

**Лабораторная работа №2**  
**«АЛГОРИТМЫ ПОИСКА»**  
по дисциплине "Методы программирования"

Выполнил Ковалев Даниил  
СКБ171, вариант 12  
МИЭМ НИУ ВШЭ

В данной лабораторной проведено сравнение следующих алгоритмов поиска:

- Собственноручно реализованный линейный поиск. Асимптотическая сложность  $O(n)$ .
- Собственноручно реализованный бинарный поиск на отсортированном массиве. Асимптотическая сложность  $O(\log n)$ .
- Сортировка произвольного массива функцией `my::quick_sort` (сложность  $O(n \log n)$ ) и последующий двоичный поиск в отсортированном массиве собственной функцией (сложность  $O(\log n)$ ). Суммарная асимптотическая сложность  $O(n \log n)$ .
- Бинарный поиск из стандартной библиотеки C++ на отсортированном массиве. Асимптотическая сложность  $O(\log n)$ .
- Сортировка произвольного массива функцией `std::sort` (сложность  $O(n \log n)$ ) и последующий двоичный поиск в отсортированном массиве стандартной функцией (сложность  $O(\log n)$ ). Суммарная асимптотическая сложность  $O(n \log n)$ .

Алгоритмы применяются для поиска в массиве элементов **Entry** — данных о футбольных командах, принимающих участие в чемпионате страны по футболу: страна, название клуба, город, год, ФИО главного тренера (сравнение по полям — год, страна количество набранных очков (по убыванию), название клуба).

Считывание данных в программе происходит либо из csv-файла с заголовком `country;city;club;trainer;year;score`, либо из базы данных SQLite с таблицей `entries` и полями `country` (TEXT), `city` (TEXT), `club` (TEXT), `trainer` (TEXT), `year` (INTEGER), `score` (INTEGER).

Схема запуска: запускать нужно `python run.py`, он запустит генерацию массива из 100000 элементов типа **Entry** и положит их базу данных `data.sqlite`:

```
1 ../generate_data/generate_data --size=100000 --output=data.sqlite
```

После этого необходимо запустить тестирование алгоритмов сортировок, взяв в качестве данных сгенерированный `data.sqlite`; результаты тестирования положить в `results.csv`. Размеры массивов, на которых тестируются сортировки, берутся из `sizes.txt`, и являются равномерно распределенными на логарифмической оси от 100 до 100000.

```
1 ./2_test_search --input=data.sqlite --sizes=sizes.txt --output=results.csv 2> test_search.log
```

После этого средствами пакета `matplotlib` для `python` на основе `results.csv` строится график зависимости времени выполнения сортировки от размера массива данных в логарифмических осях.

Структура проекта:

📁 3rd\_party — см. ЛР1

📁 entry — см. ЛР1

📁 generate\_data — см. ЛР1

📁 helpers — см. ЛР1

📁 lab2:

📄 CMakeLists.txt

📄 binary\_search.h — файл с реализацией пирамидальной сортировки

📄 linear\_search.h — файл с реализацией быстрой сортировки

📄 main.cpp — файл с реализацией тестирования сортировок

📄 run.py — скрипт запуска генерации данных и тестирования алгоритмов поиска

📄 CMakeLists.txt — см. ЛР1

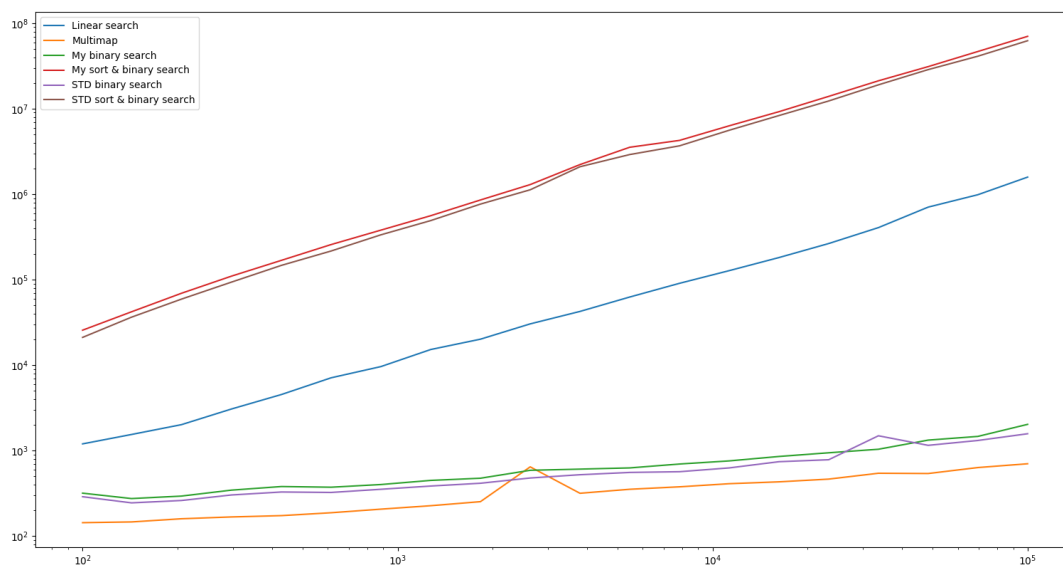
Приведем таблицу, построенную по файлу **results.csv**. В ней записано время работы каждого из шести сравниваемых алгоритмов в наносекундах на диапазонах данных размерами от 100 до 100000.

