Lab 3 Hashing

1.0

Generated by Doxygen 1.9.5

Chapter 1

Bug List

File generator.cpp

Currently, there are no any known bugs.

File generator.hpp

Currently, there are no any known bugs.

File model.cpp

Currently, there are no any known bugs.

File model.hpp

Currently, there are no any known bugs.

File search.cpp

Currently, there are no any known bugs.

File sorting.cpp

Currently, there are no any known bugs.

File sorting.hpp

Currently, there are no any known bugs.

Currently, there are no any known bugs.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Generator Class Reference

Public Member Functions

- Model **model_generator** ()

The documentation for this class was generated from the following files:

- src/generator/generator.hpp
- src/generator/generator.cpp

4.2 Hashing Class Reference

Static Public Member Functions

- static std::uint32_t **basic_hashing_function** (const std::string &value)
- static std::uint32_t **djb2_hashing_function** (const std::string &value)
- static std::uint32_t **advanced_hashing_function** (const std::string &value)
- static std::uint32_t **count_collisions** (const std::vector< std::list< Model > > &hash_table)
- static std::vector< std::list< Model > > **hash_model** (std::vector< Model > &model_vector, std::function< std::size_t(const std::string &value)> hash_function)
- static std::optional< Model > **find_in_hash_table** (const std::vector< std::list< Model > > &hash_table, std::uint32_t hash, std::size_t size)

The documentation for this class was generated from the following files:

- src/model/model.hpp
- src/model/model.cpp

4.3 Model Class Reference

Public Member Functions

- **Model** (std::string full_name, std::string department, std::string job_title, std::chrono::year_month_day employment_date, std::uint32_t model_hash=0, std::uint8_t hash_field=255, const std::optional< std::function< std::size_t(const std::string &value)> > &optional_func=std::nullopt)
- **Model** (std::string full_name, std::string department, std::string job_title, std::string employment_date, std::uint32_t model_hash=0, std::uint8_t hash_field=255, const std::optional< std::function< std::size_t(const std::string &value)> > &optional_func=std::nullopt)
- **Model** (std::uint8_t decor_type)
- **Model** (const Model &other)
- void **set_model** (std::string full_name, std::string department, std::string job_title, std::chrono::year_month_day employment_date, std::uint32_t model_hash=0, std::uint8_t hash_field=255, const std::optional< std::function< std::size_t(const std::string &value)> > &optional_func=std::nullopt)
- void **set_model** (std::string full_name, std::string department, std::string job_title, std::string employment_date, std::uint32_t model_hash=0, std::uint8_t hash_field=255, const std::optional< std::function< std::size_t(const std::string &value)> > &optional_func=std::nullopt)
- void **set_decor** (std::uint8_t decor_type)
- void **set_hash_func** (std::function< std::size_t(const std::string &value)> hash_fucntion)
- void **set_hash** (std::uint32_t model_hash)
- void **set_hash_field** (std::uint8_t hash_field)
- ModelComp **compare_type** (const Model &r_model, std::uint8_t mode)
- template<typename T >
  ModelComp **compare_type** (std::uint8_t mode, T r_value)
- template<typename T >
  T **get_field** (std::uint8_t field) const
- template<typename T >
  void **set_field** (std::uint8_t field, T value)
- std::uint32_t **get_hash** () const
- std::uint8_t **get_hash_field** () const
- template<typename T >
  T **get_hash_field** () const

Static Public Member Functions

- static void **save_model** (const std::vector< Model > &model_vector, std::filesystem::path file_path)
- static void **load_model** (std::vector< Model > &model_vector, std::filesystem::path file_path)
- static void **print_model** (const std::vector< Model > &model_vector)

Friends

- std::ostream & **operator<<** (std::ostream &stream, const Model &model)
- ModelComp **operator<** (const Model &l_model, const Model &r_model)
- ModelComp **operator>** (const Model &l_model, const Model &r_model)
- ModelComp **operator<=** (const Model &l_model, const Model &r_model)
- ModelComp **operator>=** (const Model &l_model, const Model &r_model)
- ModelComp **operator==** (const Model &l_model, const Model &r_model)
- ModelComp **operator!=** (const Model &l_model, const Model &r_model)

The documentation for this class was generated from the following files:

- src/model/model.hpp
- src/model/model.cpp

4.4 ModelComp Class Reference

Public Member Functions

- **ModelComp** (std::uint8_t value)
- void **set_type_masked** (std::uint8_t value, std::uint8_t offset)
- void **set_name_type** (std::uint8_t value)
- void **set_dept_type** (std::uint8_t value)
- void **set_jobt_type** (std::uint8_t value)
- void **set_date_type** (std::uint8_t value)
- std::uint8_t **get_type_masked** (std::uint8_t offset) const
- std::uint8_t **get_name_type** () const
- std::uint8_t **get_dept_type** () const
- std::uint8_t **get_jobt_type** () const
- std::uint8_t **get_date_type** () const
- ModelComp **operator!** () const

Friends

- ModelComp **operator==** (const ModelComp &l_bool, const ModelComp &r_bool)
- ModelComp **operator!=** (const ModelComp &l_bool, const ModelComp &r_bool)
- std::ostream & **operator**<< (std::ostream &stream, const ModelComp &model)

The documentation for this class was generated from the following files:

- src/model/model.hpp
- src/model/model.cpp

4.5 Search Class Reference

A class that provides static searching methods for sorting a vector of Model objects based on a specific field.

`#include <search.hpp>`

Static Public Member Functions

- template<typename T >
  static int **binary_search** (std::vector< Model > &model_vector, T search_value, std::uint8_t field)
- template<typename T >
  static int **straight_search** (std::vector< Model > &model_vector, T search_value, std::uint8_t field)

4.5.1 Detailed Description

A class that provides static searching methods for sorting a vector of Model objects based on a specific field.

Note

Currently provides implementations for binary search.

The documentation for this class was generated from the following file:

- src/search/search.hpp

4.6 Sorting Class Reference

A class that provides static sorting methods for sorting a vector of Model objects based on a specific field.

`#include <sorting.hpp>`

Static Public Member Functions

- static void bubble_sort (std::vector< Model > &model_vector, uint8_t field)

  Sorts the given vector of Model objects using bubble sort algorithm.
- static void heap_sort (std::vector< Model > &model_vector, uint8_t field)

  Performs heap sort on a vector of Model objects.
- static void merge_sort (std::vector< Model > &model_vector, uint8_t field, std::size_t left=0, std::size_t right=0, bool initial=true)

  Sorts a vector of Model objects using merge sort algorithm.

4.6.1 Detailed Description

A class that provides static sorting methods for sorting a vector of Model objects based on a specific field.

Note

Currently provides implementations for bubble sort, heap sort, and merge sort.

