

哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业核心基础课（必修）

实验项目：内存排序算法及其应用

实验题目：内存排序算法实验比较

实验日期：2023/11/8

班级：2203101

学号：2022113573

姓名：张宇杰

设计成绩	报告成绩	指导老师
		张岩

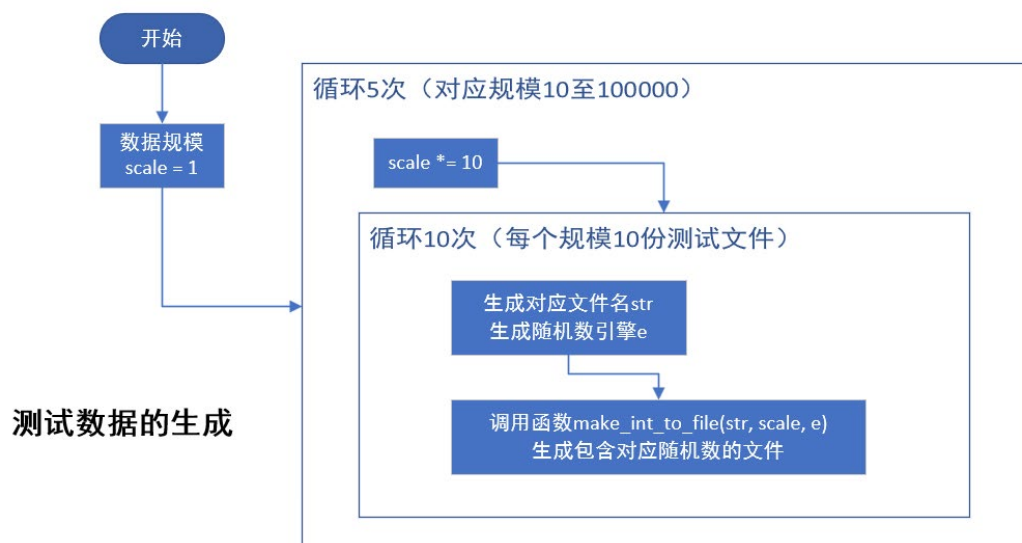
一、实验目的

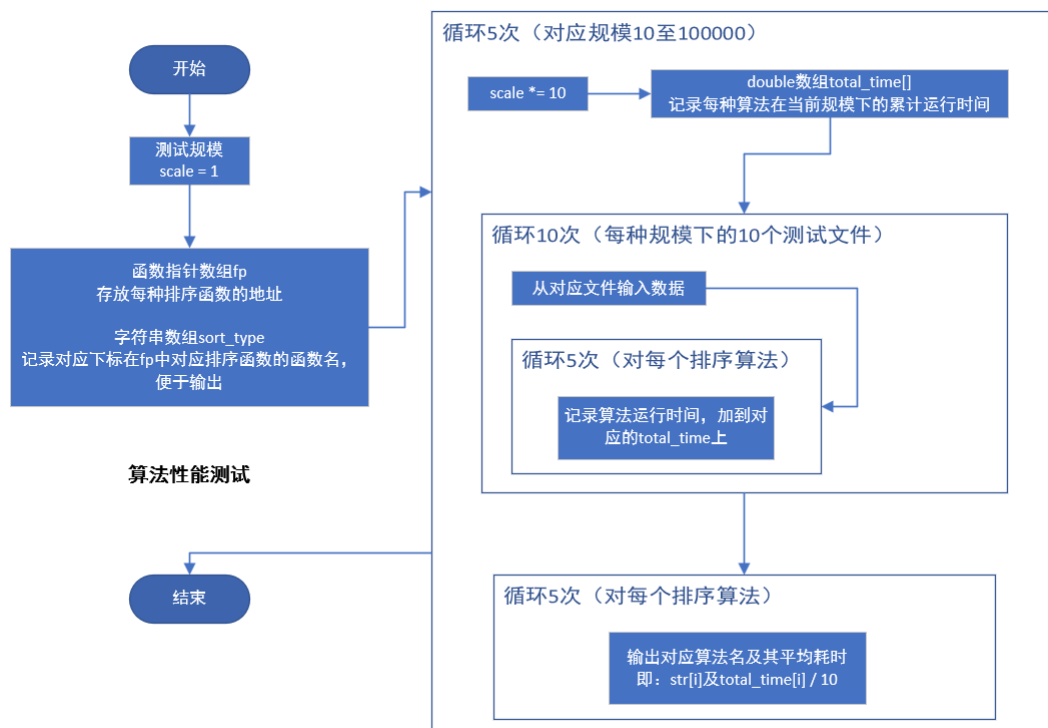
排序是计算机科学中的常见任务，它将一组无序的数据元素按照某种规则重新排列，以使得数据呈现有序的状态，便于后续的查找、统计和分析等操作。当数据量较小时，将数据全部读入内存并进行排序的算法称为内存排序算法，常见的内存排序算法有：插入排序、冒泡排序、归并排序、快速排序、堆排序、基数排序等。本实验要求设计并实现上述内存排序算法并比较其运行速度。

