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。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

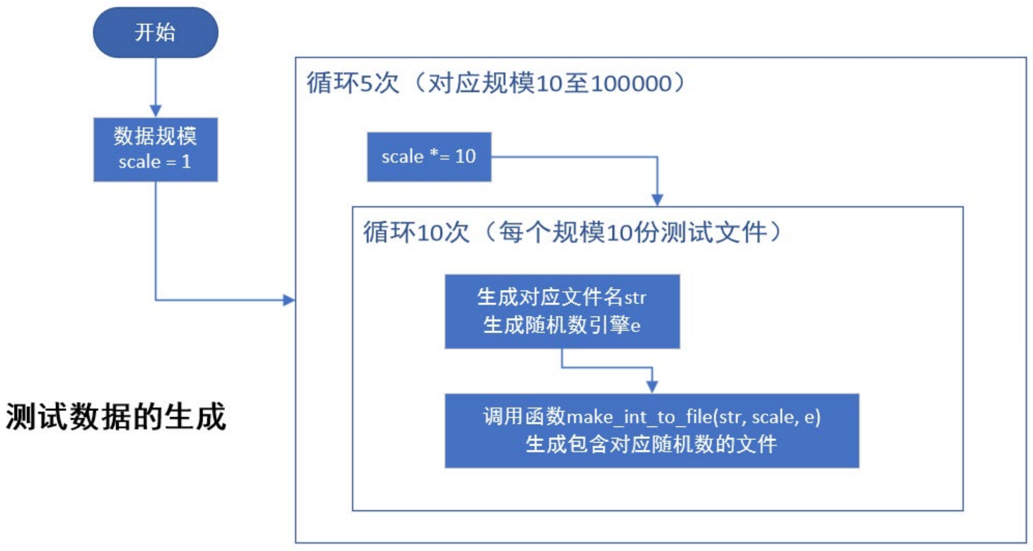
1. 从文本文件中将两行数据读入内存，其中第一行有一个整数 n(n≤100000)，表示待排序序列的长度，第二行有 n 个整数，用空格隔开，表示待排序序列