4.6.2 Member Function Documentation

4.6.2.1 bubble_sort()

```
void Sorting::bubble_sort (
            std::vector< Model > & model_vector,
            uint8_t field )  [static]
```

Sorts the given vector of Model objects using bubble sort algorithm.

This function uses bubble sort algorithm to sort the given vector of Model objects based on the field specified by the `field` parameter. The objects are compared using the `compare_type()` method of the Model class.

Parameters

model_vector	The vector of Model objects to be sorted.
field	The index of the field to be used for sorting the objects.

Returns

void.

Note

This function modifies the original vector passed to it.

The `compare_type()` method of the Model class must return a Type object.

The Type object must have a method `get_type_masked()` that takes an offset and returns a boolean indicating whether the specified bit is set or not.

This function prints the number of iterations taken to sort the vector.

4.6.2.2 heap_sort()

```
void Sorting::heap_sort (
            std::vector< Model > & model_vector,
            uint8_t field )  [static]
```

Performs heap sort on a vector of Model objects.

This function sorts a vector of Model objects using the heap sort algorithm. The function uses the make_heap function to create a heap from the input vector, then sorts the heap by repeatedly extracting the maximum element from the heap and placing it at the end of the vector.

Parameters

model_vector	The vector of Model objects to be sorted.
field	The field of the Model object to sort by.

Returns

void.

4.6.2.3 merge_sort()

```
void Sorting::merge_sort (
            std::vector< Model > & model_vector,
            uint8_t field,
            std::size_t left = 0,
            std::size_t right = 0,
            bool initial = true )  [static]
```

Sorts a vector of Model objects using merge sort algorithm.

This function sorts a given vector of Model objects using merge sort algorithm. It takes the field to be sorted as input, along with left and right indices of the sub-vector to be sorted. If left and right indices are not provided, it sorts the entire vector by setting left and right indices accordingly.

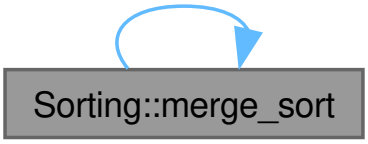
Parameters

model_vector	The vector of Model objects to be sorted.
field	The field to be sorted.
left	The left index of the sub-vector to be sorted (default is 0).
right	The right index of the sub-vector to be sorted (default is size-1).
initial	A boolean flag indicating whether this is the initial call to the function (default is true).

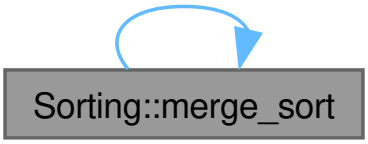
Returns

void.

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- src/sorting/sorting.hpp
- src/sorting/sorting.cpp

Chapter 5

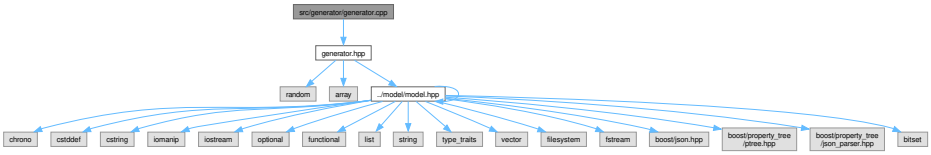
File Documentation

5.1 src/generator/generator.cpp File Reference

This source file holds implementation of Generator class.

```
#include "generator.hpp"
```

Include dependency graph for generator.cpp:



5.1.1 Detailed Description

This source file holds implementation of Generator class.

>

```
This calss implements model generator needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright

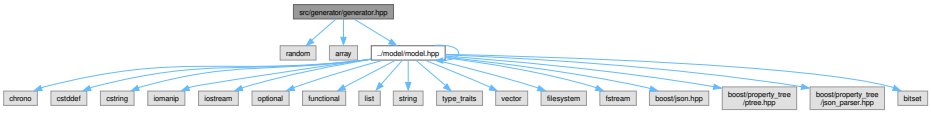

```
(Not really)
```

5.2 src/generator/generator.hpp File Reference

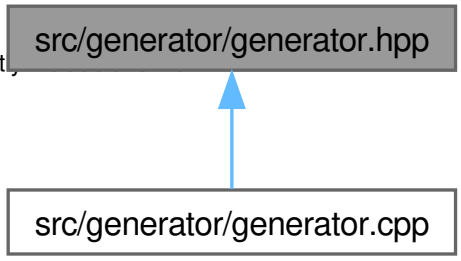
This header file holds implementation of Generator class.

```
#include <random>
#include <array>
#include "../model/model.hpp"
```

Include dependency graph for generator.hpp:



This graph shows which files directly or indirectly



Classes

- class Generator

5.2.1 Detailed Description

This header file holds implementation of Generator class.

>

```
This calss implements model generator needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

5.3 generator.hpp

Go to the documentation of this file.

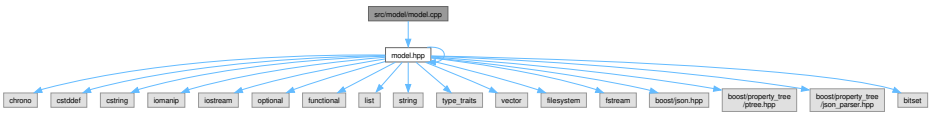
```
1
20 #ifndef GENERATOR_HPP
21 #define GENERATOR_HPP
22
23 #include <random>
24
25 #include <array>
26
27
28 #ifndef MODEL_HPP
29 #include "../model/model.hpp"
30 #endif // MODEL_HPP
31
32 class Generator
33 {
34 public:
35     Generator();
36     ~Generator();
```

```
37
38     Model model_generator();
39
40 private:
41     std::array<std::string, 2738> _first_name_list;
42     std::array<std::string, 1000> _last_name_list;
43     std::array<std::string, 449> _department_list;
44     std::array<std::string, 357> _job_title_list;
45
46     std::random_device       _random_device;
47     std::mt19937             _generator;
48
49     std::uniform_int_distribution<uint32_t> _first_name_distribution;
50     std::uniform_int_distribution<uint32_t> _last_name_distribution;
51     std::uniform_int_distribution<uint32_t> _department_distribution;
52     std::uniform_int_distribution<uint32_t> _job_title_distribution;
53     std::uniform_int_distribution<uint16_t> _year_distribution;
54     std::uniform_int_distribution<uint16_t> _month_distribution;
55     std::uniform_int_distribution<uint16_t> _day_distribution;
56     std::uniform_int_distribution<uint16_t> _sex_distribution;
57 };
58
59 #endif // GENERATOR_HPP
```

5.4 src/model/model.cpp File Reference

This source file holds implementation of Model class.