	100	143	206	297	428	615	885
Линейный поиск	1196	1541	2008	3063	4526	7096	9602
Поиск в multimap	143	146	159	167	173	187	206
Свой бин. поиск	317	274	293	344	379	372	400
Своя сорт-ка + свой бин. поиск	25609	42038	69258	109917	168303	257275	379644
Станд. бин. поиск	288	244	260	302	327	323	352
Станд. сорт-ка + станд. бин. поиск	21085	36370	59160	93574	146997	216260	334569
1274	1832	2636	3792	5455	7847	11288	
15211	20091	30351	42446	62571	90530	127413	
226	252	644	316	352	376	409	
447	474	587	606	626	695	756	
560224	855431	1295677	2220675	3542074	4261243	6315128	
384	414	477	521	554	565	626	
491055	765611	1126244	2094916	2910047	3679666	5601570	
16237	23357	33598	48329	69519	100000		
181268	264518	406314	705764	986222	1585244		
430	463	542	538	632	700		
852	941	1035	1324	1463	2025		
9238104	13956133	21230365	31115164	46791801	70387533		
740	780	1488	1150	1311	1572		
8322557	12328059	19046869	28706949	41216476	62557612		

Из полученного графика можно сделать следующие выводы:

- собственные реализации функций работают практически такое же время, как и стандартные
- поиск в `std::multimap` по порядку времени работы совпадает с двоичным поиском на отсортированном массиве, но имеет меньшую константу и работает быстрее
- данные о времени работы алгоритмов согласуются с теоретическими оценками асимптотической сложности

собственные реализации функций работают практически порядок времени работы пирамидальной и быстрой сортировки совпадает и отличается константой (у пирамидальной она больше), а порядок времени работы шейкер-сортировки больше. Это согласуется с теоретическими данными.



Листинги с исходным кодом всех файлов расположены на следующих страницах отчета.

### Листинг 1: lab2/CMakeLists.txt

```
1 set(PROJECT_NAME 2_test_search)
2
3 project(${PROJECT_NAME} LANGUAGES CXX)
4
5 set(SOURCES main.cpp)
6 set(HEADERS binary_search.h
7       linear_search.h)
8
9 add_executable(${PROJECT_NAME} ${SOURCES} ${HEADERS})
10
11 target_include_directories(${PROJECT_NAME} PRIVATE ${Helpers_INCLUDE_DIR} ${
12     Entry_INCLUDE_DIR})
13 target_link_libraries(${PROJECT_NAME} PRIVATE entry)
14 target_link_libraries(${PROJECT_NAME} PRIVATE helpers)
15 target_link_libraries(${PROJECT_NAME} PRIVATE Boost::program_options)
16
17 file(COPY run.py DESTINATION ${PROJECT_BINARY_DIR})
```

