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**:

```
../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.

```
./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

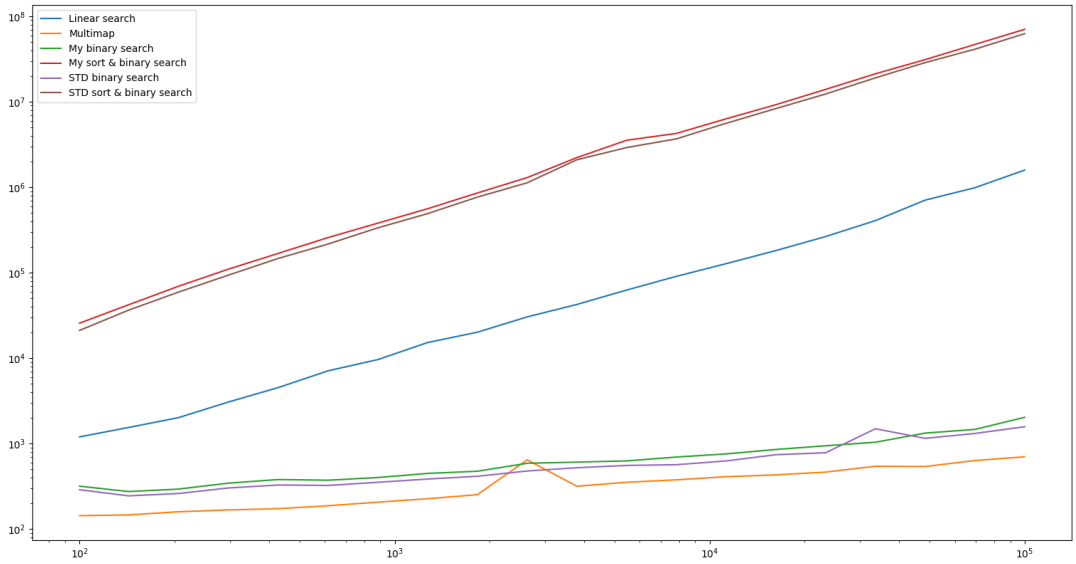
1274	1832	2636	3792	5455	7847	11288
15211	20091	30351	42446	62571	90530	127413
226	252	644	316	352	376	409
447	474	587	606	626	695	756
560224	855431	1295677	2220675	3542074	4261243	6315128
384	414	477	521	554	565	626
491055	765611	1126244	2094916	2910047	3679666	5601570

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 практическипорядок 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

```
set(PROJECT_NAME 2_test_search)

project(${PROJECT_NAME} LANGUAGES CXX)

set(SOURCES main.cpp)
set(HEADERS binary_search.h
            linear_search.h)

add_executable(${PROJECT_NAME} ${SOURCES} ${HEADERS})

target_include_directories(${PROJECT_NAME} PRIVATE ${Helpers_INCLUDE_DIR} ${
    Entry_INCLUDE_DIR})
target_link_libraries(${PROJECT_NAME} PRIVATE entry)
target_link_libraries(${PROJECT_NAME} PRIVATE helpers)
target_link_libraries(${PROJECT_NAME} PRIVATE Boost::program_options)

file(COPY run.py DESTINATION ${PROJECT_BINARY_DIR})
```