`#include "model.hpp"`
Include dependency graph for model.cpp:



Functions

- std::ostream & **operator**<< (std::ostream &stream, const Model &model)
- ModelComp **operator**< (const Model &l_model, const Model &r_model)
- ModelComp **operator**>= (const Model &l_model, const Model &r_model)
- ModelComp **operator**<= (const Model &l_model, const Model &r_model)
- ModelComp **operator**> (const Model &l_model, const Model &r_model)
- ModelComp **operator==** (const Model &l_model, const Model &r_model)
- ModelComp **operator!=** (const Model &l_model, const Model &r_model)
- ModelComp **operator==** (const ModelComp &l_bool, const ModelComp &r_bool)
- ModelComp **operator!=** (const ModelComp &l_bool, const ModelComp &r_bool)
- std::ostream & **operator**<< (std::ostream &stream, const ModelComp &r_bool)

5.4.1 Detailed Description

This source file holds implementation of Model class.

>

```
This calss implements model needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

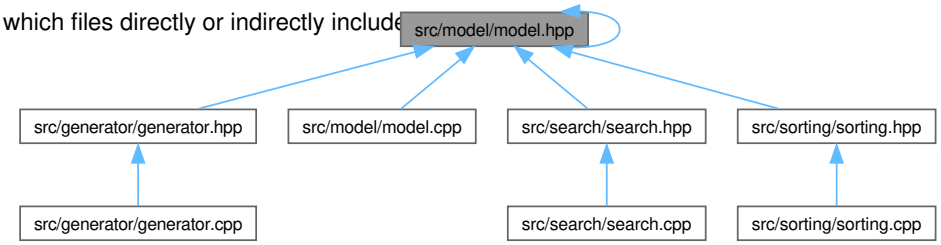
5.5 src/model/model.hpp File Reference

This header file holds implementation of Model class.

```
#include <chrono>
#include <cstddef>
#include <cstring>
#include <iomanip>
#include <iostream>
#include <optional>
#include <functional>
#include <list>
#include <string>
#include <type_traits>
#include <vector>
#include <filesystem>
#include <fstream>
#include <boost/json.hpp>
#include <boost/property_tree/ptree.hpp>
#include <boost/property_tree/json_parser.hpp>
#include <bitset>
#include "../model/model.hpp"
```

Include dependency graph for model.hpp:

This graph shows which files directly or indirectly include



Classes

- class Hashing
- class Model
- class ModelComp

5.5.1 Detailed Description

This header file holds implementation of Model class.

>

```
This calss implements model needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

5.6 model.hpp

Go to the documentation of this file.

```
1
20 #ifndef MODEL_HPP
21 #define MODEL_HPP
22
23
24 #include <chrono>
25 #include <cstddef>
26 #include <cstring> // strcmp has better performance
27 #include <iomanip>
28 #include <iostream>
29 #include <optional>
30 #include <functional>
31 #include <list>
32 #include <optional>
33 #include <string>
34 #include <type_traits>
35 #include <vector>
36
37
38 #include <filesystem>
39 #include <fstream>
40 #include <boost/json.hpp>
41 #include <boost/property_tree/ptree.hpp>
42 #include <boost/property_tree/json_parser.hpp>
43
44 #include <bitset>
45
46 #include "../model/model.hpp"
47
48 class Model;
49
50 class Hashing
51 {
52 public:
53     static std::uint32_t basic_hashing_function(const std::string& value);
54     static std::uint32_t djb2_hashing_function(const std::string& value);
55     static std::uint32_t advanced_hashing_function(const std::string& value);
56
57     static std::uint32_t count_collisions(const std::vector<std::list<Model>& hash_table);
58
59     static std::vector<std::list<Model> hash_model(std::vector<Model>& model_vector,
        std::function<std::size_t(const std::string& value)> hash_function);
60
61     static std::optional<Model> find_in_hash_table(const std::vector<std::list<Model>& hash_table,
        std::uint32_t hash, std::size_t size);
62 };
63
64 class ModelComp;
65
66 class Model
67 {
68 public:
69     Model(std::string full_name, std::string department, std::string job_title,
        std::chrono::year_month_day employment_date, std::uint32_t model_hash = 0, std::uint8_t hash_field =
        255, const std::optional<std::function<std::size_t(const std::string& value)>& optional_func =
        std::nullopt);
70     Model(std::string full_name, std::string department, std::string job_title, std::string
        employment_date, std::uint32_t model_hash = 0, std::uint8_t hash_field = 255, const
        std::optional<std::function<std::size_t(const std::string& value)>& optional_func = std::nullopt);
71     Model(std::uint8_t decor_type);
72     Model(const Model& other) {
73     try {
74         this->_full_name = other._full_name;
75         this->_department = other._department;
76         this->_job_title = other._job_title;
77         this->_employment_date = other._employment_date;
78         this->_decor_type = other._decor_type;
79         this->_hash_field = other._hash_field;
80         this->_model_hash = other._model_hash;
81     } catch (const std::exception& e) {
82         std::cout << e.what() << std::endl;
83     }
84 }
85
86     ~Model();
87
88     void set_model(std::string full_name, std::string department, std::string job_title,
        std::chrono::year_month_day employment_date, std::uint32_t model_hash = 0, std::uint8_t hash_field =
        255, const std::optional<std::function<std::size_t(const std::string& value)>& optional_func =
        std::nullopt);
89     void set_model(std::string full_name, std::string department, std::string job_title, std::string
        employment_date, std::uint32_t model_hash = 0, std::uint8_t hash_field = 255, const
        std::optional<std::function<std::size_t(const std::string& value)>& optional_func = std::nullopt);
```

```
90
91      void set_decor(std::uint8_t decor_type);
92      void set_hash_func(std::function<std::size_t(const std::string& value)> hash_fucntion);
93      void set_hash(std::uint32_t model_hash);
94      void set_hash_field(std::uint8_t hash_field);
95
96      ModelComp compare_type(const Model& r_model, std::uint8_t mode);
97
98      template<typename T>
99      ModelComp compare_type(std::uint8_t mode, T r_value);
100
101      template<typename T>
102      T get_field(std::uint8_t field) const;
103
104      template<typename T>
105      void set_field(std::uint8_t field, T value);
106
107      std::uint32_t get_hash() const;
108      std::uint8_t get_hash_field() const;
109
110      template<typename T>
111      T get_hash_field() const;
112
113      static void save_model(const std::vector<Model>& model_vector, std::filesystem::path file_path);
114      static void load_model(std::vector<Model>& model_vector, std::filesystem::path file_path);
115      static void print_model(const std::vector<Model>& model_vector);
116
117      friend std::ostream& operator« (std::ostream& stream, const Model& model);
118
119      friend ModelComp     operator<  (const Model& l_model, const Model& r_model);
120      friend ModelComp     operator>  (const Model& l_model, const Model& r_model);
121      friend ModelComp     operator<= (const Model& l_model, const Model& r_model);
122      friend ModelComp     operator>= (const Model& l_model, const Model& r_model);
123
124      friend ModelComp     operator== (const Model& l_model, const Model& r_model);
125      friend ModelComp     operator!= (const Model& l_model, const Model& r_model);
126
127  private:
128      std::string                     _full_name;
129      std::string                     _department;
130      std::string                     _job_title;
131      std::chrono::year_month_day     _employment_date;
132
133      std::uint8_t                    _decor_type;
134
135      //HASHING
136      std::uint8_t                    _hash_field;
137
138      std::uint32_t                   _model_hash;
139  };
140
141  class ModelComp
142  {
143  public:
144      ModelComp();
145      ModelComp(std::uint8_t value);
146      ~ModelComp();
147
148      void set_type_masked(std::uint8_t value, std::uint8_t offset);
149
150      void set_name_type(std::uint8_t value);
151      void set_dept_type(std::uint8_t value);
152      void set_jobt_type(std::uint8_t value);
153      void set_date_type(std::uint8_t value);
154
155      std::uint8_t get_type_masked(std::uint8_t offset) const;
156
157      std::uint8_t get_name_type() const;
158      std::uint8_t get_dept_type() const;
159      std::uint8_t get_jobt_type() const;
160      std::uint8_t get_date_type() const;
161
162      friend ModelComp operator== (const ModelComp& l_bool, const ModelComp& r_bool);
163      friend ModelComp operator!= (const ModelComp& l_bool, const ModelComp& r_bool);
164
165      friend std::ostream& operator« (std::ostream& stream, const ModelComp& model);
166
167      ModelComp operator! () const;
168
169  private:
170      std::uint8_t _value;
171  };
172
173
174  template<typename T>
175  ModelComp Model::compare_type(std::uint8_t mode, T r_value)
176  {
```

```
177     int8_t comp_result = 4;
178     ModelComp model_comp;
179 
180     if (mode > 3)
181     {
182         mode = 0;
183     }
184 
185     switch (mode)
186     {
187     case 0:
188         {
189             if (std::is_same<T, decltype(this->_full_name)>::value)
190             {
191                 comp_result = this->_full_name.compare(r_value);
192                 if (comp_result < 0)
193                 {
194                     comp_result = 1;
195                 }
196                 else if(comp_result > 0)
197                 {
198                     comp_result = 2;
199                 }
200                 else
201                 {
202                     comp_result = 0;
203                 }
204             }
205             else
206             {
207                 throw std::invalid_argument("r_value should be _full_name");
208             }
209             model_comp.set_name_type(comp_result);
210             break;
211         }
212     case 1:
213         {
214             if (std::is_same<T, decltype(this->_department)>::value)
215             {
216                 comp_result = this->_department.compare(r_value);
217                 if (comp_result < 0)
218                 {
219                     comp_result = 1;
220                 }
221                 else if(comp_result > 0)
222                 {
223                     comp_result = 2;
224                 }
225                 else
226                 {
227                     comp_result = 0;
228                 }
229             }
230             else
231             {
232                 throw std::invalid_argument("r_value should be _department");
233             }
234             model_comp.set_dept_type(comp_result);
235             break;
236         }
237     case 2:
238         {
239             if (std::is_same<T, decltype(this->_job_title)>::value)
240             {
241                 comp_result = this->_job_title.compare(r_value);
242                 if (comp_result < 0)
243                 {
244                     comp_result = 1;
245                 }
246                 else if(comp_result > 0)
247                 {
248                     comp_result = 2;
249                 }
250                 else
251                 {
252                     comp_result = 0;
253                 }
254             }
255             else
256             {
257                 throw std::invalid_argument("r_value should be _job_title");
258             }
259             model_comp.set_jobt_type(comp_result);
260             break;
261         }
262     case 3:
263         {
```