二、实验要求及实验环境

1. 从文本文件中将两行数据读入内存，其中第一行有一个整数 n ($n \leq 100000$)，表示待排序序列的长度，第二行有 n 个整数，用空格隔开，表示待排序序列
 2. 实现归并排序、快速排序算法，输出排序好的序列，并记录算法运行时间
 3. 实现选择排序算法或插入排序算法，并将其运行时间与归并排序、快速排序算法比较，随机生成多个适当规模的数据进行实验并绘制折线图，反映不同算法运行时间随着输入规模的变化趋势，并与理论分析结果进行比较
- ## 三、设计思想
- （本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系、核心算法的主要步骤）

1. 逻辑设计





2. 物理设计（即存储结构设计）

仅使用数组作为数据的存储结构

四、测试结果（包括测试数据、结果数据及结果的简单分析和结论，可以用截图得形式贴入此报告）

1、测试数据的生成

make_input.cpp

```

1.  #include "./src/sort.hpp"
2.
3.  using namespace std;
4.
5.  int main() {
6.      int scale = 1;
7.
8.      for (int i = 1; i <= 5; ++i) {
9.          scale *= 10;
10.
11.          string title = "./inputs/input_" + to_string(scale) + '_';
12.          string suffix = ".txt";
13.          default_random_engine e;
14.          e.seed(time(0));
15.          for (int j = 0; j < 10; ++j)
  
```

```

16.         My::make_int_to_file(title + to_string(j) + suffix, s
           cale, e);
17.     }
18.
19.     return 0;
20. }

```

My::make_int_to_file

```

1.     namespace My {
2.
3.         /**
4.          * @brief generate random integer
5.          * @param dest output stream destination
6.          * @param length generated number count
7.          * @note separated with space (aka char ' ')
8.          */
9.         void make_int(::std::ostream& dest, int length, ::std::default_r
           andom_engine& e) {
10.             ::std::uniform_int_distribution<int> u(INT32_MIN, INT32_MAX);
11.             for (int i = 0; i < length; ++i) {
12.                 dest << u(e) << ' ';
13.             }
14.         }
15.
16.         /**
17.          * @brief generate random integer to file
18.          * @param path file path
19.          * @param length generated number count
20.          * @note output: first line the <length>, second line numbers
21.          */
22.         void make_int_to_file(const ::std::string& path, int length, ::s
           td::default_random_engine& e) {
23.             ::std::ofstream file(path);
24.             file << length << '\n';
25.             make_int(file, length, e);
26.             file.close();
27.         }
28.
29.         .....
30.     }