Листинг 2: lab2/binary_search.h

```
/**
 * @file
 * @brief Заголовочный файл, содержащий реализацию бинарного поиска
 * @date Январь 2020
*/
#ifndef BINARY_SEARCH_H
#define BINARY_SEARCH_H

#include <iterator>
#include <functional>
#include <type_traits>

namespace my
{

namespace
{

// helper aliases
template<typename Iterator>
using elem_type = typename std::iterator_traits<Iterator>::value_type;

template<typename Iterator>
using diff_t = typename std::iterator_traits<Iterator>::difference_type;

// Iterator if functions can be invoked in a right way, nothing otherwise
template<typename Iterator, typename Key, typename KeyExtractor>
using check_key_extractor = typename std::enable_if<
    std::is_invocable_r<const Key&, KeyExtractor, const elem_type<Iterator>&>::value,
    Iterator>::type;

template<typename Iterator, typename Comparator>
using check_comparator = typename std::enable_if<
    std::is_invocable_r<bool, Comparator, const elem_type<Iterator>&, const elem_type<
    Iterator>&>::value,
    Iterator>::type;

template<typename Iterator, typename Key, typename Comparator, typename KeyExtractor>
using check_comparator_and_key_extractor = typename std::enable_if<
        std::is_invocable_r<const Key&, KeyExtractor, const elem_type<Iterator>&>::value
     &&
        std::is_invocable_r<bool, Comparator, const Key&, const Key&>::value,
    Iterator>::type;

template<typename Iterator>
const elem_type<Iterator>& trivial_extractor(const elem_type<Iterator>& elem)
{
```

```
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 указывает Iterator, и возвращающий Key
60   * @param[in] begin,end итераторы, указывающие на отсортированный диапазон, в котором бу
       дет производиться поиск
61   * @param[in] key элемент, по которому производится поиск
62   * @param[in] cmp компаратор: возвращает `true`, если его первый аргумент должен стоять
63   * в отсортированном диапазоне строго левее второго, `false` иначе
64   * @param[in] extractor функция, возвращающая по объекту значение, которое будет использ
       оваться
65   * при сравнении с `key`: например, `Iterator` ссылается на диапазон пар чисел,
66   * отсортированных сначала по первому элементу пары, затем по второму, и требуется найти
       первую пару, первый
67   * элемент которой не меньше `key`. В этом случае эта функция должна по паре возвращать
       ее первый элемент.
68   * Необходимо, чтобы `extractor` сохранял свойство отсортированности диапазона
69   * @return итератор на первый элемент диапазона, значение `extractor`
70   * от которого не меньше `key`; `end`, если такого не нашлось
71   */
72  template<typename Iterator, typename Key, typename Comparator, typename KeyExtractor>
73  auto lower_bound(Iterator begin, Iterator end, const Key& key, Comparator cmp,
        KeyExtractor extractor) ->
74      check_comparator_and_key_extractor<Iterator, Key, Comparator, KeyExtractor>
75  {
76      diff_t<Iterator> length = std::distance(begin, end), half;
77      while (length > 0)
78      {
79          Iterator current = begin;
80          half = length / 2;
81          std::advance(current, half);
82          if (cmp(extractor(*current), key))
83          {
84              begin = ++current;
85              length -= half + 1;
86          }
87          else
88          {
89              length = half;
90          }
91      }
92      return begin;
93  }
94
95  /**
96   * Ищет в отсортированном диапазоне первый элемент, не меньший заданного
97   * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
98   * @tparam Key тип элемента, с которым будет производиться сравнение
99   * @tparam Comparator бинарный предикат
100  * @param[in] begin,end итераторы, указывающие на отсортированный диапазон, в котором бу
       дет производиться поиск
101  * @param[in] key элемент, по которому производится поиск
102  * @param[in] cmp компаратор: возвращает `true`, если его первый аргумент должен стоять
103  * в отсортированном диапазоне строго левее второго, `false` иначе
104  * @return итератор на первый элемент диапазона, значение которого не меньше `key`;
105  * `end`, если такого не нашлось
106  */
107 template<typename Iterator, typename Key, typename Comparator>
108 auto lower_bound(Iterator begin, Iterator end, const Key& key, Comparator cmp) ->
109     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
        дет производиться поиск
121  * @param[in] key элемент, по которому производится поиск
122  * @param[in] extractor функция, возвращающая по объекту значение, которое будет использ
        оваться
123  * при сравнении с `key`: например, `Iterator` ссылается на диапазон пар чисел,
124  * отсортированных сначала по первому элементу пары, затем по второму, и требуется найти
         первую пару, первый
125  * элемент которой не меньше `key`. В этом случае эта функция должна по паре возвращать
        ее первый элемент.
126  * Необходимо, чтобы `extractor` сохранял свойство отсортированности диапазона
127  * @return итератор на первый элемент диапазона, значение `extractor`
128  * от которого не меньше `key`; `end`, если такого не нашлось
129 */
130 template<typename Iterator, typename Key, typename KeyExtractor>
131 auto lower_bound(Iterator begin, Iterator end, const Key& key, KeyExtractor extractor) -
        >
132     check_key_extractor<Iterator, Key, KeyExtractor>
133 {
134     return my::lower_bound(begin, end, key, std::less<Key>(), extractor);
135 }
136
137 /**
138  * Ищет в отсортированном диапазоне первый элемент, не меньший заданного
139  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
140  * @tparam Key тип элемента, с которым будет производиться сравнение
141  * @param[in] begin,end итераторы, указывающие на отсортированный диапазон, в котором бу
        дет производиться поиск
142  * @param[in] key элемент, по которому производится поиск
143  * @return итератор на первый элемент диапазона, значение которого не меньше `key`;
144  * `end`, если такого не нашлось
145 */
146 template<typename Iterator, typename Key>
147 Iterator lower_bound(Iterator begin, Iterator end, const Key& key)
148 {
149     return my::lower_bound(begin, end, key, std::less<elem_type<Iterator>>(),
        trivial_extractor<Iterator>);
150 }
151
152 // upper_bound
153
154 /**
155  * Ищет в отсортированном диапазоне первый элемент, строго больший заданного
156  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
157  * @tparam Key тип элемента, с которым будет производиться сравнение
158  * @tparam Comparator бинарный предикат
159  * @tparam KeyExtractor тип, объект которого может быть вызван с аргументом типа,
160  * на который указывает Iterator, и возвращающий Key
161  * @param[in] begin,end итераторы, указывающие на отсортированный диапазон, в котором бу