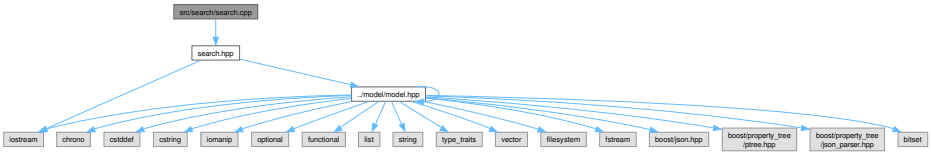
```
264
265             throw std::invalid_argument("r_value should be _employment_date");
266             model_comp.set_date_type(comp_result);
267             break;
268         }
269     default:
270         {
271             break;
272         }
273     }
274
275     return model_comp;
276 }
277
278 template<typename T>
279 T Model::get_field(std::uint8_t field) const
280 {
281     switch (field)
282     {
283
284         case 1:
285         {   if (std::is_same<T, decltype(this->_department)>::value)
286             {
287                 return this->_department;
288             }
289             else
290             {
291                 throw std::invalid_argument("T should be _department");
292             }
293             break;
294         }
295
296         case 2:
297         {   if (std::is_same<T, decltype(this->_job_title)>::value)
298             {
299                 return this->_job_title;
300             }
301             else
302             {
303                 throw std::invalid_argument("T should be _job_title");
304             }
305             break;
306         }
307
308         case 3:
309         {
310             throw std::invalid_argument("T should be _employment_date");
311             break;
312         }
313
314         default:
315         {   if (std::is_same<T, decltype(this->_full_name)>::value)
316             {
317                 return this->_full_name;
318             }
319             else
320             {
321                 throw std::invalid_argument("T should be _full_name");
322             }
323             break;
324         }
325
326     }
327 }
328
329
330 template<typename T>
331 void Model::set_field(std::uint8_t field, T value)
332 {
333     switch (field)
334     {
335
336         case 1:
337         {   if (std::is_same<T, decltype(this->_department)>::value)
338             {
339                 this->_department = value;
340             }
341             else
342             {
343                 throw std::invalid_argument("T should be _department");
344             }
345             break;
346         }
347
348         case 2:
349         {   if (std::is_same<T, decltype(this->_job_title)>::value)
350             {
```

```
351                 this->_job_title = value;
352             }
353             else
354             {
355                 throw std::invalid_argument("T should be _job_title");
356             }
357             break;
358         }
359 
360         case 3:
361         {
362             throw std::invalid_argument("T should be _employment_date");
363             break;
364         }
365 
366         default:
367         {   if (std::is_same<T, decltype(this->_full_name)>::value)
368             {
369                 this->_full_name = value;
370             }
371             else
372             {
373                 throw std::invalid_argument("T should be _full_name");
374             }
375             break;
376         }
377 
378     }
379 }
380 
381 template<typename T>
382 T Model::get_hash_field() const
383 {
384     return this->get_field<T>(this->_hash_field);
385 }
386 
387 #endif // MODEL_HPP
```

5.7 src/search/search.cpp File Reference

This source file holds implementation of Search class.

