

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное автономное образовательное
учреждение
высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»**

Московский институт электроники и математики им. А.Н. Тихонова

Департамент прикладной математики

**Отчёт
по лабораторной работе №8
по курсу «Алгоритмизация и программирование»
Задание № 13**

ФИО студента	Номер группы	Дата
Кейер Александр Петрович	БПМ-231	3 марта 2024 г.

Москва, 2023

1 Задание (вариант № 13)

1. Реализовать указанные в варианте алгоритмы сортировки для массива объектов в соответствии с вариантом. Структура объекта описана в варианте ЛР8. Определить функции сравнения объектов по следующему принципу: приоритет сравнения определяется порядком поля в структуре, т.е. если структурный тип X содержит (в описании варианта) поля с именами a, b, c, d , то при сравнении двух объектов типа X надо сравнить поля с именем a , в случае их равенства сравнить поля b и т.д.
2. Каждый алгоритм сортировки должен поддерживать сортировку в обоих направлениях: по неубыванию и по невозрастанию.
3. Выбор и запуск требуемого алгоритма и направления сортировки осуществляется через меню на этапе выполнения.
4. Провести сортировку каждым алгоритмом массивов следующих размеров: 100, 1000, 5000, 10000, 20000, 50000, 100000. Засечь (программно) время сортировки каждым алгоритмом. По полученным точкам построить графики зависимости времени сортировки от размерности массива для каждого из алгоритмов сортировки на одной оси координат. Полученные графики включить в отчет к работе.

Реализовать сортировки:

- В – вставками
- D – шейкер
- E – слиянием

2 Структура заголовочного файла

```
1  #define fieldLength 64
2  #define fieldSize fieldLength * sizeof(char)
3  #define entryLength 6
4
5  struct footballerType {
6      char fullName[fieldLength];
7      char clubName[fieldLength];
8      char role[fieldLength];
9      int age;
10     int numberOfGames;
11     int numberOfGoals;
12 };
13
```

3 Решение

```
1  #include "football.h" // Footballer info.
2
3  #include <stdio.h> // Input/output library.
4  #include <stdlib.h> // Memory allocation.
5  #include <time.h> // Time library.
6  #include <assert.h> // Assertion library.
7  #include <string.h> // String functions library.
8
9  // Function getting direction of sorting by user.
10 void getSortDirectionByUser(int* direction) {
11     printf("Enter sort direction (1 - increasing, -1 -
12     decreasing): ");
13
14     fflush(stdin); // Clear stdin flow.
15     scanf("%d", direction);
16 }
17
18 // Function printing horizontal line.
19 void printHr(int length) {
20     for (int j = 0; j < length; j++) {
21         printf("- ");
22     }
23     printf("\n");
24 }
```

```

25 // Function printing table header.
26 void printTableHeader() {
27     printHr(65);
28     printf("%4s|%30s|%30s|%30s|%10s|%10s|%10s|\n", "ID", "
FULL NAME", "CLUB NAME", "ROLE", "AGE", "GAMES", "GOALS");
29     printHr(65);
30 }
31
32 // Function printing footballer.
33 void printFootballer(struct footballerType footballer,
int id) {
34     printf("%4d|%30s|%30s|%30s|%10d|%10d|%10d|\n", id,
footballer.fullName, footballer.clubName, footballer.role,
    footballer.age, footballer.numberOfGames, footballer.
numberOfGoals);
35     printHr(65);
36 }
37
38 // Function printing footballers.
39 void printFootballers(struct footballerType* footballers,
int length) {
40     if (footballers == NULL) {
41         printf("Incorrect array.\n");
42         return;
43     }
44
45     printTableHeader();
46
47     for (int i = 0; i < length; i++) {
48         printFootballer(footballers[i], i);
49     }
50 }
51
52 // Function generating string.
53 char* generateString(int length, int countOfUsedSymbols)
{
54     assert(countOfUsedSymbols <= 26);
55
56     char* out = (char*)malloc(sizeof(char) * (length + 1));
57     char alphabet[26] = {
58         'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'
, 'l', 'm',
59         'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x'
, 'y', 'z'
60     };