```

运行结果:

data (D:) > File > 大二秋 > DSA > 实验4 排序 > sort > inputs

在 inputs 中搜索

名称	修改日期	类型	大小
input_10_0.txt	2023/11/7 周二 18:24	文本文档	1 KB
input_10_1.txt	2023/11/7 周二 18:24	文本文档	1 KB
input_10_2.txt	2023/11/7 周二 18:24	文本文档	1 KB
input_10_3.txt	2023/11/7 周二 18:24	文本文档	1 KB
input_10_4.txt	2023/11/7 周二 18:24	文本文档	1 KB
input_10_5.txt	2023/11/7 周二 18:24	文本文档	1 KB
input_10_6.txt	2023/11/7 周二 18:24	文本文档	1 KB
input_10_7.txt	2023/11/7 周二 18:24	文本文档	1 KB
input_10_8.txt	2023/11/7 周二 18:24	文本文档	1 KB
input_10_9.txt	2023/11/7 周二 18:24	文本文档	1 KB
input_100_0.txt	2023/11/7 周二 18:24	文本文档	2 KB
input_100_1.txt	2023/11/7 周二 18:24	文本文档	2 KB
input_100_2.txt	2023/11/7 周二 18:24	文本文档	2 KB
input_100_3.txt	2023/11/7 周二 18:24	文本文档	2 KB
input_100_4.txt	2023/11/7 周二 18:24	文本文档	2 KB
input_100_5.txt	2023/11/7 周二 18:24	文本文档	2 KB

input_100_0.txt

文件

编辑

查看

100
1247300606 1859825283 1671744721 -1040846729
750244014 -1348987962 -1052614918 -188958140 -1125649019 -1979355728 -469883481 1046520524
790275455 -2002390406 555637636 92666246 -1226247759
1033189203 -234861389 -281909366 -2123980912 -1233488554 -1782404765 -216604729 829373820
957557854 -1011700430 -1208205682 -1651866128 837997288 275103886 -2000073667 -657142898 122497801 2072177478
1054023392 -2084238476 -128910497 -817634410 -1480754070 -1908502176 785153956 -267214629 1497044859
1929667692 -1177632934
1890147033 -1253879950 -359034007 -1902418161 -1286161291 -323032246 -700354247 -1851985010 -514215570
1195070578 -215374160 1360394782 447916572 1177170287 -419049105 1864020421 1345713992 -1340880468 806431310
432523357 -1944223411 2090557503 698649421 -1104146207 875376602 -1258480998 785874122 346702954 228653910
2075113371 777953642 1591744855 1953304380 -1910890968 2039254147 -165481997 -1465613723
1862693904 -1042533298 -1749845600 -1187507011 -791062128 1182094692
1194152765 -902045933 -1218231299 -176791013 965220511 -1112460900 174632357 2044626239 1347430768 630031215
540288494