        дет производиться поиск
162  * @param[in] key элемент, по которому производится поиск
163  * @param[in] cmp компаратор: возвращает `true`, если его первый аргумент должен стоять
164  * в отсортированном диапазоне строго левее второго, `false` иначе
165  * @param[in] extractor функция, возвращающая по объекту значение, которое будет использ
        оваться
166  * при сравнении с `key`: например, `Iterator` ссылается на диапазон пар чисел,
167  * отсортированных сначала по первому элементу пары, затем по второму, и требуется найти
         первую пару, первый
168  * элемент которой строго больше `key`. В этом случае эта функция должна по паре возвращ
        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,
         KeyExtractor extractor) ->
175     check_comparator_and_key_extractor<Iterator, Key, Comparator, KeyExtractor>
176 {
177     diff_t<Iterator> length = std::distance(begin, end), half;
178     while (length > 0)
179     {
180         Iterator current;
181         current = begin;
182         half = length / 2;
183         std::advance(current, half);
184         if (!cmp(key, extractor(*current)))
185         {
186             begin = ++current;
187             length -= half + 1;
188         }
189         else
190         {
191             length = half;
192         }
193     }
194     return begin;
195 }
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
         дет производиться поиск
203  * @param[in] key элемент, по которому производится поиск
204  * @param[in] cmp компаратор: возвращает `true`, если его первый аргумент должен стоять
205  * в отсортированном диапазоне строго левее второго, `false` иначе
206  * @return итератор на первый элемент диапазона, значение которого строго больше `key`;
207  * `end`, если такого не нашлось
208  */
209 template<typename Iterator, typename Key, typename Comparator>
210 auto upper_bound(Iterator begin, Iterator end, const Key& key, Comparator cmp) ->
211     check_comparator<Iterator, Comparator>
212 {
213     return my::upper_bound(begin, end, key, cmp, trivial_extractor<Iterator>);
214 }
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
         дет производиться поиск
223  * @param[in] key элемент, по которому производится поиск
224  * @param[in] extractor функция, возвращающая по объекту значение, которое будет использ
         оваться
225  * при сравнении с `key`: например, `Iterator` ссылается на диапазон пар чисел,
226  * отсортированных сначала по первому элементу пары, затем по второму, и требуется найти
          первую пару, первый
227  * элемент которой строго больше `key`. В этом случае эта функция должна по паре возвращ
         ать ее первый элемент.
228  * Необходимо, чтобы `extractor` сохранял свойство отсортированности диапазона
229  * @return итератор на первый элемент диапазона, значение `extractor`
230  * от которого строго больше `key`; `end`, если такого не нашлось
231  */
232 template<typename Iterator, typename Key, typename KeyExtractor>
```

```
233 auto upper_bound(Iterator begin, Iterator end, const Key& key, KeyExtractor extractor) -
        >
234     check_key_extractor<Iterator, Key, KeyExtractor>
235 {
236     return my::upper_bound(begin, end, key, std::less<Key>(), extractor);
237 }
238
239 /**
240  * Ищет в отсортированном диапазоне первый элемент, строго больший заданного
241  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
242  * @tparam Key тип элемента, с которым будет производиться сравнение
243  * @param[in] begin,end итераторы, указывающие на отсортированный диапазон, в котором бу
        дет производиться поиск
244  * @param[in] key элемент, по которому производится поиск
245  * @return итератор на первый элемент диапазона, значение которого строго больше `key`;
246  * `end`, если такого не нашлось
247 */
248 template<typename Iterator, typename Key>
249 Iterator upper_bound(Iterator begin, Iterator end, const Key& key)
250 {
251     return my::upper_bound(begin, end, key, std::less<elem_type<Iterator>>(),
        trivial_extractor<Iterator>);
252 }
253
254 // equal_range
255
256 /**
257  * Ищет в отсортированном диапазоне отрезок с элементами, эквивалентными заданному
258  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
259  * @tparam Key тип элемента, с которым будет производиться сравнение
260  * @tparam Comparator бинарный предикат
261  * @tparam KeyExtractor тип, объект которого может быть вызван с аргументом типа,
262  * на который указывает Iterator, и возвращающий Key
263  * @param[in] begin,end итераторы, указывающие на отсортированный диапазон, в котором бу
        дет производиться поиск
264  * @param[in] key элемент, по которому производится поиск
265  * @param[in] cmp компаратор: возвращает `true`, если его первый аргумент должен стоять
266  * в отсортированном диапазоне строго левее второго, `false` иначе
267  * @param[in] extractor функция, возвращающая по объекту значение, которое будет использ
        оваться
268  * при сравнении с `key`: например, `Iterator` ссылается на диапазон пар чисел,
269  * отсортированных сначала по первому элементу пары, затем по второму, и требуется найти
         диапазон пар, первый
270  * элемент которых равен `key`. В этом случае эта функция должна по паре возвращать ее п
        ервый элемент.
271  * Необходимо, чтобы `extractor` сохранял свойство отсортированности диапазона
272  * @return пара итераторов, первый элемент которой указывает на первый элемент найденног
        о отрезка,
273  * второй – на элемент, следующий за последним в отрезке эквивалентных; если нужного отр
        езка не
274  * нашлось, оба элемента пары будут равны `end`
275 */
276 template<typename Iterator, typename Key, typename Comparator, typename KeyExtractor>
277 std::pair<Iterator, Iterator> equal_range(Iterator begin, Iterator end, const Key& key,
        Comparator cmp, KeyExtractor extractor)
278 {
279     return {my::lower_bound(begin, end, key, cmp, extractor),
280             my::upper_bound(begin, end, key, cmp, extractor)};
281 }
282
283 /**
284  * Ищет в отсортированном диапазоне отрезок с элементами, эквивалентными заданному
285  * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
286  * @tparam Key тип элемента, с которым будет производиться сравнение
287  * @tparam Comparator бинарный предикат
288  * @param[in] begin,end итераторы, указывающие на отсортированный диапазон, в котором бу
        дет производиться поиск
289  * @param[in] key элемент, по которому производится поиск
290  * @param[in] cmp компаратор: возвращает `true`, если его первый аргумент должен стоять
291  * 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: la21/linear_search.h