```

```

61
62     for (int i = 0; i < length; i++) {
63         out[i] = alphabet[rand() % countOfUsedSymbols];
64     }
65
66     out[length] = '\0';
67
68     return out;
69 }
70
71 // Function generating footballer.
72 struct footballerType generateFootballer() {
73     struct footballerType out = {
74         .fullName=generateString(1, 1),
75         .clubName=generateString(1, 1),
76         .role=generateString(1, 1),
77         .age=(rand() % 100),
78         .numberOfGames=(rand() % 100),
79         .numberOfGoals=(rand() % 100),
80     };
81
82     return out;
83 }
84
85 // Function generating footballer array.
86 struct footballerType* generateFootballersArray(int
length) {
87     struct footballerType* arr = (struct footballerType*)
malloc(sizeof(struct footballerType) * length);
88
89     for (int i = 0; i < length; i++) {
90         arr[i] = generateFootballer();
91     }
92
93     printf("Successfully generated %d footballers array.\n"
, length);
94     return arr;
95 }
96
97 // Function comparing footballers.
98 int compareFootballers(struct footballerType footballer1,
struct footballerType footballer2, int direction) {
99     int cmpFullNames = strcmp(footballer1.fullName,
footballer2.fullName);
100

```

```

101     if (cmpFullNames != 0) {
102         return direction * cmpFullNames;
103     }
104
105     int cmpClubNames = strcmp(footballer1.clubName,
106 footballer2.clubName);
107
108     if (cmpClubNames != 0) {
109         return direction * cmpClubNames;
110     }
111
112     int cmpRole = strcmp(footballer1.role, footballer2.role
113 );
114
115     if (cmpRole != 0) {
116         return direction * cmpRole;
117     }
118
119     if (footballer1.age != footballer2.age) {
120         return direction * (footballer1.age - footballer2.age
121 );
122     }
123
124     if (footballer1.numberOfGames != footballer2.
125 numberOfGames) {
126         return direction * (footballer1.numberOfGames -
127 footballer2.numberOfGames);
128     }
129
130     if (footballer1.numberOfGoals != footballer2.
131 numberOfGoals) {
132         return direction * (footballer1.numberOfGoals -
133 footballer2.numberOfGoals);
134     }
135
136     }
137
138     // Bubble sorting function.
139     void bubbleSort(struct footballerType* arr, int n, char
140 direction) {
141         if (arr == NULL) {
142             printf("Incorrect array.\n");
143             return;
144         }
145
146         struct footballerType tmp;

```

```

138     clock_t start = clock();
139
140     for (int i = n - 1; i >= 0; i--) {
141         for (int j = 0; j < i; j++) {
142             if (compareFootballers(arr[j], arr[j + 1],
direction) > 0) {
143                 tmp = arr[j + 1];
144                 arr[j + 1] = arr[j];
145                 arr[j] = tmp;
146             }
147         }
148     }
149     clock_t stop = clock();
150
151     printf("%5s%6d: %.03fs\n", "n=", n, (double)(stop -
start) / CLOCKS_PER_SEC);
152 }
153
154 // Function sorting by insertions.
155 void insertSort(struct footballerType* arr, int n, char
direction) {
156     if (arr == NULL) {
157         printf("Incorrect array.\n");
158         return;
159     }
160
161     int j;
162     struct footballerType tmp;
163
164     clock_t start = clock();
165
166     for (int i = 1; i < n; i++) {
167         tmp = arr[i];
168
169         for (j = i - 1; j >= 0 && compareFootballers(arr[j],
tmp, direction) > 0; j--) {
170             arr[j + 1] = arr[j];
171         }
172
173         arr[j + 1] = tmp;
174     }
175
176     clock_t stop = clock();
177
178     printf("%5s%6d: %.03fs\n", "n=", n, (double)(stop -

```

```

179     start) / CLOCKS_PER_SEC);
180     }
181     // Shaker sorting function.
182     void shakerSort(struct footballerType* arr, int n, char
183     direction) {
184         if (arr == NULL) {
185             printf("Incorrect array.\n");
186             return;
187         }
188         struct footballerType tmp;
189         int l = 1, r = n;
190         int j;
191
192         clock_t start = clock();
193
194         do {
195             for(j = --r; j >= l; j--) {
196                 if (compareFootballers(arr[j - 1], arr[j],
197                 direction) > 0) {
198                     tmp = arr[j - 1];
199                     arr[j - 1] = arr[j];
200                     arr[j] = tmp;
201                 }
202             }
203             for (j = ++l; j <= r; j++) {
204                 if (compareFootballers(arr[j - 1], arr[j],
205                 direction) > 0) {
206                     tmp = arr[j - 1];
207                     arr[j - 1] = arr[j];
208                     arr[j] = tmp;
209                 }
210             } while (l < r);
211
212         clock_t stop = clock();
213
214         printf("%5s%6d: %.03fs\n", "n=", n, (double)(stop -
215         start) / CLOCKS_PER_SEC);
216     }
217
218     // Functio mergein two arrays.
219     void merge(struct footballerType* arr, int l, int m, int