`#include "search.hpp"`
Include dependency graph for search.cpp:



5.7.1 Detailed Description

This source file holds implementation of Search class.

>

```
This calss implements search algorithms needed for successfull completion of laboratory work 2.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
        (Not really)
```

5.8 search.hpp

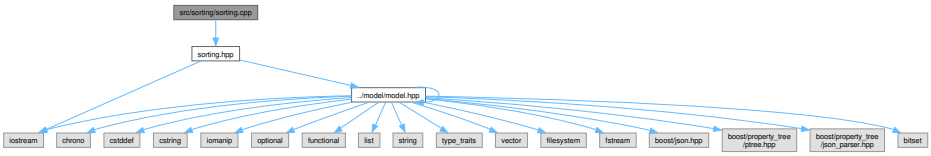
```
1
20 #ifndef SEARCH_HPP
21 #define SEARCH_HPP
22
23 #ifndef MODEL_HPP
24 #include "../model/model.hpp"
25 #endif // MODEL_HPP
26
27 #include <iostream>
28
36 class Search
37 {
38 public:
39     template<typename T>
40     static int binary_search(std::vector<Model>& model_vector, T search_value, std::uint8_t field);
41
42     template<typename T>
43     static int straight_search(std::vector<Model>& model_vector, T search_value, std::uint8_t field);
44 };
45
46
47 template<typename T>
48 int Search::binary_search(std::vector<Model>& model_vector, T search_value, std::uint8_t field)
49 {
50     int dummy = 0;
51     int left = 0;
52     int right = model_vector.size() - 1;
53     uint8_t offset = field * 2;
54
55     while (left <= right)
56     {
57         int mid = (left + right) / 2;
58
59         if (((int)model_vector.at(mid).compare_type<T>(field, search_value).get_type_masked(offset) ==
    0))
60         {
61             return mid;
62         }
63
64         else if (((int)model_vector.at(mid).compare_type<T>(field, search_value).get_type_masked(offset)
    == 1))
65         {
66             left = mid + 1;
67         }
```

```
68
69         else if (((int)model_vector.at(mid).compare_type<T>(field, search_value).get_type_masked(offset)
       == 2))
70         {
71             right = mid - 1;
72         }
73     }
74     return -1;
75 }
76
77 template<typename T>
78 int Search::straight_search(std::vector<Model>& model_vector, T search_value, std::uint8_t field)
79 {
80     uint8_t offset = field * 2;
81
82     for (std::size_t index = 0; index < model_vector.size(); ++index)
83     {
84         if (((int)model_vector.at(index).compare_type<T>(field, search_value).get_type_masked(offset) ==
       0))
85         {
86             return index;
87         }
88     }
89     return -1;
90 }
91
92 #endif // SEARCH_HPP
```

5.9 src/sorting/sorting.cpp File Reference

This source file holds implementation of Sorting class.

`#include "sorting.hpp"`
Include dependency graph for sorting.cpp:



5.9.1 Detailed Description

This source file holds implementation of Sorting class.

>

```
This calss implements sorting algorithms needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

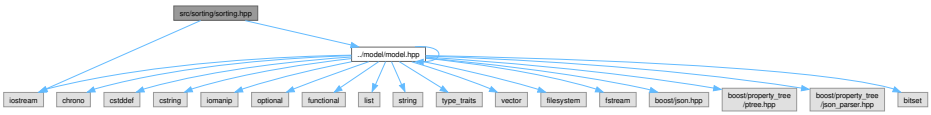
Copyright

5.10 src/sorting/sorting.hpp File Reference

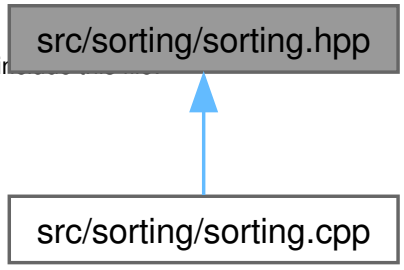
This header file holds implementation of Search class.

```
#include "../model/model.hpp"
#include <iostream>
```

Include dependency graph for sorting.hpp:



This graph shows which files directly or indirectly i



Classes

- class Sorting

  A class that provides static sorting methods for sorting a vector of Model objects based on a specific field.

5.10.1 Detailed Description

This header file holds implementation of Search class.

This header file holds implementation of Sorting class.

>

```
This calss implements search algorithms needed for successfull completion of laboratory work 2.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

>

```
This calss implements sorting algorithms needed for successfull completion of laboratory work 1.
```

Author

Alexander Chudnikov (THE_CHOODICK)

Date

15-02-2023

Version

0.0.1

Warning

This library is under development, so there might be some bugs in it.

Bug Currently, there are no any known bugs.

```
In order to submit new ones, please contact me via admin@redline-software.xyz.
```

Copyright


```
(Not really)
```

5.11 sorting.hpp

Go to the documentation of this file.

```
1
20 #ifndef SORTING_HPP
21 #define SORTING_HPP
22
23 #ifndef MODEL_HPP
24 #include "../model/model.hpp"
25 #endif // MODEL_HPP
26
27 #include <iostream>
28
36 class Sorting
37 {
38 public:
39     static void bubble_sort(std::vector<Model>& model_vector, uint8_t field);
40     static void heap_sort(std::vector<Model>& model_vector, uint8_t field);
41     static void merge_sort(std::vector<Model>& model_vector, uint8_t field, std::size_t left = 0,
       std::size_t right = 0, bool initial = true);
42
43 private:
44     static void make_heap(std::vector<Model>& model_vector, std::size_t index, uint8_t field, std::size_t
       last_index = 0);
45     static void make_merge(std::vector<Model>& model_vector, uint8_t field, std::size_t left, std::size_t
       right, std::size_t middle);
46 };
47
48 #endif // SORTING_HPP
```

Chapter 6

Hashing statistics

6.0.1 Definition for hashing

FULL NAME	DEPARTMENT	JOB TITLE	DATE
MODE 0	**MODE 1**	**MODE 2**	**MODE 3**
Alexzander Oliver Baxterovna	Data Entry	Architectural Technologist	2015/09/09
...	...	...	...
Billy Barrett Okeogheneovich	Audio Engineering	Production Manager	1999/01/15