正确生成了对应测试文件

算法运行测试

运行程序

```
1.  int main(){
2.      int scale = 1;
3.
4.      void (*fp[5]) (int*, int) = {
5.          My::selection_sort<int>,
6.          My::insertion_sort<int>,
7.          My::quick_sort<int>,
8.          My::merge_sort<int>,
9.          My::heap_sort<int>
10.     };
11.
12.     string sort_type[5] = {
13.         "selection_sort",
14.         "insertion_sort",
15.         "quick_sort",
16.         "merge_sort",
17.         "heap_sort"
18.     };
19.
20.     double total_time[5] = {0};
21.     string prefix = "./inputs/input_";
22.     string suffix = ".txt";
23.
24.     for (int i = 1; i <= 5; ++i) { // for every scale
25.         scale *= 10;
26.
27.         // clear
28.         for (int m = 0; m < 5; ++m)
29.             total_time[m] = 0;
30.
31.         for (int j = 0; j < 10; ++j) { // for every file in same
32.             scale
33.             string filename = prefix + to_string(scale) + '_' + t
34.                 o_string(j) + suffix;
35.
36.             // read file
37.             ifstream input(filename);
38.             int n = 0;
39.             input >> n;
40.             for (int m = 0; m < n; ++m)
41.                 input >> arr[m];
42.         }
43.         for (int m = 0; m < 5; ++m)
44.             total_time[m] += clock() - start;
45.     }
46.     for (int i = 0; i < 5; ++i)
47.         cout << sort_type[i] << " " << total_time[i] << endl;
48. }
```

```

40.
41.         for (int k = 0; k < 5; ++k) { // for every sort way
42.             memcpy(temp, arr, sizeof(int) * scale);
43.             auto start = chrono::system_clock::now();
44.
45.             fp[k](temp, scale);
46.
47.             auto end = chrono::system_clock::now();
48.             auto duration = chrono::duration_cast<chrono::mic
roseconds>(end - start);
49.             total_time[k] += double(duration.count()) * chron
o::microseconds::period::num / chrono::microseconds::period
::den;
50.         }
51.     }
52.
53.     cout << "In scale " << scale << '\n';
54.     for (int m = 0; m < 5; ++m) {
55.         cout << sort_type[m] << ": " << total_time[m] / 10 <<
"s\n";
56.     }
57.     cout << endl;
58. }
59.
60.     return 0;
61. }

```

输出结果

```

(base) PS D:\File\大二秋\DSA\实验4 排序\sort> .\main.exe
In scale 10
selection_sort: 0 s
insertion_sort: 0 s
quick_sort: 0 s
merge_sort: 0 s
heap_sort: 0 s

In scale 100
selection_sort: 0 s
insertion_sort: 0 s
quick_sort: 0 s
merge_sort: 0 s
heap_sort: 0 s

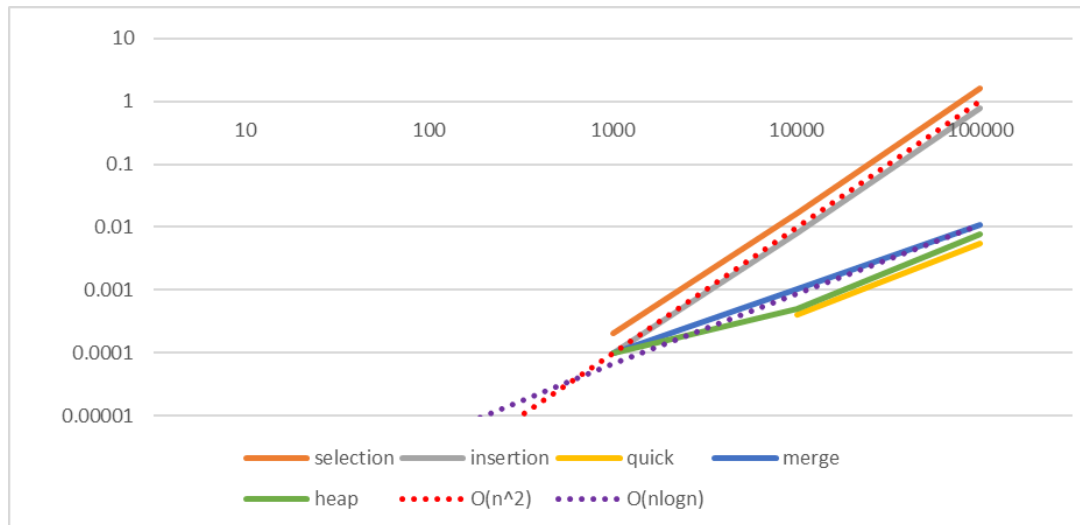
In scale 1000
selection_sort: 0.0002 s
insertion_sort: 0.0002005 s
quick_sort: 0 s
merge_sort: 0.0001011 s
heap_sort: 0.0002007 s

In scale 10000
selection_sort: 0.0164953 s
insertion_sort: 0.0079283 s
quick_sort: 0.0003952 s
merge_sort: 0.0011105 s
heap_sort: 0.0004923 s

In scale 100000
selection_sort: 1.60312 s
insertion_sort: 0.794793 s
quick_sort: 0.00535 s
merge_sort: 0.0110809 s
heap_sort: 0.0075119 s