### Листинг 2: lab2/binary\_search.h

```
1 /**
2  * @file
3  * @brief Заголовочный файл, содержащий реализацию бинарного поиска
4  * @date Январь 2020
5  */
6 #ifndef BINARY_SEARCH_H
7 #define BINARY_SEARCH_H
8
9 #include <iterator>
10 #include <functional>
11 #include <type_traits>
12
13 namespace my
14 {
15
16     namespace
17     {
18
19         // helper aliases
20         template<typename Iterator>
21         using elem_type = typename std::iterator_traits<Iterator>::value_type;
22
23         template<typename Iterator>
24         using diff_t = typename std::iterator_traits<Iterator>::difference_type;
25
26         // Iterator if functions can be invoked in a right way, nothing otherwise
27         template<typename Iterator, typename Key, typename KeyExtractor>
28         using check_key_extractor = typename std::enable_if<
29             std::is_invocable_r<const Key&, KeyExtractor, const elem_type<Iterator>&>::value,
30             Iterator>::type;
31
32         template<typename Iterator, typename Comparator>
33         using check_comparator = typename std::enable_if<
34             std::is_invocable_r<bool, Comparator, const elem_type<Iterator>&, const elem_type<
35             Iterator>&>::value,
36             Iterator>::type;
37
38         template<typename Iterator, typename Key, typename Comparator, typename KeyExtractor>
39         using check_comparator_and_key_extractor = typename std::enable_if<
40             std::is_invocable_r<const Key&, KeyExtractor, const elem_type<Iterator>&>::value
41             &&
42             std::is_invocable_r<bool, Comparator, const Key&, const Key&>::value,
43             Iterator>::type;
44
45         template<typename Iterator>
46         const elem_type<Iterator>& trivial_extractor(const elem_type<Iterator>& elem)
47         {
```

```

46     return elem;
47 }
48
49 }
50
51 // lower_bound
52
53 /**
54  * Ищет в отсортированном диапазоне первый элемент, не меньший заданного
55  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
56  * @tparam Key тип элемента, с которым будет производиться сравнение
57  * @tparam Comparator бинарный предикат
58  * @tparam KeyExtractor тип, объект которого может быть вызван с аргументом типа,
59  * на который указываем Iterator, и возвращающий Key
60  * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
61   дет производиться поиск
62  * @param[in] key элемент, по которому производится поиск
63  * @param[in] cmp компаратор: возвращает 'true', если его первый аргумент должен стоять
64  * в отсортированном диапазоне строго левее второго, 'false' иначе
65  * @param[in] extractor функция, возвращающая по объекту значение, которое будет использ-
66   оваться
67  * при сравнении с 'key': например, 'Iterator' ссылается на диапазон пар чисел,
68  * отсортированных сначала по первому элементу пары, затем по второму, и требуется найти
69  * первую пару, первый
70  * элемент которой не меньше 'key'. В этом случае эта функция должна по паре возвращать
71  * ее первый элемент.
72  * Необходимо, чтобы 'extractor' сохранял свойство отсортированности диапазона
73  * @return итератор на первый элемент диапазона, значение 'extractor'
74  * от которого не меньше 'key'; 'end', если такого не нашлось
75  */
76 template<typename Iterator, typename Key, typename Comparator, typename KeyExtractor>
77 auto lower_bound(Iterator begin, Iterator end, const Key& key, Comparator cmp,
78                 KeyExtractor extractor) ->
79     check_comparator_and_key_extractor<Iterator, Key, Comparator, KeyExtractor>
80 {
81     diff_t<Iterator> length = std::distance(begin, end), half;
82     while (length > 0)
83     {
84         Iterator current = begin;
85         half = length / 2;
86         std::advance(current, half);
87         if (cmp(extractor(*current), key))
88         {
89             begin = ++current;
90             length -= half + 1;
91         }
92         else
93         {
94             length = half;
95         }
96     }
97     return begin;
98 }
99
100 /**
101  * Ищет в отсортированном диапазоне первый элемент, не меньший заданного
102  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
103  * @tparam Key тип элемента, с которым будет производиться сравнение
104  * @tparam Comparator бинарный предикат
105  * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
106   дет производиться поиск
107  * @param[in] key элемент, по которому производится поиск
108  * @param[in] cmp компаратор: возвращает 'true', если его первый аргумент должен стоять
109  * в отсортированном диапазоне строго левее второго, 'false' иначе
110  * @return итератор на первый элемент диапазона, значение которого не меньше 'key';
111  * 'end', если такого не нашлось
112  */
113 template<typename Iterator, typename Key, typename Comparator>
114 auto lower_bound(Iterator begin, Iterator end, const Key& key, Comparator cmp) ->
115     check_comparator<Iterator, Comparator>

```

```

110 {
111     return my::lower_bound(begin, end, key, cmp, trivial_extractor<Iterator>);
112 }
113
114 /**
115  * Ищет в отсортированном диапазоне первый элемент, не меньший заданного
116  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
117  * @tparam Key тип элемента, с которым будет производиться сравнение
118  * @tparam KeyExtractor тип, объект которого может быть вызван с аргументом типа,
119  * на который указывает Iterator, и возвращающий Key
120  * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
121     дет производиться поиск
122  * @param[in] key элемент, по которому производится поиск
123  * @param[in] extractor функция, возвращающая по объекту значение, которое будет использ-
124     оваться
125  * при сравнении с 'key': например, 'Iterator' ссылается на диапазон пар чисел,
126  * отсортированных сначала по первому элементу пары, затем по второму, и требуется найти
127  * первую пару, первый
128  * элемент которой не меньше 'key'. В этом случае эта функция должна по паре возвращать
129  * ее первый элемент.
130  * Необходимо, чтобы 'extractor' сохранял свойство отсортированности диапазона
131  * @return итератор на первый элемент диапазона, значение 'extractor'
132  * от которого не меньше 'key'; 'end', если такого не нашлось
133 */
134 template<typename Iterator, typename Key, typename KeyExtractor>
135 auto lower_bound(Iterator begin, Iterator end, const Key& key, KeyExtractor extractor) -
136 >
137     check_key_extractor<Iterator, Key, KeyExtractor>
138 {
139     return my::lower_bound(begin, end, key, std::less<Key>(), extractor);
140 }
141
142 /**
143  * Ищет в отсортированном диапазоне первый элемент, не меньший заданного
144  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
145  * @tparam Key тип элемента, с которым будет производиться сравнение
146  * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
147     дет производиться поиск
148  * @param[in] key элемент, по которому производится поиск
149  * @return итератор на первый элемент диапазона, значение которого не меньше 'key';
150  * 'end', если такого не нашлось
151 */
152 template<typename Iterator, typename Key>
153 Iterator lower_bound(Iterator begin, Iterator end, const Key& key)
154 {
155     return my::lower_bound(begin, end, key, std::less<elem_type<Iterator>>(),
156         trivial_extractor<Iterator>);
157 }
158
159 // upper_bound
160
161 /**
162  * Ищет в отсортированном диапазоне первый элемент, строго больший заданного
163  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
164  * @tparam Key тип элемента, с которым будет производиться сравнение
165  * @tparam Comparator бинарный предикат
166  * @tparam KeyExtractor тип, объект которого может быть вызван с аргументом типа,
167  * на который указывает Iterator, и возвращающий Key
168  * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
169     дет производиться поиск
170  * @param[in] key элемент, по которому производится поиск
171  * @param[in] cmp компаратор: возвращает 'true', если его первый аргумент должен стоять
172  * в отсортированном диапазоне строго левее второго, 'false' иначе
173  * @param[in] extractor функция, возвращающая по объекту значение, которое будет использ-
174     оваться
175  * при сравнении с 'key': например, 'Iterator' ссылается на диапазон пар чисел,
176  * отсортированных сначала по первому элементу пары, затем по второму, и требуется найти
177  * первую пару, первый
178  * элемент которой строго больше 'key'. В этом случае эта функция должна по паре возвращ-
179  * ать ее первый элемент.

