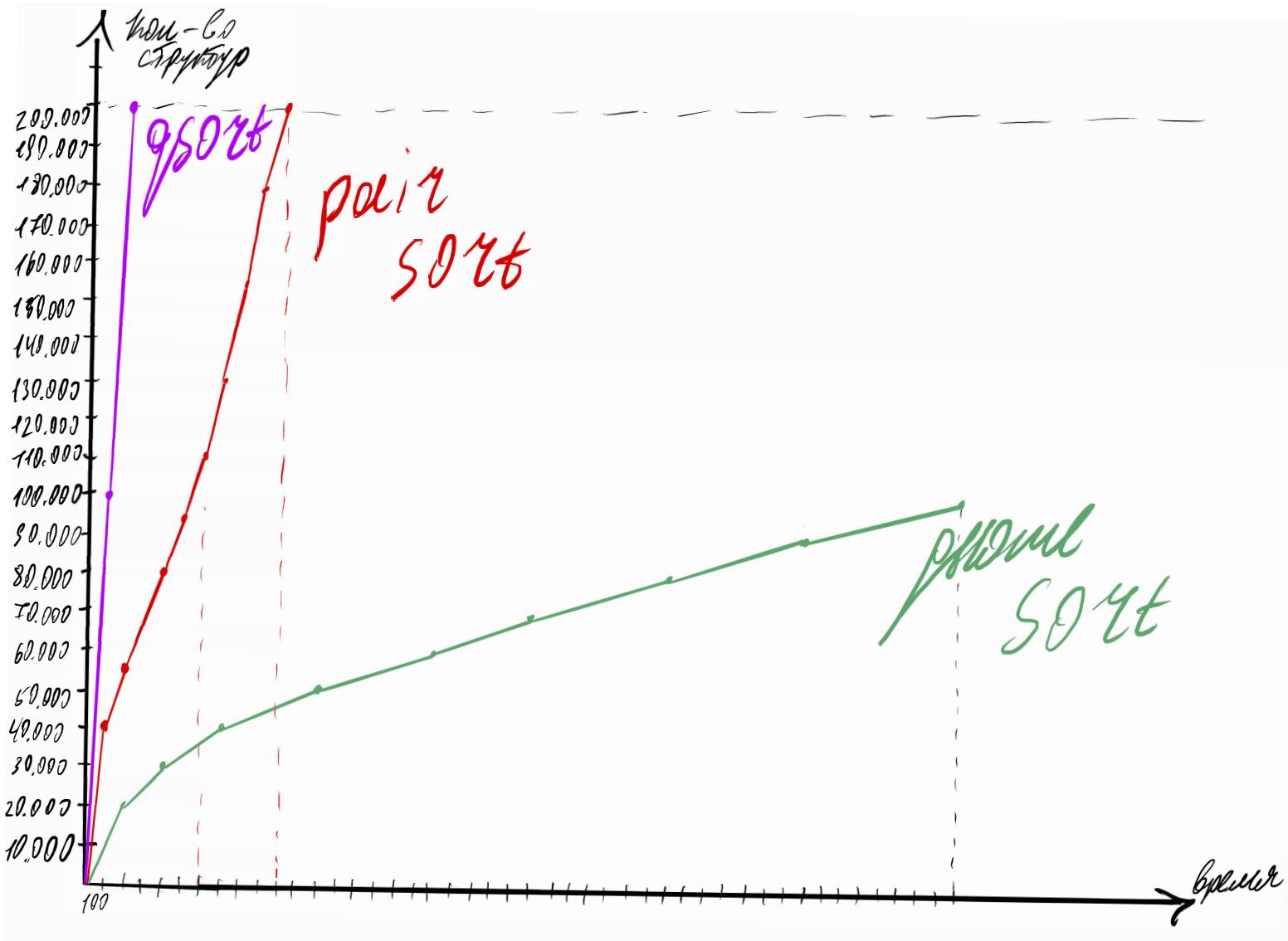
```
1) id
2) varn
3) txt
Enter: 2
```

```
1) UP
2) Down
Enter: 2
```

1692.000000

```
0) Начать сортировать
1) Стоп
ENTER: █
```

кол-во структур	вид сортировки	поле	по возрастанию; по убыванию	время выполнения, мс
1000	Гномья	id	по возрастанию	3
10000	Гномья	level	по возрастанию	7
1000	Гномья	text	по возрастанию	13
100000	Гномья	level	по возрастанию	70
1000000	Гномья	id	по убыванию	707
Парная сортировка				
1000	вставкой	id	по убыванию	14
Парная сортировка				
10000	вставкой	level	по возрастанию	1
Парная сортировка				
1000000	вставкой	id	по возрастанию	684
Парная сортировка				
100000	вставкой	text	по возрастанию	84
10000	qsort	text	по возрастанию	1
100000	qsort	id	по убыванию	81
100000	qsort	level	по возрастанию	84
1000000	qsort	id	по убыванию	780



## 7. Выводы

В ходе выполнения данной работы на примере программы, выполняющей работу со структурами были реализованы следующие моменты:

1. Массив структур
2. 3 вида сортировок по полям структур
3. Запись в текстовый и бинарный файл