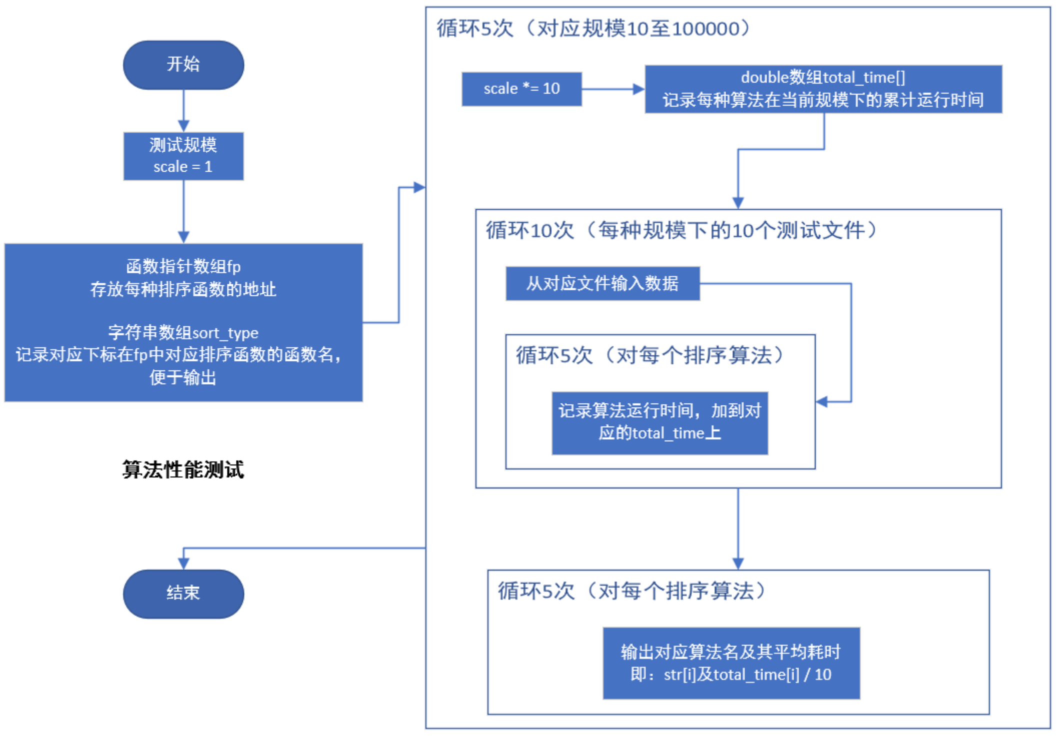
2. 实现归并排序、快速排序算法，输出排序好的序列，并记录算法运行时间

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





开始
测试规模 scale = 1
函数指针数组fp 存放每种排序函数的地址
字符串数组sort_type 记录对应下标在fp中对应排序函数的函数名，便于输出
算法性能测试
结束
循环5次（对应规模10至100000）
scale *= 10
double数组total_time[] 记录每种算法在当前规模下的累计运行时间
循环10次（每种规模下的10个测试文件）
从对应文件输入数据
循环5次（对每个排序算法）
记录算法运行时间，加到对应的total_time上
循环5次（对每个排序算法）
输出对应算法名及其平均耗时 即：str[i]及total_time[i] / 10

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