```

```

169 * Необходимо, чтобы 'extractor' сохранял свойство отсортированности диапазона
170 * @return итератор на первый элемент диапазона, значение 'extractor'
171 * от которого строго больше 'key'; 'end', если такого не нашлось
172 */
173 template<typename Iterator, typename Key, typename Comparator, typename KeyExtractor>
174 auto upper_bound(Iterator begin, Iterator end, const Key& key, Comparator cmp,
175                 KeyExtractor extractor) ->
176     check_comparator_and_key_extractor<Iterator, Key, Comparator, KeyExtractor>
177 {
178     diff_t<Iterator> length = std::distance(begin, end), half;
179     while (length > 0)
180     {
181         Iterator current;
182         current = begin;
183         half = length / 2;
184         std::advance(current, half);
185         if (!cmp(key, extractor(*current)))
186         {
187             begin = ++current;
188             length -= half + 1;
189         }
190         else
191         {
192             length = half;
193         }
194     }
195     return begin;
196 }
197 /**
198 * Ищет в отсортированном диапазоне первый элемент, строго больший заданного
199 * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
200 * @tparam Key тип элемента, с которым будет производиться сравнение
201 * @tparam Comparator бинарный предикат
202 * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
203     дет производиться поиск
204 * @param[in] key элемент, по которому производится поиск
205 * @param[in] стр компаратор: возвращает 'true', если его первый аргумент должен стоять
206     в отсортированном диапазоне строго левее второго, 'false' иначе
207 * @return итератор на первый элемент диапазона, значение которого строго больше 'key';
208     'end', если такого не нашлось
209 */
210 template<typename Iterator, typename Key, typename Comparator>
211 auto upper_bound(Iterator begin, Iterator end, const Key& key, Comparator cmp) ->
212     check_comparator<Iterator, Comparator>
213 {
214     return my::upper_bound(begin, end, key, cmp, trivial_extractor<Iterator>);
215 }
216 /**
217 * Ищет в отсортированном диапазоне первый элемент, строго больший заданного
218 * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
219 * @tparam Key тип элемента, с которым будет производиться сравнение
220 * @tparam KeyExtractor тип, объект которого может быть вызван с аргументом типа,
221     на который указывает Iterator, и возвращающий Key
222 * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
223     дет производиться поиск
224 * @param[in] key элемент, по которому производится поиск
225 * @param[in] extractor функция, возвращающая по объекту значение, которое будет использ-
226     оваться
227 * при сравнении с 'key': например, 'Iterator' ссылается на диапазон пар чисел,
228 * отсортированных сначала по первому элементу пары, затем по второму, и требуется найти
229 * первую пару, первый
230 * элемент которой строго больше 'key'. В этом случае эта функция должна по паре возвращ-
231     ать ее первый элемент.
232 * Необходимо, чтобы 'extractor' сохранял свойство отсортированности диапазона
233 * @return итератор на первый элемент диапазона, значение 'extractor'
234 * от которого строго больше 'key'; 'end', если такого не нашлось
235 */
236 template<typename Iterator, typename Key, typename KeyExtractor>