Table 6.1 Employee information

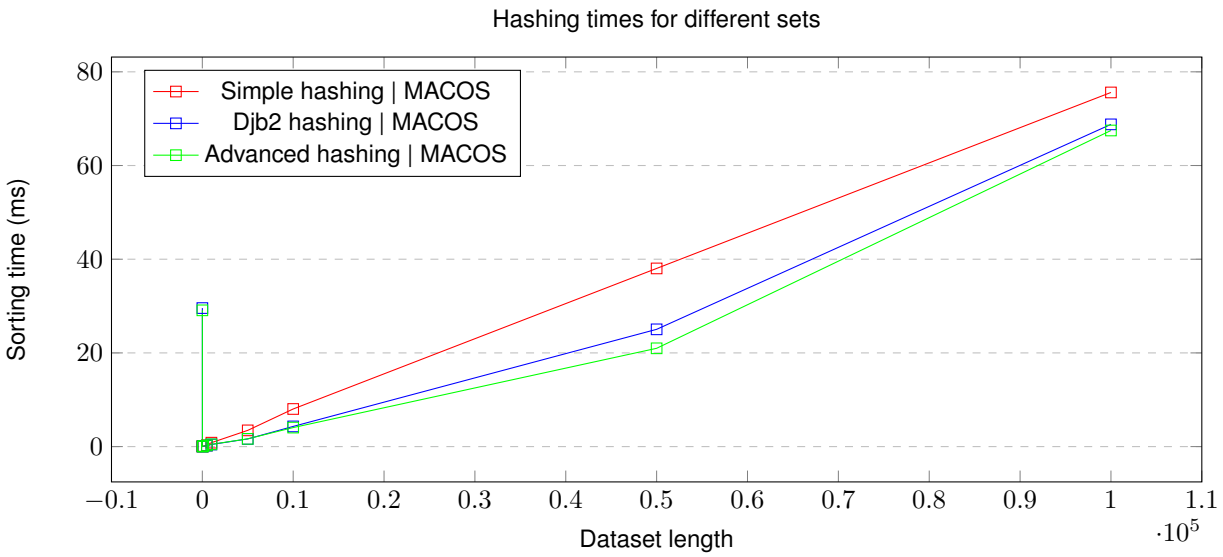
Table 6.2 MACOS specs

Model	MacBook Air (11-inch, Early 2015)
OS	MacOS Montery 12.6
Processor	1.6 GHz 2-core Intel Core i5
Memory	4 GB 1600 MHz DDR3
Graphics	Intel HD Graphics 6000 1536 MB

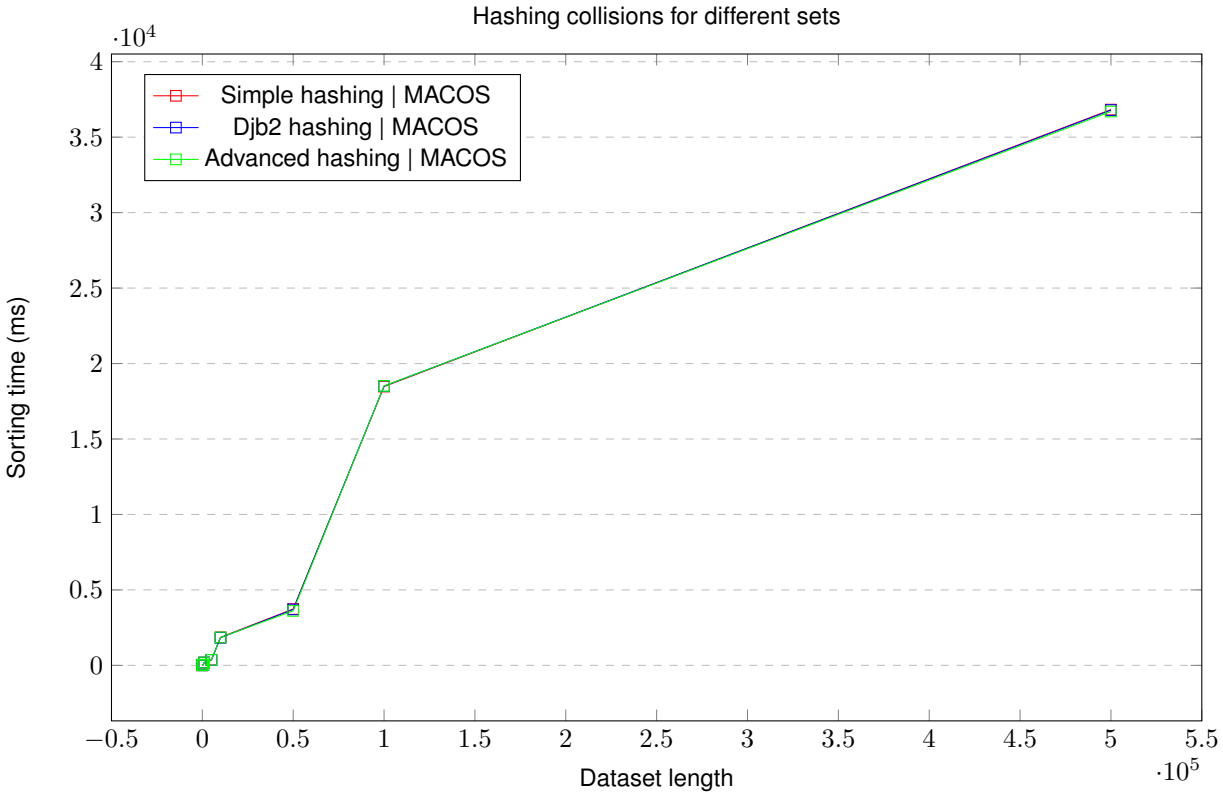
Table 6.3 WIN specs

Model	-
OS	Windows 10 Pro 20H2
Processor	Intel Core i7-8086K @ 4.50 GHz
Memory	80,0 GB 2333 MHz DDR4
Graphics	NVIDIA GeForce GTX 1080