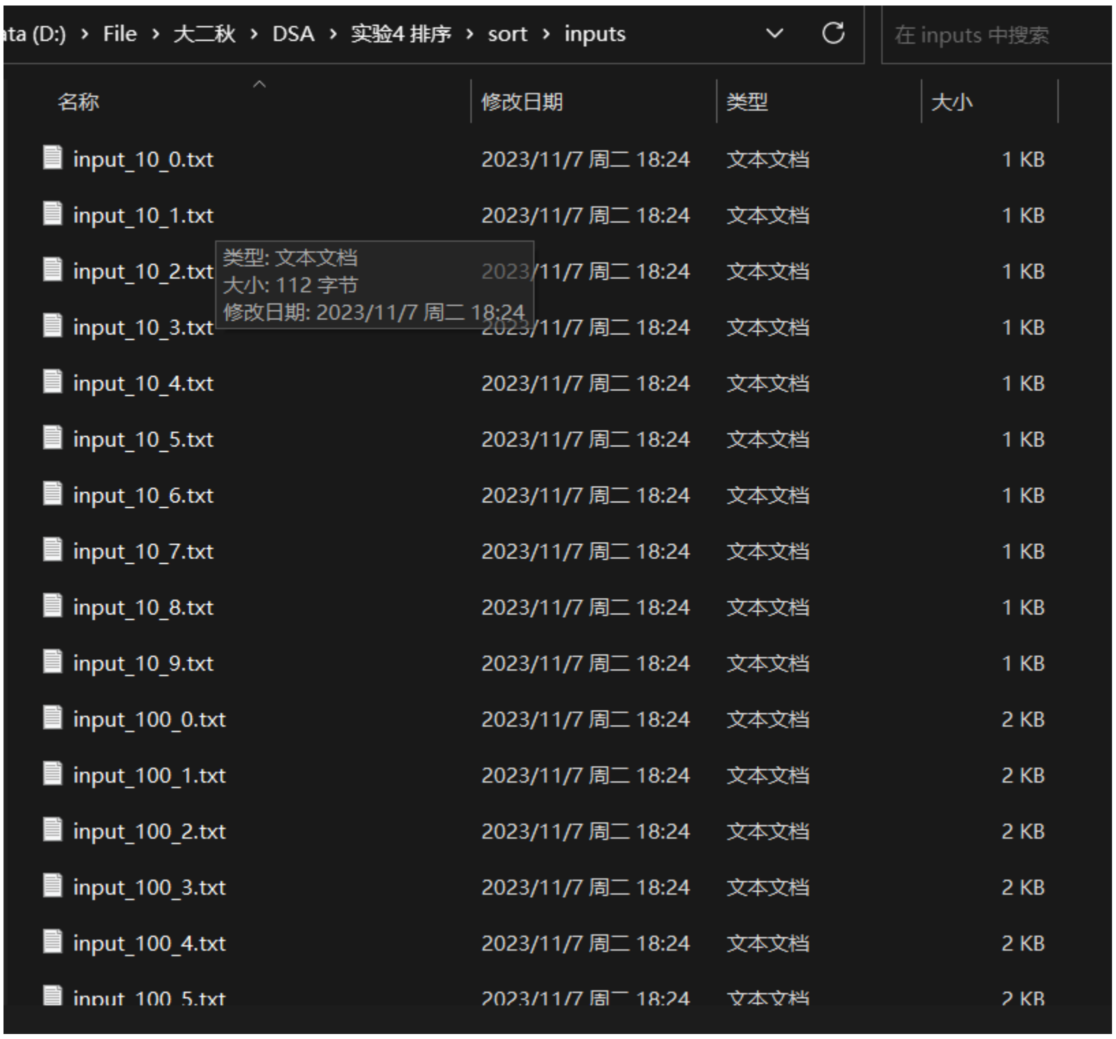
```cpp
#include "./src/sort.hpp"

using namespace std;

int main() {
    int scale = 1;

    for (int i = 1; i <= 5; ++i) {
        scale *= 10;

        string title = "./inputs/input_" + to_string(scale) + '_';
        string suffix = ".txt";
        default_random_engine e;
        e.seed(time(0));
        for (int j = 0; j < 10; ++j)
```

```
            My::make_int_to_file(title + to_string(j) + suffix, scale, e);
        }

        return 0;
    }
```