```



```

233 auto upper_bound(Iterator begin, Iterator end, const Key& key, KeyExtractor extractor) -
234 >
235     check_key_extractor<Iterator, Key, KeyExtractor>
236 {
237     return my::upper_bound(begin, end, key, std::less<Key>(), extractor);
238 }
239
240 /**
241  * Ищет в отсортированном диапазоне первый элемент, строго больший заданного
242  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
243  * @tparam Key тип элемента, с которым будет производиться сравнение
244  * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
245   * дет производиться поиск
246  * @param[in] key элемент, по которому производится поиск
247  * @return итератор на первый элемент диапазона, значение которого строго больше 'key';
248  * 'end', если такого не нашлось
249 */
250 template<typename Iterator, typename Key>
251 Iterator upper_bound(Iterator begin, Iterator end, const Key& key)
252 {
253     return my::upper_bound(begin, end, key, std::less<elem_type<Iterator>>(),
254                             trivial_extractor<Iterator>);
255 }
256
257 // equal_range
258
259 /**
260  * Ищет в отсортированном диапазоне отрезок с элементами, эквивалентными заданному
261  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
262  * @tparam Key тип элемента, с которым будет производиться сравнение
263  * @tparam Comparator бинарный предикат
264  * @tparam KeyExtractor тип, объект которого может быть вызван с аргументом типа,
265   * на который указывает Iterator, и возвращающий Key
266  * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
267   * дет производиться поиск
268  * @param[in] key элемент, по которому производится поиск
269  * @param[in] cmp компаратор: возвращает 'true', если его первый аргумент должен стоять
270   * в отсортированном диапазоне строго левее второго, 'false' иначе
271  * @param[in] extractor функция, возвращающая по объекту значение, которое будет использ-
272   * оваться
273  * при сравнении с 'key': например, 'Iterator' ссылается на диапазон пар чисел,
274  * отсортированных сначала по первому элементу пары, затем по второму, и требуется найти
275   * диапазон пар, первый
276  * элемент которых равен 'key'. В этом случае эта функция должна по паре возвращать ее п-
277   * ервый элемент.
278  * Необходимо, чтобы 'extractor' сохранял свойство отсортированности диапазона
279  * @return пара итераторов, первый элемент которой указывает на первый элемент найденног-
280   * о отрезка,
281  * второй – на элемент, следующий за последним в отрезке эквивалентных; если нужного от-
282   * резка не
283  * нашлось, оба элемента пары будут равны 'end'
284 */
285 template<typename Iterator, typename Key, typename Comparator, typename KeyExtractor>
286 std::pair<Iterator, Iterator> equal_range(Iterator begin, Iterator end, const Key& key,
287                                           Comparator cmp, KeyExtractor extractor)
288 {
289     return {my::lower_bound(begin, end, key, cmp, extractor),
290            my::upper_bound(begin, end, key, cmp, extractor)};
291 }
292
293 /**
294  * Ищет в отсортированном диапазоне отрезок с элементами, эквивалентными заданному
295  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
296  * @tparam Key тип элемента, с которым будет производиться сравнение
297  * @tparam Comparator бинарный предикат
298  * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
299   * дет производиться поиск
300  * @param[in] key элемент, по которому производится поиск
301  * @param[in] cmp компаратор: возвращает 'true', если его первый аргумент должен стоять
302   * в отсортированном диапазоне строго левее второго, 'false' иначе

```

```

292 * @return пара итераторов, первый элемент которой указывает на первый элемент найденног
    о отрезка,
293 * Второй – на элемент, следующий за последним в отрезке эквивалентных; если нужного отр
    езка не
294 * нашлось, оба элемента пары будут равны 'end'
295 */
296 template<typename Iterator, typename Key, typename Comparator>
297 std::pair<Iterator, Iterator> equal_range(Iterator begin, Iterator end, const Key& key,
    Comparator cmp)
298 {
299     return {my::lower_bound(begin, end, key, cmp),
300            my::upper_bound(begin, end, key, cmp)};
301 }
302
303 /**
304 * Ищет в отсортированном диапазоне отрезок с элементами, эквивалентными заданному
305 * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
306 * @tparam Key тип элемента, с которым будет производиться сравнение
307 * @tparam KeyExtractor тип, объект которого может быть вызван с аргументом типа,
308 * на который указывает Iterator, и возвращающий Key
309 * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу
    дет производиться поиск
310 * @param[in] key элемент, по которому производится поиск
311 * @param[in] extractor функция, возвращающая по объекту значение, которое будет использ
    оваться
312 * при сравнении с 'key': например, 'Iterator' ссылается на диапазон пар чисел,
313 * отсортированных сначала по первому элементу пары, затем по второму, и требуется найти
    диапазон пар, первый
314 * элемент которых равен 'key'. В этом случае эта функция должна по паре возвращать ее п
    ервый элемент.
315 * Необходимо, чтобы 'extractor' сохранял свойство отсортированности диапазона
316 * @return пара итераторов, первый элемент которой указывает на первый элемент найденног
    о отрезка,
317 * Второй – на элемент, следующий за последним в отрезке эквивалентных; если нужного отр
    езка не
318 * нашлось, оба элемента пары будут равны 'end'
319 */
320 template<typename Iterator, typename Key, typename KeyExtractor>
321 std::pair<Iterator, Iterator> upper_bound(Iterator begin, Iterator end, const Key& key,
    KeyExtractor extractor)
322 {
323     return {my::lower_bound(begin, end, key, extractor),
324            my::upper_bound(begin, end, key, extractor)};
325 }
326
327 /**
328 * Ищет в отсортированном диапазоне отрезок с элементами, эквивалентными заданному
329 * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
330 * @tparam Key тип элемента, с которым будет производиться сравнение
331 * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу
    дет производиться поиск
332 * @param[in] key элемент, по которому производится поиск
333 * @return пара итераторов, первый элемент которой указывает на первый элемент найденног
    о отрезка,
334 * Второй – на элемент, следующий за последним в отрезке эквивалентных; если нужного отр
    езка не
335 * нашлось, оба элемента пары будут равны 'end'
336 */
337 template<typename Iterator, typename Key, typename Comparator>
338 std::pair<Iterator, Iterator> equal_range(Iterator begin, Iterator end, const Key& key)
339 {
340     return {lower_bound(begin, end, key),
341            upper_bound(begin, end, key)};
342 }
343
344 } // namespace my
345
346 #endif // BINARY_SEARCH_H