```
/**
 * @file
 * @brief Заголовочный файл, содержащий реализацию линейного поиска
 * @date Январь 2020
*/
#ifndef LINEAR_SEARCH_H
#define LINEAR_SEARCH_H

#include <iterator>
#include <functional>
#include <type_traits>
#include <vector>

namespace my
{

/**
 * Ищет в диапазоне все элементы, эквивалентные заданному
 * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
 * @tparam Key тип элемента, с которым будет производиться сравнение
 * @tparam BinaryPredicate бинарный предикат
 * @param[in] begin,end итераторы, указывающие на отсортированный диапазон, в котором буд
    ет производиться поиск
 * @param[in] key элемент, по которому производится поиск
 * @param[in] are_equal объект, производящий сравнение на равенство
 * @return `std::vector`, содержащий итераторы на все элементы диапазона, эквивалентные
    `key`
*/
template<typename Iterator, typename Key, typename BinaryPredicate>
std::vector<Iterator> find(Iterator begin, Iterator end, const Key& key, BinaryPredicate
     are_equal)
{
    std::vector<Iterator> answer;
    for (; begin != end; std::advance(begin, 1))
        if (are_equal(*begin, key))
            answer.push_back(begin);
    return answer;
}

/**
 * Ищет в диапазоне все элементы, эквивалентные заданному
 * @tparam Iterator тип, удовлетворяющий C++ named requirement LegacyForwardIterator
 * @tparam Key тип элемента, с которым будет производиться сравнение
 * @param[in] begin,end итераторы, указывающие на отсортированный диапазон, в котором буд
    ет производиться поиск
 * @param[in] key элемент, по которому производится поиск
 * @return `std::vector`, содержащий итераторы на все элементы диапазона, эквивалентные
    `key`
*/
template<typename Iterator, typename Key>
std::vector<Iterator> find(Iterator begin, Iterator end, const Key& key)
{
    return find(begin, end, key, std::equal_to<typename std::iterator_traits<Iterator>::
    value_type>());
}