```

绘制图表



实际数据与理论符合性较好

五、经验体会与不足

加深了对各排序算法的理解

六、附录：源代码（带注释）

main.cpp

```
1.  #define DEBUG 0
2.
3.  #include "../src/sort.hpp"
4.  #include <chrono>
5.
6.  using namespace std;
7.
8.  int arr[100007];
9.  int temp[100007];
10.
11. #if DEBUG // DEBUG == 1
12.
13. int main() {
14.     ifstream input_1("../inputs/input_100_0.txt");
15.     ifstream input_2("../inputs/input_100_1.txt");
16.     int n = 0;
17.
18.     input_1 >> n;
19.     for (int i = 0; i < n; ++i)
20.         input_1 >> arr[i];
21.
22.     auto start = chrono::system_clock::now();
```



```

23.     My::quick_sort(arr, 0, 99);
24.     auto end = chrono::system_clock::now();
25.     auto duration = chrono::duration_cast<chrono::microseconds>(e
        nd - start);
26.     double t1 = double(duration.count()) * chrono::microseconds::
        period::num / chrono::microseconds::period::den;
27.
28.     cout << "sorted by quick_sort\n";
29.     for (int i = 0; i < n; ++i)
30.         cout << arr[i] << '\n';
31.     cout << "using time: " << t1 << " seconds" << endl;
32.
33.     input_2 >> n;
34.     for (int i = 0; i < n; ++i)
35.         input_2 >> arr[i];
36.
37.     start = chrono::system_clock::now();
38.     My::merge_sort(arr, 0, 99);
39.     end = chrono::system_clock::now();
40.     duration = chrono::duration_cast<chrono::microseconds>(end -
        start);
41.     double t2 = double(duration.count()) * chrono::microseconds::
        period::num / chrono::microseconds::period::den;
42.
43.     cout << "sorted by merge_sort\n";
44.     for (int i = 0; i < n; ++i)
45.         cout << arr[i] << '\n';
46.     cout << "using time: " << t2 << " seconds" << endl;
47. }
48.
49. #else // DEBUG == 0
50.
51. int main(){
52.     int scale = 1;
53.
54.     void (*fp[5]) (int*, int) = {
55.         My::selection_sort<int>,
56.         My::insertion_sort<int>,
57.         My::quick_sort<int>,
58.         My::merge_sort<int>,
59.         My::heap_sort<int>
60.     };
61.
62.     string sort_type[5] = {

```

```

63.         "selection_sort",
64.         "insertion_sort",
65.         "quick_sort",
66.         "merge_sort",
67.         "heap_sort"
68.     };
69.
70.     double total_time[5] = {0};
71.     string prefix = "./inputs/input_";
72.     string suffix = ".txt";
73.
74.     for (int i = 1; i <= 5; ++i) { // for every scale
75.         scale *= 10;
76.
77.         // clear
78.         for (int m = 0; m < 5; ++m)
79.             total_time[m] = 0;
80.
81.         for (int j = 0; j < 10; ++j) { // for every file in same
            scale
82.             string filename = prefix + to_string(scale) + '_' + t
                o_string(j) + suffix;
83.
84.             // read file
85.             ifstream input(filename);
86.             int n = 0;
87.             input >> n;
88.             for (int m = 0; m < n; ++m)
89.                 input >> arr[m];
90.
91.             for (int k = 0; k < 5; ++k) { // for every sort way
92.                 memcpy(temp, arr, sizeof(int) * scale);
93.                 auto start = chrono::system_clock::now();
94.
95.                 fp[k](temp, scale);
96.
97.                 auto end = chrono::system_clock::now();
98.                 auto duration = chrono::duration_cast<chrono::mic
                    roseconds>(end - start);
99.                 total_time[k] += double(duration.count()) * chron
                    o::microseconds::period::num / chrono::microseconds::period
                    ::den;
100.            }
101.        }