```

### Листинг 3: lab2/linear\_search.h

```

1  /**
2   * @file
3   * @brief Заголовочный файл, содержащий реализацию линейного поиска
4   * @date Январь 2020
5   */
6  #ifndef LINEAR_SEARCH_H
7  #define LINEAR_SEARCH_H
8
9  #include <iterator>
10 #include <functional>
11 #include <type_traits>
12 #include <vector>
13
14 namespace my
15 {
16
17  /**
18   * Ищет в диапазоне все элементы, эквивалентные заданному
19   * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
20   * @tparam Key тип элемента, с которым будет производиться сравнение
21   * @tparam BinaryPredicate бинарный предикат
22   * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
23   *   дет производиться поиск
24   * @param[in] key элемент, по которому производится поиск
25   * @param[in] are_equal объект, производящий сравнение на равенство
26   * @return 'std::vector', содержащий итераторы на все элементы диапазона, эквивалентные
27   *   'key'
28   */
29  template<typename Iterator, typename Key, typename BinaryPredicate>
30  std::vector<Iterator> find(Iterator begin, Iterator end, const Key& key, BinaryPredicate
31    are_equal)
32  {
33    std::vector<Iterator> answer;
34    for (; begin != end; std::advance(begin, 1))
35      if (are_equal(*begin, key))
36        answer.push_back(begin);
37    return answer;
38  }
39
40  /**
41   * Ищет в диапазоне все элементы, эквивалентные заданному
42   * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
43   * @tparam Key тип элемента, с которым будет производиться сравнение
44   * @tparam BinaryPredicate бинарный предикат
45   * @param[in] begin, end итераторы, указывающие на отсортированный диапазон, в котором бу-
46   *   дет производиться поиск
47   * @param[in] key элемент, по которому производится поиск
48   * @param[in] are_equal объект, производящий сравнение на равенство
49   * @return 'std::vector', содержащий итераторы на все элементы диапазона, эквивалентные
50   *   'key'
51   */
52  template<typename Iterator, typename Key>
53  std::vector<Iterator> find(Iterator begin, Iterator end, const Key& key)
54  {
55    return find(begin, end, key, std::equal_to<typename std::iterator_traits<Iterator>::
56      value_type>());
57  }
58
59 } // namespace my
60
61 #endif // LINEAR_SEARCH_H

```

### Листинг 4: lab2/main.cpp

```

1  #include "entry.h"
2  #include "io_operations.h"
3  #include "binary_search.h"
4  #include "linear_search.h"
5  #include "../lab1/quick_sort.h"

```

```

6 #include <boost/program_options.hpp>
7 #include <algorithm>
8 #include <chrono>
9 #include <fstream>
10 #include <iostream>
11 #include <map>
12 #include <string>
13 #include <vector>
14 #include <cassert>
15 #include <random>
16 #include <type_traits>
17
18 using ArraySize = std::size_t;
19 using AlgoName = std::string;
20 using Time = std::int64_t;
21 using Data = std::vector<Entry>;
22 using Iterator = Data::iterator;
23 using SizeToTime = std::map<ArraySize, Time>;
24 using TestResult = std::map<AlgoName, SizeToTime>;
25
26 static std::mt19937 prng(std::random_device{}());
27
28 enum class Algorithm
29 {
30     LINEAR_SEARCH,
31     MY_BINARY_SEARCH,
32     MY_SORT_AND_BINARY_SEARCH,
33     STD_BINARY_SEARCH,
34     STD_SORT_AND_BINARY_SEARCH,
35     MULTIMAP,
36 };
37
38 static const std::vector<Algorithm> algos = {Algorithm::LINEAR_SEARCH, Algorithm::
39     MY_BINARY_SEARCH, Algorithm::MY_SORT_AND_BINARY_SEARCH,
40     Algorithm::STD_BINARY_SEARCH, Algorithm::
41     STD_SORT_AND_BINARY_SEARCH, Algorithm::MULTIMAP};
42
43 std::vector<Entry::Club> pick_random_elements(Data::const_iterator begin, Data::
44     const_iterator end, std::size_t length)
45 {
46     std::ptrdiff_t size = std::distance(begin, end);
47     std::uniform_int_distribution<std::ptrdiff_t> dist(0, size - 1);
48     std::vector<Entry::Club> answer;
49     for (std::size_t i = 0; i < length; ++i)
50         answer.push_back(std::next(begin, dist(prng))->club());
51     return answer;
52 }
53
54 TestResult test_all(const Data& data, const std::vector<ArraySize>& sizes)
55 {
56     auto key_extractor = [](const Entry& entry)
57     {
58         return entry.club();
59     };
60
61     TestResult answer;
62     const std::size_t SEARCH_COUNT = 50;
63
64     for (ArraySize size : sizes)
65     {
66         size = std::min(size, data.size());
67         Data::const_iterator data_size_it = std::next(data.begin(), static_cast<std::
68             ptrdiff_t>(size));
69         std::vector<Entry::Club> elements_to_search = pick_random_elements(data.begin(),
70             data_size_it, SEARCH_COUNT);
71
72         #ifndef NDEBUG
73             std::map<Algorithm, std::vector<std::size_t>> num_of_elems_found;
74         #endif
75
76         for (Algorithm algo : algos)
77         {