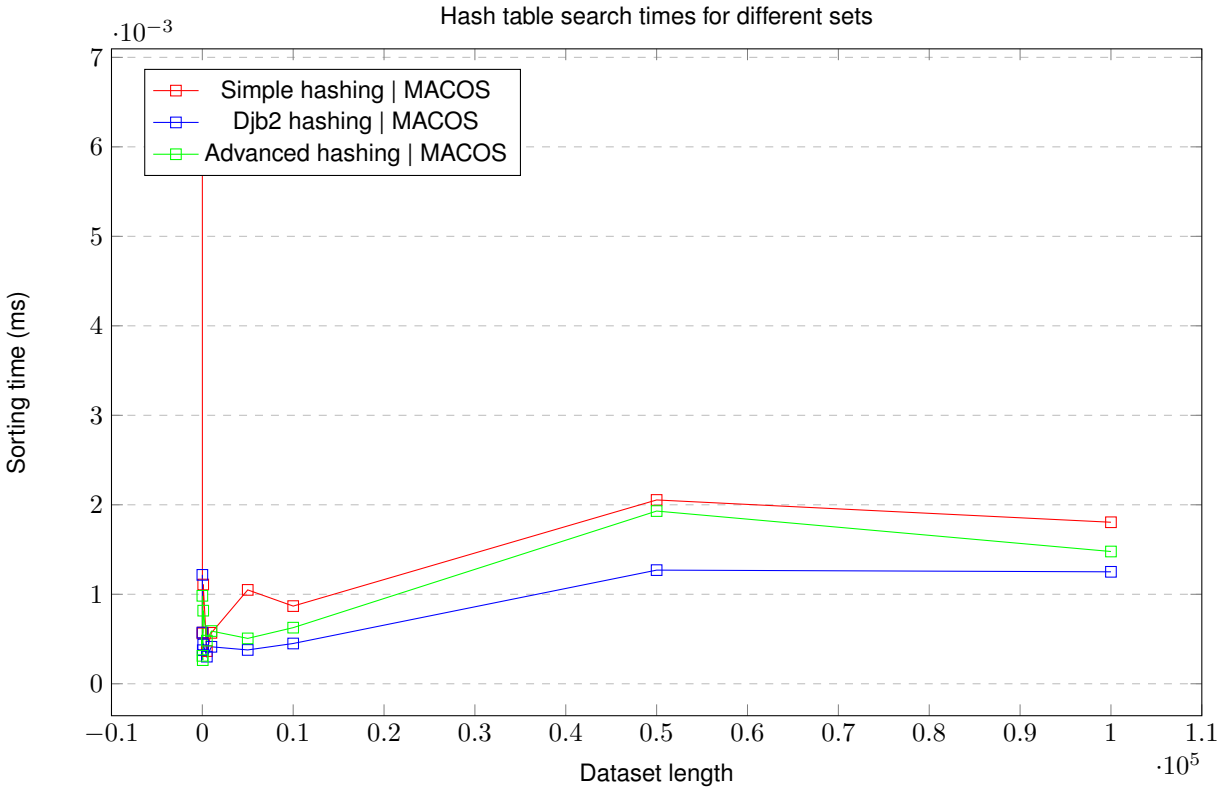
6.0.2 Hashing comparison



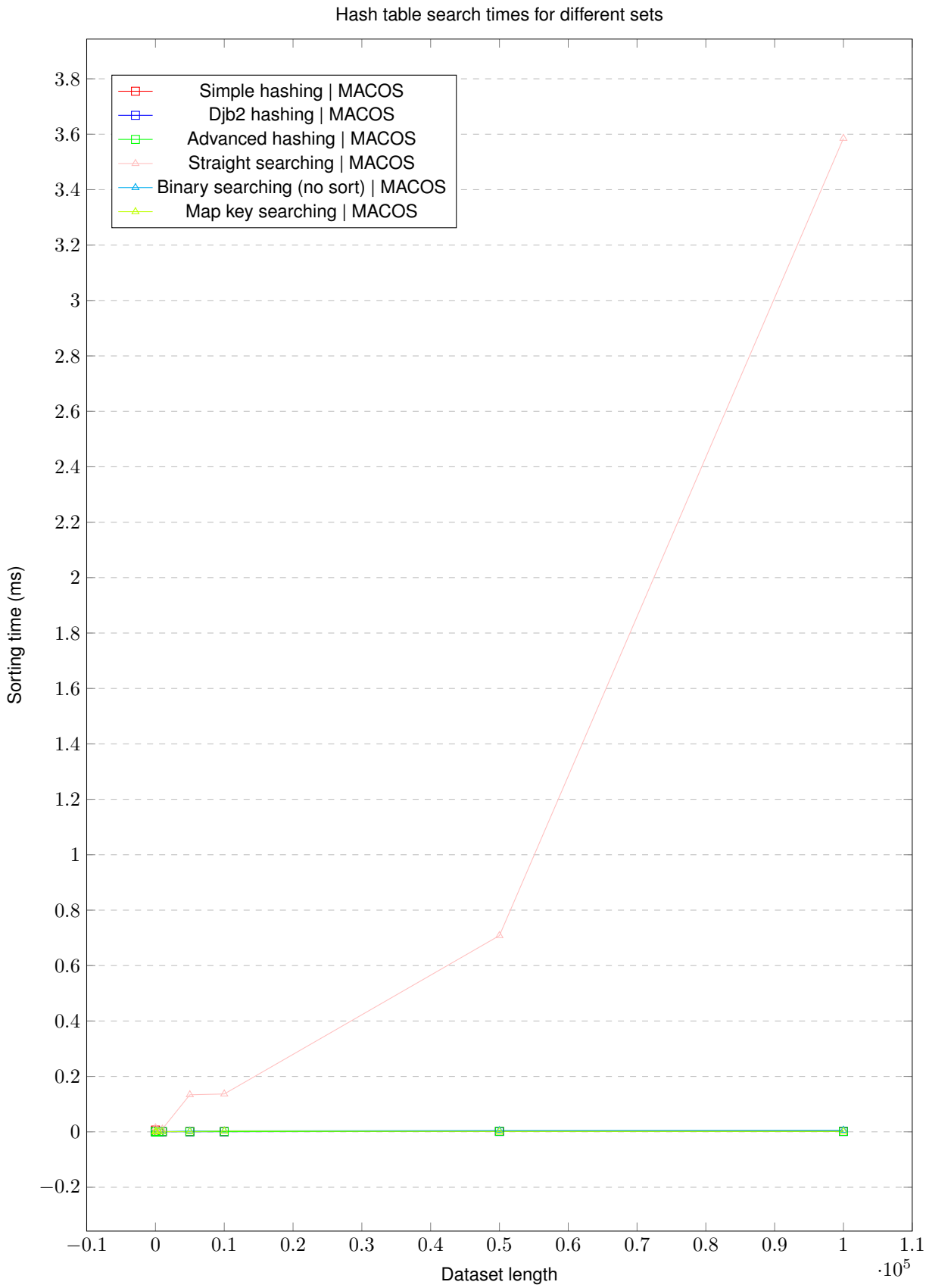
6.0.3 Hashing collision comparison



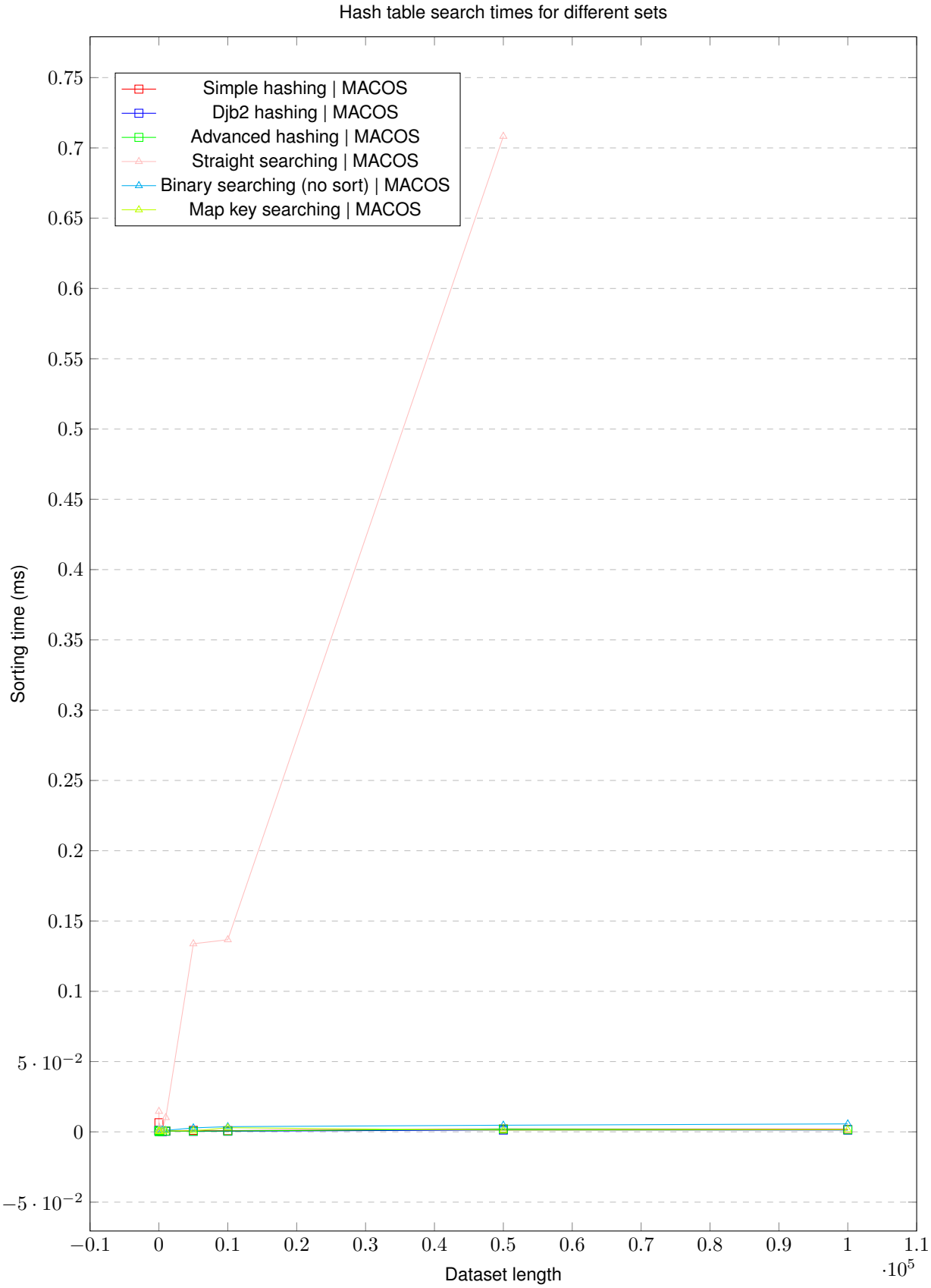