```

```

102.
103.         cout << "In scale " << scale << '\n';
104.         for (int m = 0; m < 5; ++m) {
105.             cout << sort_type[m] << ": " << total_time[m] / 10 <<
               " s\n";
106.         }
107.         cout << endl;
108.     }
109.
110.     return 0;
111. }
112.
113. #endif // DEBUG

```

make_input.cpp

```

1.     #include "./src/sort.hpp"
2.
3.     using namespace std;
4.
5.     int main() {
6.         int scale = 1;
7.
8.         for (int i = 1; i <= 5; ++i) {
9.             scale *= 10;
10.
11.             string title = "./inputs/input_" + to_string(scale) + '_';
12.             string suffix = ".txt";
13.             default_random_engine e;
14.             e.seed(time(0));
15.             for (int j = 0; j < 10; ++j)
16.                 My::make_int_to_file(title + to_string(j) + suffix, s
                    cale, e);
17.         }
18.
19.         return 0;
20.     }

```

sort.hpp

```

1.     #ifndef _SORTS_HPP_
2.     #define _SORTS_HPP_
3.
4.     #include <iostream>
5.     #include <functional>
6.     #include <random>
7.     #include <limits>

```

```

8.  #include <ctime>
9.  #include <string>
10. #include <fstream>
11.
12. namespace My {
13.
14.     /**
15.      * @brief generate random integer
16.      * @param dest output stream destination
17.      * @param length generated number count
18.      * @note separated with space (aka char ' ')
19.      */
20.     void make_int(::std::ostream& dest, int length, ::std::default_random_engine& e) {
21.         ::std::uniform_int_distribution<int> u(INT32_MIN, INT32_MAX);
22.         for (int i = 0; i < length; ++i) {
23.             dest << u(e) << ' ';
24.         }
25.     }
26.
27.     /**
28.      * @brief generate random integer to file
29.      * @param path file path
30.      * @param length generated number count
31.      * @note output: first line the <length>, second line numbers
32.      */
33.     void make_int_to_file(const ::std::string& path, int length, ::std::default_random_engine& e) {
34.         ::std::ofstream file(path);
35.         file << length << '\n';
36.         make_int(file, length, e);
37.         file.close();
38.     }
39.
40.     template <typename Elem, class CmpFunc>
41.     void selection_sort(Elem* arr, int len, CmpFunc cmp) {
42.         for (int i = 0; i < len; i++) {
43.             int l_index = i;
44.             for (int j = i + 1; j < len; j++)
45.                 if (cmp(arr[j], arr[l_index]))
46.                     l_index = j;
47.             ::std::swap(arr[i], arr[l_index]);
48.         }
49.     }

```

```

50.
51.     template <typename Elem>
52.     void selection_sort(Elem* arr, int len) {
53.         selection_sort(arr, len, ::std::less<Elem>());
54.     }
55.
56.     template <typename Elem, class CmpFunc>
57.     void insertion_sort(Elem* arr, int len, CmpFunc cmp) {
58.         for (int i = 1; i < len; i++) {
59.             Elem temp = arr[i];
60.             int j = i - 1;
61.             while (cmp(temp, arr[j]) && j >= 0) {
62.                 arr[j + 1] = arr[j];
63.                 --j;
64.             }
65.             arr[j + 1] = temp;
66.         }
67.     }
68.
69.     template <typename Elem>
70.     void insertion_sort(Elem* arr, int len) {
71.         insertion_sort(arr, len, ::std::less<Elem>());
72.     }
73.
74.     template <typename Elem, class CmpFunc>
75.     void merge_sort(Elem* arr, int l, int r, CmpFunc cmp) {
76.         if (l >= r)
77.             return;
78.         int mid = l + (r - l) / 2;
79.
80.         merge_sort(arr, l, mid, cmp);
81.         merge_sort(arr, mid + 1, r, cmp);
82.
83.         Elem* temp = new Elem[r - l + 1];
84.         int i = l, j = mid + 1, total = 0;
85.         while (i <= mid && j <= r) {
86.             if (cmp(arr[i], arr[j]))
87.                 temp[total++] = arr[i++];
88.             else
89.                 temp[total++] = arr[j++];
90.         }
91.
92.         if (i <= mid)
93.             memcpy(temp + total, arr + i, (mid - i + 1) * sizeof(Elem));