My::make_int_to_file

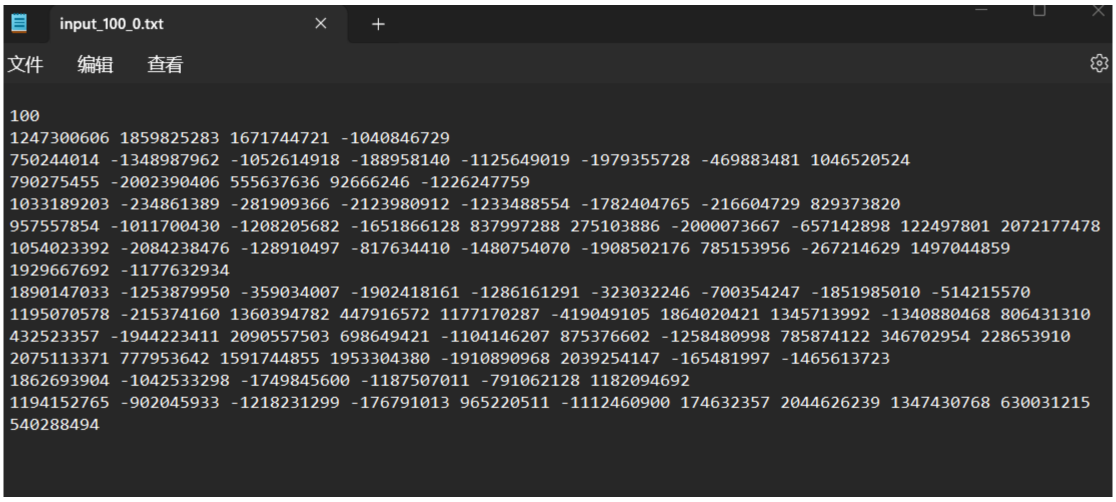
```
namespace My {

 /**
  * @brief generate random integer
  * @param dest output stream destination
  * @param length generated number count
  * @note separated with space (aka char ' ')
 */
 void make_int(::std::ostream& dest, int length, ::std::default_random_engine& e) {
  ::std::uniform_int_distribution<int> u(INT32_MIN, INT32_MAX);
  for (int i = 0; i < length; ++i) {
   dest << u(e) << ' ';
  }
 }

 /**
  * @brief generate random integer to file
  * @param path file path
  * @param length generated number count
  * @note output: first line the <length>, second line numbers
 */
 void make_int_to_file(const ::std::string& path, int length, ::std::default_random_engine& e) {
  ::std::ofstream file(path);
  file << length << '\n';
  make_int(file, length, e);
  file.close();
 }

 ............
}
```