} // namespace my

#endif // LINEAR_SEARCH_H
```

Листинг 4: lab2/main.cpp

```
#include "entry.h"
#include "io_operations.h"
#include "binary_search.h"
#include "linear_search.h"
#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::
       MY_BINARY_SEARCH, Algorithm::MY_SORT_AND_BINARY_SEARCH,
39                                                 Algorithm::STD_BINARY_SEARCH, Algorithm::
       STD_SORT_AND_BINARY_SEARCH, Algorithm::MULTIMAP};
40
41 std::vector<Entry::Club> pick_random_elements(Data::const_iterator begin, Data::
       const_iterator end, std::size_t length)
42 {
43     std::ptrdiff_t size = std::distance(begin, end);
44     std::uniform_int_distribution<std::ptrdiff_t> dist(0, size - 1);
45     std::vector<Entry::Club> answer;
46     for (std::size_t i = 0; i < length; ++i)
47         answer.push_back(std::next(begin, dist(prng))->club());
48     return answer;
49 }
50
51 TestResult test_all(const Data& data, const std::vector<ArraySize>& sizes)
52 {
53     auto key_extractor = [](const Entry& entry)
54     {
55         return entry.club();
56     };
57
58     TestResult answer;
59     const std::size_t SEARCH_COUNT = 50;
60
61     for (ArraySize size : sizes)
62     {
63         size = std::min(size, data.size());
64         Data::const_iterator data_size_it = std::next(data.begin(), static_cast<std::
       ptrdiff_t>(size));
65         std::vector<Entry::Club> elements_to_search = pick_random_elements(data.begin(),
        data_size_it, SEARCH_COUNT);
66 #ifndef NDEBUG
67         std::map<Algorithm, std::vector<std::size_t>> num_of_elems_found;
68 #endif
69         for (Algorithm algo : algos)
70         {
```

```
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(),
    data_copy.end(), element_to_search, std::less<Entry::Club>());
138                     add_timing();
139                 }
140                 break;
141             }
142             case Algorithm::STD_SORT_AND_BINARY_SEARCH:
143             {
144                 current_algo_result = &answer["STD sort & binary search"];
145                 for (const Entry::Club& element_to_search : elements_to_search)
146                 {
147                     Data data_copy(data.begin(), data_size_it);
148                     start = high_resolution_clock::now();
149                     std::sort(data_copy.begin(), data_copy.end());
150                     auto [range_begin, range_end] = std::equal_range(data_copy.begin(),
    data_copy.end(), element_to_search, std::less<Entry::Club>());
151                     add_timing();
152                 }
153                 break;
154             }
155             case Algorithm::MULTIMAP:
156             {
157                 current_algo_result = &answer["Multimap"];
158                 std::multimap<Entry::Club, Entry> mmap;
159                 for (Data::const_iterator it = data.begin(); it != data_size_it; ++it)
160                     mmap.emplace(it->club(), *it);
161                 for (const Entry::Club& element_to_search : elements_to_search)
162                 {
163                     start = high_resolution_clock::now();
164                     auto [range_begin, range_end] = mmap.equal_range(element_to_search);
165                     add_timing();
166 #ifndef NDEBUG
167                     num_of_elems_found[Algorithm::MULTIMAP].push_back(mmap.count(
    element_to_search));
168 #endif
169                 }
170                 break;
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
    be testes in the following format:\n"
196                                                           "size_0 size_1 size_2 ...
    size_n")
197
```

```
198         ("input,I", po::value<std::string>()->required(), "File (csv or sqlite) with
    football clubs data. Format:\n"
199                                                           "* if csv: country;club;city;
    trainer;year;score\n"
200                                                           "* if sqlite: table 'entries'
    with columns 'country', 'club', 'city', 'trainer', 'year', 'score'")
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
    results, the format is:\n"
203                                                            "algo_name;result_for_size_0
    ;...;result_for_size_n")
204         ;
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.
       sqlite", shell=True)
11     print("Done!")
12 
13     if not os.path.isfile("sizes.txt"):
14         f = open("sizes.txt", "w")
15         f.write(" ".join(np.logspace(2, 5, 20).astype(int).astype(str)) + "\n")
16         f.close()
17 
18     print("Running tests...")
19     if not os.path.isfile("results.csv"):
20         subprocess.call("./2_test_search --input=data.sqlite --sizes=sizes.txt --output=
       results.csv 2> test_search.log", shell=True)
21     print("Done!")
22 
23     raw = pd.read_csv("results.csv", sep=';')
24     data = dict()
25     columns = list(raw.columns[1:].astype(int))
26     for i in range(raw.shape[0]):
27         data[raw.iloc[i]["name"]] = list(raw.iloc[i][1:])
28     plt.figure(figsize=(12, 8))
29     for name, timings in data.items():
30         plt.plot(columns, timings, label=name)
31         plt.xscale('log')
32         plt.yscale('log')
33         plt.legend()
34     plt.show()
35 except Exception as e:
36     print(e)
```