6.0.4 Hash table search comparison



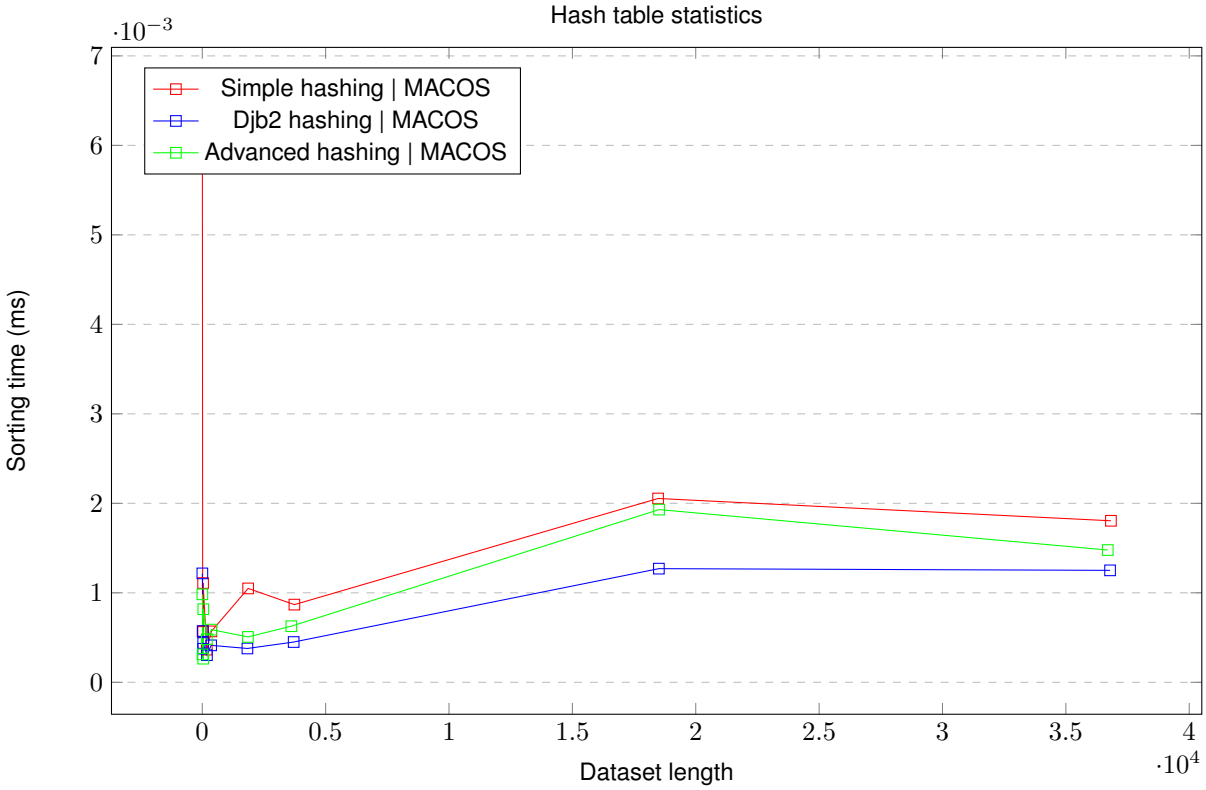
6.0.5 Comparison with previous searches



6.0.6 Comparison with previous searches (without peaks)



6.0.7 Hash table statistics



Index