```



```

219     r, char direction) {
220         int i = l;
221         int j = m + 1;
222         int k = 0;
223
224         struct footballerType* tmp = (struct footballerType*)
225         malloc(sizeof(struct footballerType) * (r - l + 1));
226
227         while (i <= m && j <= r) {
228             if (compareFootballers(arr[i], arr[j], direction) >=
229             0) {
230                 tmp[k] = arr[j];
231                 j++;
232             } else {
233                 tmp[k] = arr[i];
234                 i++;
235             }
236
237             k++;
238         }
239
240         if (i > m) {
241             while (j <= r) {
242                 tmp[k] = arr[j];
243                 j++;
244                 k++;
245             }
246         } else {
247             while (i <= m) {
248                 tmp[k] = arr[i];
249                 i++;
250                 k++;
251             }
252         }
253
254         for (k = 0; k <= r - l; k++) {
255             arr[l + k] = tmp[k];
256         };
257
258         free(tmp); // Free allocated memory.
259     }
260
261     // Function splitting and merging sorting array.

```

```

261 void splitAndMerge(struct footballerType* arr, int l, int
262 r, char direction) {
263     if (l < r) {
264         int m = (l + r) / 2;
265         splitAndMerge(arr, l, m, direction);
266         splitAndMerge(arr, m + 1, r, direction);
267         merge(arr, l, m, r, direction);
268     }
269 }
270 // Merge sorting function.
271 void mergeSort(struct footballerType* arr, int n, char
272 direction) {
273     if (arr == NULL) {
274         printf("Incorrect array.\n");
275         return;
276     }
277     clock_t start = clock();
278     splitAndMerge(arr, 0, n - 1, direction);
279     clock_t stop = clock();
280
281     printf("%5s%6d: %.03fs\n", "n=", n, (double)(stop -
282 start) / CLOCKS_PER_SEC);
283 }
284 // Function printing main operations codes list.
285 void printMainMenuOperationsList() {
286     printf("\n");
287     printf("%30s %3s", "generate array:", "1\n");
288     printf("%30s %3s", "sort by insertions:", "2\n");
289     printf("%30s %3s", "bubble sort:", "3\n");
290     printf("%30s %3s", "shaker sort:", "4\n");
291     printf("%30s %3s", "merge sort:", "5\n");
292     printf("%30s %3s", "print array:", "6\n");
293     printf("%30s %3s", "exit program:", "7\n");
294     printf("\n\n");
295 }
296
297 // Function starting main menu.
298 void startMainMenu(struct footballerType* arr, int *pn) {
299     printf("\n");
300     printMainMenuOperationsList();
301
302     int operationCode = 0;

```

```

303     int direction = 1;
304
305     printf("Enter correct operation code: ");
306
307     fflush(stdin); // Clear stdin flow.
308     scanf("%d", &operationCode);
309
310     switch (operationCode) {
311         case 1: {
312             printf("Enter footballers count: ");
313
314             fflush(stdin);
315             scanf("%d", pn);
316
317             arr = generateFootballersArray(*pn);
318
319             break;
320         }
321
322         case 2: {
323             getSortDirectionByUser(&direction);
324             insertSort(arr, *pn, direction);
325             break;
326         }
327
328         case 3: {
329             getSortDirectionByUser(&direction);
330             bubbleSort(arr, *pn, direction);
331             break;
332         }
333
334         case 4: {
335             getSortDirectionByUser(&direction);
336             shakerSort(arr, *pn, direction);
337             break;
338         }
339
340         case 5: {
341             getSortDirectionByUser(&direction);
342             mergeSort(arr, *pn, direction);
343             break;
344         }
345
346         case 6: {
347             printFootballers(arr, *pn);