```

```

94.     if (j <= r)
95.         memcpy(temp + total, arr + j, (r - j + 1) * sizeof(Elem));
96.         memcpy(arr + l, temp, (r - l + 1) * sizeof(Elem));
97.
98.     delete[] temp;
99. }
100.
101. template <typename Elem>
102. void merge_sort(Elem* arr, int l, int r) {
103.     merge_sort(arr, l, r, ::std::less<Elem>());
104. }
105.
106. template <typename Elem>
107. void merge_sort(Elem* arr, int len) {
108.     merge_sort(arr, 0, len - 1);
109. }
110.
111. template <typename Elem, class CmpFunc>
112. void quick_sort(Elem* arr, int l, int r, CmpFunc cmp) {
113.     Elem pivot = arr[l + (r - l) / 2];
114.     int i = l, j = r;
115.
116.     do {
117.         while (cmp(arr[i], pivot))
118.             ++i;
119.         while (cmp(pivot, arr[j]))
120.             --j;
121.         if (i <= j)
122.             ::std::swap(arr[i++], arr[j--]);
123.     } while (i <= j);
124.
125.     if (l < j)
126.         quick_sort(arr, l, j, cmp);
127.     if (i < r)
128.         quick_sort(arr, i, r, cmp);
129. }
130.
131. template <typename Elem>
132. void quick_sort(Elem* arr, int l, int r) {
133.     quick_sort(arr, l, r, ::std::less<Elem>());
134. }
135.
136. template <typename Elem>
137. void quick_sort(Elem* arr, int len) {

```

```

138.     quick_sort(arr, 0, len - 1);
139. }
140.
141. template <typename Elem, class CmpFunc>
142. void heapify (Elem* arr, int len, int father, CmpFunc cmp) {
143.     int child = father * 2 + 1;
144.     // sink down process
145.     while (child < len) {
146.         // get the "maximum" child (depend on the cmp func)
147.         if (child + 1 < len && cmp(arr[child], arr[child + 1]))
148.             child++;
149.
150.         // if father is "bigger" than child (depend on the cmp func)
151.         if (cmp(arr[child], arr[father]))
152.             // sink down over
153.             return;
154.         else {
155.             // swap father with the "bigger" child (depend on the cmp func)
156.             ::std::swap(arr[child], arr[father]);
157.             // child become father
158.             father = child;
159.             child = father * 2 + 1;
160.         }
161.     }
162. }
163.
164. template <typename Elem, class CmpFunc>
165. void heap_sort(Elem* arr, int len, CmpFunc cmp) {
166.     for (int i = (len - 1) / 2; i >= 0; i--)
167.         heapify(arr, len, i, cmp);
168.
169.     for (int i = len - 1; i >= 0; i--) {
170.         ::std::swap(arr[0], arr[i]);
171.         heapify(arr, i, 0, cmp);
172.     }
173. }
174.
175. template <typename Elem>
176. void heap_sort(Elem* arr, int len) {
177.     heap_sort(arr, len, ::std::less<Elem>());
178. }
179.
180. } // namespace My

```

181.

182. `#endif` `// _SORTS_HPP_`