```

```

71         using namespace std::chrono;
72         time_point<high_resolution_clock> start;
73         SizeToTime* current_algo_result;
74         auto add_timing = [&start, size, &current_algo_result]()
75         {
76             time_point<high_resolution_clock> end = high_resolution_clock::now();
77             (*current_algo_result)[size] += duration_cast<std::chrono::nanoseconds>(
end - start).count();
78         };
79
80         switch (algo)
81         {
82         case Algorithm::LINEAR_SEARCH:
83         {
84             current_algo_result = &answer["Linear search"];
85             for (const Entry::Club& element_to_search : elements_to_search)
86             {
87                 start = high_resolution_clock::now();
88                 std::vector<Data::const_iterator> elements = my::find(data.begin(),
data_size_it, element_to_search,
89                                     [])(const Entry
& elem, const Entry::Club& key){ return elem.club() == key; });
90                 add_timing();
91             #ifndef NDEBUG
92                 num_of_elems_found[Algorithm::LINEAR_SEARCH].push_back(elements.size
());
93             #endif
94             }
95             break;
96         }
97         case Algorithm::MY_BINARY_SEARCH:
98         {
99             current_algo_result = &answer["My binary search"];
100             Data data_copy(data.begin(), data_size_it);
101             my::quick_sort(data_copy.begin(), data_copy.end());
102             for (const Entry::Club& element_to_search : elements_to_search)
103             {
104                 start = high_resolution_clock::now();
105                 auto [range_begin, range_end] = my::equal_range(data_copy.begin(),
data_copy.end(), element_to_search, key_extractor);
106                 add_timing();
107             #ifndef NDEBUG
108                 num_of_elems_found[Algorithm::MY_BINARY_SEARCH].push_back(
static_cast<std::size_t>(range_end - range_begin));
109             #endif
110             }
111             break;
112         }
113         case Algorithm::MY_SORT_AND_BINARY_SEARCH:
114         {
115             current_algo_result = &answer["My sort & binary search"];
116             for (const Entry::Club& element_to_search : elements_to_search)
117             {
118                 Data data_copy(data.begin(), data_size_it);
119                 start = high_resolution_clock::now();
120                 my::quick_sort(data_copy.begin(), data_copy.end());
121                 auto [range_begin, range_end] = my::equal_range(data_copy.begin(),
data_copy.end(), element_to_search, key_extractor);
122                 add_timing();
123             #ifndef NDEBUG
124                 num_of_elems_found[Algorithm::MY_SORT_AND_BINARY_SEARCH].push_back(
static_cast<std::size_t>(range_end - range_begin));
125             #endif
126             }
127             break;
128         }
129         case Algorithm::STD_BINARY_SEARCH:
130         {
131             current_algo_result = &answer["STD binary search"];
132             Data data_copy(data.begin(), data_size_it);

```

```

133         std::sort(data_copy.begin(), data_copy.end());
134         for (const Entry::Club& element_to_search : elements_to_search)
135         {
136             start = high_resolution_clock::now();
137             auto [range_begin, range_end] = std::equal_range(data_copy.begin(),
138 data_copy.end(), element_to_search, std::less<Entry::Club>());
139             add_timing();
140         }
141         break;
142     }
143     case Algorithm::STD_SORT_AND_BINARY_SEARCH:
144     {
145         current_algo_result = &answer["STD sort & binary search"];
146         for (const Entry::Club& element_to_search : elements_to_search)
147         {
148             Data data_copy(data.begin(), data_size_it);
149             start = high_resolution_clock::now();
150             std::sort(data_copy.begin(), data_copy.end());
151             auto [range_begin, range_end] = std::equal_range(data_copy.begin(),
152 data_copy.end(), element_to_search, std::less<Entry::Club>());
153             add_timing();
154         }
155         break;
156     }
157     case Algorithm::MULTIMAP:
158     {
159         current_algo_result = &answer["Multimap"];
160         std::multimap<Entry::Club, Entry> mmap;
161         for (Data::const_iterator it = data.begin(); it != data_size_it; ++it)
162             mmap.emplace(it->club(), *it);
163         for (const Entry::Club& element_to_search : elements_to_search)
164         {
165             start = high_resolution_clock::now();
166             auto [range_begin, range_end] = mmap.equal_range(element_to_search);
167             add_timing();
168         }
169         num_of_elems_found[Algorithm::MULTIMAP].push_back(mmap.count(
170 element_to_search));
171     }
172     }
173     (*current_algo_result)[size] /= elements_to_search.size();
174 }
175 #ifndef NDEBUG
176     std::vector<std::size_t> ethalon = num_of_elems_found[Algorithm::MULTIMAP];
177     assert(ethalon == num_of_elems_found[Algorithm::LINEAR_SEARCH]);
178     assert(ethalon == num_of_elems_found[Algorithm::MY_BINARY_SEARCH]);
179     assert(ethalon == num_of_elems_found[Algorithm::MY_SORT_AND_BINARY_SEARCH]);
180 #endif
181 }
182
183     return answer;
184 }
185
186 int main(int argc, char* argv[])
187 {
188     std::ios::sync_with_stdio(false);
189     std::cin.tie(nullptr);
190
191     namespace po = boost::program_options;
192     po::options_description desc("Allowed options");
193     desc.add_options()
194         ("help,H", "Print this message")
195         ("sizes,S", po::value<std::string>()->required(), "Text file with data sizes to
196 be testes in the following format:\n"
197                                     "size_0 size_1 size_2 ...
198                                     size_n")