```

```

348         break;
349     }
350
351     case 7: {
352         return;
353     }
354 }
355
356 printf("\n");
357 startMainMenu(arr, pn);
358 }
359
360 // Function running tests.
361 void runTests() {
362     printf("Sort by insertions.\n\n");
363
364     insertSort(generateFootballersArray(100), 100, 1);
365     insertSort(generateFootballersArray(1000), 1000, 1);
366     insertSort(generateFootballersArray(5000), 5000, 1);
367     insertSort(generateFootballersArray(10000), 10000, 1);
368     insertSort(generateFootballersArray(20000), 20000, 1);
369     insertSort(generateFootballersArray(50000), 50000, 1);
370     insertSort(generateFootballersArray(100000), 100000, 1)
371 ;
372
373     printf("\n-----\n\n");
374
375     printf("Bubble sort.\n\n");
376
377     bubbleSort(generateFootballersArray(100), 100, 1);
378     bubbleSort(generateFootballersArray(1000), 1000, 1);
379     bubbleSort(generateFootballersArray(5000), 5000, 1);
380     bubbleSort(generateFootballersArray(10000), 10000, 1);
381     bubbleSort(generateFootballersArray(20000), 20000, 1);
382     bubbleSort(generateFootballersArray(50000), 50000, 1);
383     bubbleSort(generateFootballersArray(100000), 100000, 1)
384 ;
385
386     printf("\n-----\n\n");
387
388     printf("Shaker sort.\n\n");
389
390     shakerSort(generateFootballersArray(100), 100, 1);
391     shakerSort(generateFootballersArray(1000), 1000, 1);
392     shakerSort(generateFootballersArray(5000), 5000, 1);

```

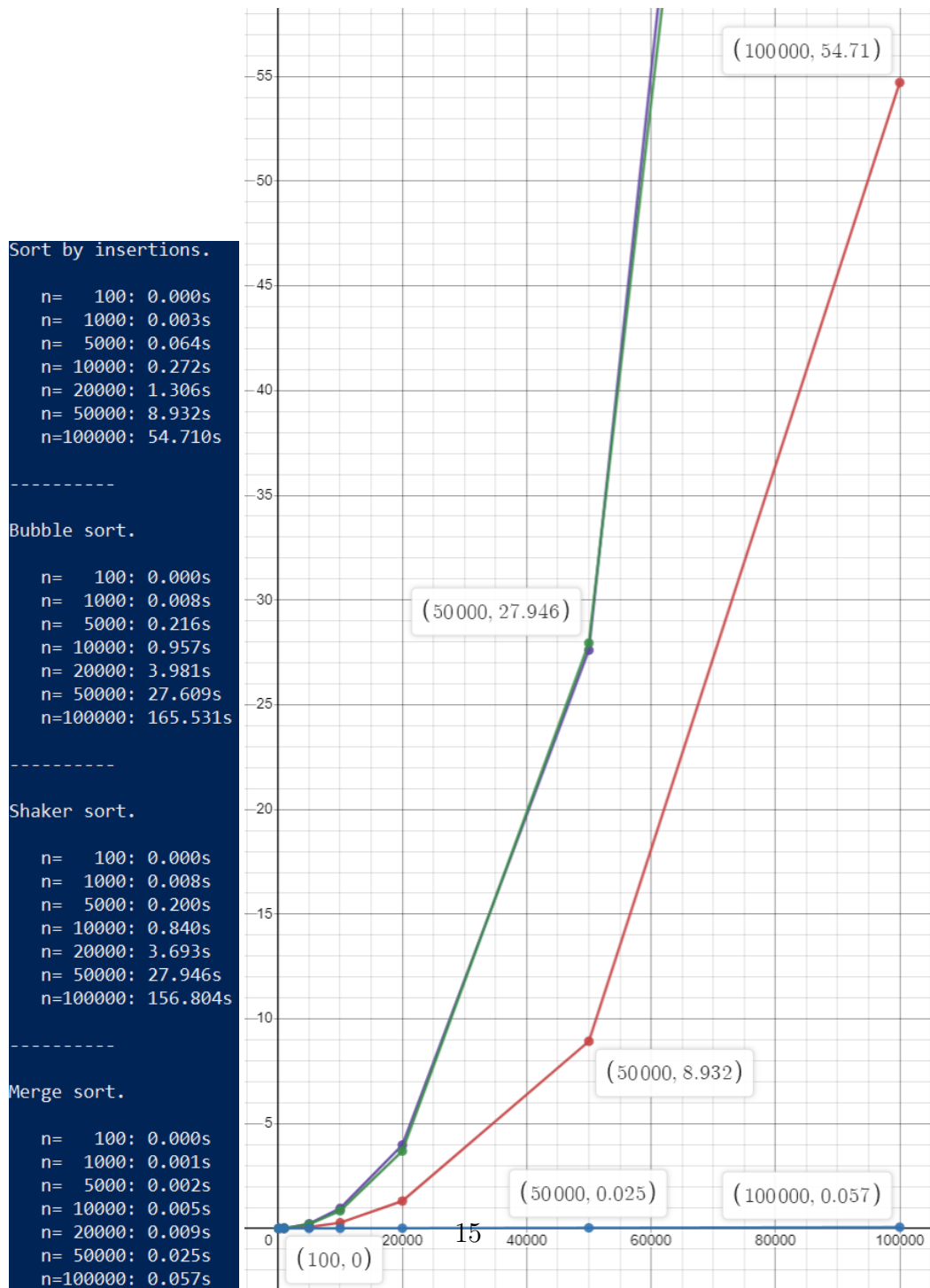
```

391     shakerSort(generateFootballersArray(10000), 10000, 1);
392     shakerSort(generateFootballersArray(20000), 20000, 1);
393     shakerSort(generateFootballersArray(50000), 50000, 1);
394     shakerSort(generateFootballersArray(100000), 100000, 1)
;
395
396     printf("\n-----\n\n");
397
398     printf("Merge sort.\n\n");
399
400     mergeSort(generateFootballersArray(100), 100, 1);
401     mergeSort(generateFootballersArray(1000), 1000, 1);
402     mergeSort(generateFootballersArray(5000), 5000, 1);
403     mergeSort(generateFootballersArray(10000), 10000, 1);
404     mergeSort(generateFootballersArray(20000), 20000, 1);
405     mergeSort(generateFootballersArray(50000), 50000, 1);
406     mergeSort(generateFootballersArray(100000), 100000, 1);
407 }
408
409 int main() {
410     srand(time(NULL)); // Init first random number.
411
412     struct footballerType* arr = NULL;
413     int pn;
414
415     startMainMenu(arr, &pn);
416     // runTests();
417
418     return 0;
419 }
420