```

```

198         ("input,l", po::value<std::string>()->required(), "File (csv or sqlite) with
199         football clubs data. Format:\n"
200         trainer;year;score\n"
201         "format,F", po::value<std::string>(), "Input file format (csv or sqlite)")
202         ("output,O", po::value<std::string>()->required(), "csv file to write test
203         results, the format is:\n"
204         ;...;result_for_size_n")
205         ;
206 po::variables_map vm;
207 try
208 {
209     po::store(parse_command_line(argc, argv, desc), vm);
210     if (vm.contains("help"))
211     {
212         std::cout << desc << "\n";
213         return 0;
214     }
215     po::notify(vm);
216 }
217 catch (const po::error& error)
218 {
219     std::cerr << "Error while parsing command-line arguments: "
220     << error.what() << "\nPlease use --help to see help message\n";
221     return 1;
222 }
223
224 std::string input_filename = vm["input"].as<std::string>();
225 std::string sizes_filename = vm["sizes"].as<std::string>();
226 std::string output_filename = vm["output"].as<std::string>();
227 std::string format;
228 if (vm.contains("format"))
229 {
230     format = vm["format"].as<std::string>();
231 }
232 else
233 {
234     if (input_filename.ends_with(".csv"))
235     {
236         format = "csv";
237     }
238     else if (input_filename.ends_with(".sqlite"))
239     {
240         format = "sqlite";
241     }
242     else
243     {
244         std::cerr << "Invalid format. Please either specify format manually with "
245         "--format or use --input with extension .csv or .sqlite.\n"
246         "Please use --help to see detailed help message";
247     }
248 }
249
250 if (format != "csv" && format != "sqlite")
251 {
252     std::cerr << "Invalid format. Please use --help see help message\n";
253     return 1;
254 }
255
256 std::cerr << "Reading data..." << std::endl;
257 Data data;
258 if (format == "csv")
259     data = read_data_from_csv(input_filename);
260 else if (format == "sqlite")
261     data = read_data_from_sqlite(input_filename);
262 std::vector<ArraySize> sizes = read_sizes(sizes_filename);

```

```

263     shrink_sizes(sizes, data.size());
264     std::cerr << "Done!" << std::endl;
265
266     // csv header
267     std::ofstream output(output_filename);
268     output << "name";
269     for (ArraySize size : sizes)
270         output << ';' << size;
271     output << '\n';
272
273     TestResult results = test_all(data, sizes);
274     for (auto& [name, timings] : results)
275     {
276         std::cerr << std::endl << "Algorithm: " << name << std::endl;
277         for (auto [size, time] : timings)
278             std::cerr << size << ": " << time << std::endl;
279         print_timings_csv_line(output, name, timings);
280     }
281
282     return 0;
283 }

```

#### Листинг 5: lab2/run.py

```

1 import subprocess
2 import os
3 import numpy as np
4 import pandas as pd
5 from matplotlib import pyplot as plt
6
7 try:
8     print("Generating data...")
9     if not os.path.isfile("data.sqlite"):
10         subprocess.call("../generate_data/generate_data --size=100000 --output=data.
11             sqlite", shell=True)
12         print("Done!")
13
14     if not os.path.isfile("sizes.txt"):
15         f = open("sizes.txt", "w")
16         f.write(" ".join(np.logspace(2, 5, 20).astype(int).astype(str)) + "\n")
17         f.close()
18
19     print("Running tests...")
20     if not os.path.isfile("results.csv"):
21         subprocess.call("./2_test_search --input=data.sqlite --sizes=sizes.txt --output=
22             results.csv 2> test_search.log", shell=True)
23         print("Done!")
24
25     raw = pd.read_csv("results.csv", sep=';')
26     data = dict()
27     columns = list(raw.columns[1:].astype(int))
28     for i in range(raw.shape[0]):
29         data[raw.iloc[i]["name"]] = list(raw.iloc[i][1:])
30     plt.figure(figsize=(12, 8))
31     for name, timings in data.items():
32         plt.plot(columns, timings, label=name)
33         plt.xscale('log')
34         plt.yscale('log')
35         plt.legend()
36     plt.show()
37 except Exception as e:
38     print(e)

```