```


4 Тесты

4.1 Тест 1 (графики и оценка эффективности)



По времени эффективнее всех оказывается сортировка слиянием. Даль-

ше по порядку идут: сортировка вставками, сортировка шейкером, пузырьковая сортировка. Естественно, по затратам памяти самой неэффективной оказывается сортировка слиянием, но затраты по памяти полностью компенсируются временными.

4.2 Тест 2

```
generate array: 1
sort by insertions: 2
bubble sort: 3
shaker sort: 4
merge sort: 5
print array: 6
exit program: 7

Enter correct operation code: 1
Enter footballers count: 100
Successfully generated 100 footballers array.
```

Успешная генерация массива футболистов соответствующего размера.

4.3 Тест 3

```
Enter correct operation code: 6
```

ID	FULL NAME	CLUB NAME	ROLE	AGE	GAMES	GOALS
0	a	a	a	25	81	71
1	a	a	a	53	54	16
2	a	a	a	29	74	2
3	a	a	a	87	69	94
4	a	a	a	43	40	3
5	a	a	a	1	40	96
6	a	a	a	93	13	77
7	a	a	a	96	25	72
8	a	a	a	83	77	18
9	a	a	a	61	76	82

Успешный вывод массива футболистов соответствующего размера.

4.4 Тест 4

```
Enter correct operation code: 5
Enter sort direction (1 - increasing, -1 - decreasing): -1
n= 10: 0.000s

generate array: 1
sort by insertions: 2
bubble sort: 3
shaker sort: 4
merge sort: 5
print array: 6
exit program: 7
```

```
Enter correct operation code: 6
```

ID	FULL NAME	CLUB NAME	ROLE	AGE	GAMES	GOALS
0	a	a	a	96	25	72
1	a	a	a	93	13	77
2	a	a	a	87	69	94
3	a	a	a	83	77	18
4	a	a	a	61	76	82
5	a	a	a	53	54	16
6	a	a	a	43	40	3
7	a	a	a	29	74	2
8	a	a	a	25	81	71
9	a	a	a	1	40	96

Успешная сортировка массива футболистов в соответствующем направлении.