运行结果:



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

类型: 文本文档
大小: 112 字节
修改日期: 2023/11/7 周二 18:24



```
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
```

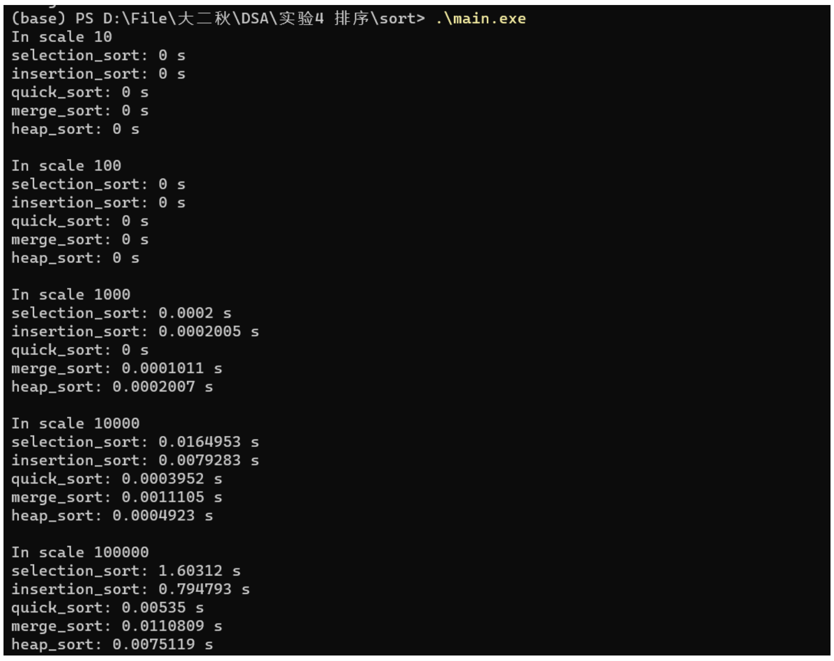
正确生成了对应测试文件

算法运行测试

运行程序

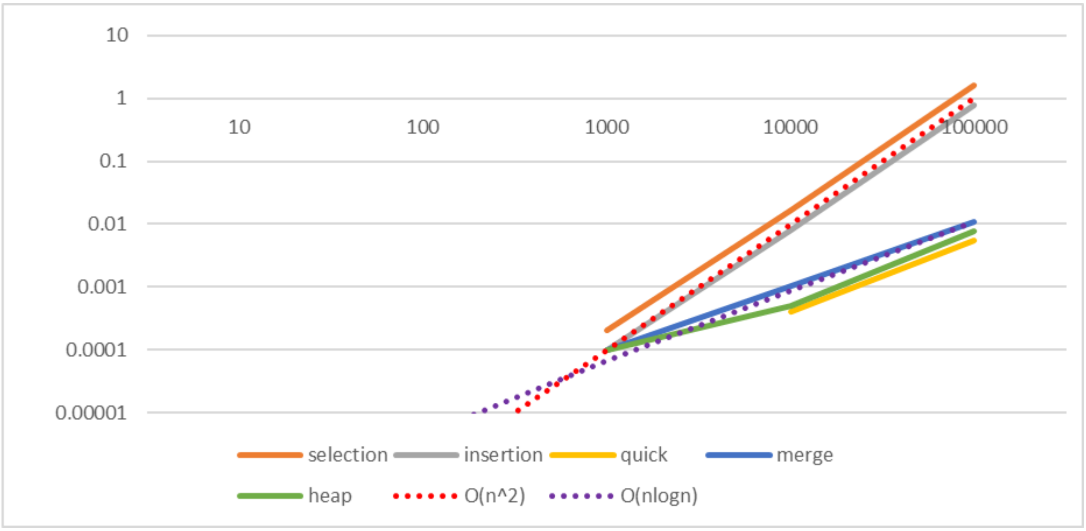
```cpp
int main(){
    int scale = 1;

    void (*fp[5]) (int*, int) = {
        My::selection_sort<int>,
        My::insertion_sort<int>,
        My::quick_sort<int>,
        My::merge_sort<int>,
        My::heap_sort<int>
    };

    string sort_type[5] = {
        "selection_sort",
        "insertion_sort",
        "quick_sort",
        "merge_sort",
        "heap_sort"
    };

    double total_time[5] = {0};
    string prefix = "./inputs/input_";
    string suffix = ".txt";

    for (int i = 1; i <= 5; ++i) { // for every scale
        scale *= 10;

        // clear
        for (int m = 0; m < 5; ++m)
            total_time[m] = 0;

        for (int j = 0; j < 10; ++j) { // for every file in same scale
            string filename = prefix + to_string(scale) + '_' + to_string(j) + suffix;

            // read file
            ifstream input(filename);
            int n = 0;
            input >> n;
            for (int m = 0; m < n; ++m)
                input >> arr[m];

            for (int k = 0; k < 5; ++k) { // for every sort way
                memcpy(temp, arr, sizeof(int) * scale);
                auto start = chrono::system_clock::now();

                fp[k](temp, scale);

                auto end = chrono::system_clock::now();
                auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
                total_time[k] += double(duration.count()) * chrono::microseconds::period::num / chrono::microseconds::period::den;
            }
        }

        cout << "In scale " << scale << '\n';
        for (int m = 0; m < 5; ++m) {
            cout << sort_type[m] << ": " << total_time[m] / 10 << " s\n";
        }
        cout << endl;
    }

    return 0;
}
```

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

```cpp
#define DEBUG 0

#include "./src/sort.hpp"
#include <chrono>

using namespace std;

int arr[100007];
int temp[100007];

#if DEBUG // DEBUG == 1

int main() {
    ifstream input_1("./inputs/input_100_0.txt");
    ifstream input_2("./inputs/input_100_1.txt");
    int n = 0;

    input_1 >> n;
    for (int i = 0; i < n; ++i)
        input_1 >> arr[i];

    auto start = chrono::system_clock::now();
    My::quick_sort(arr, 0, 99);
    auto end = chrono::system_clock::now();
    auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
    double t1 = double(duration.count()) * chrono::microseconds::period::num / chrono::microseconds::period::den;

    cout << "sorted by quick_sort\n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << '\n';
    cout << "using time: " << t1 << " seconds" << endl;

    input_2 >> n;
    for (int i = 0; i < n; ++i)
        input_2 >> arr[i];

    start = chrono::system_clock::now();
    My::merge_sort(arr, 0, 99);
    end = chrono::system_clock::now();
    duration = chrono::duration_cast<chrono::microseconds>(end - start);
    double t2 = double(duration.count()) * chrono::microseconds::period::num / chrono::microseconds::period::den;

    cout << "sorted by merge_sort\n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << '\n';
    cout << "using time: " << t2 << " seconds" << endl;
}

#else // DEBUG == 0

int main(){
    int scale = 1;

    void (*fp[5]) (int*, int) = {
        My::selection_sort<int>,
        My::insertion_sort<int>,
        My::quick_sort<int>,
        My::merge_sort<int>,
        My::heap_sort<int>
    };

    string sort_type[5] = {
        "selection_sort",
        "insertion_sort",
        "quick_sort",
        "merge_sort",
        "heap_sort"
    };

    double total_time[5] = {0};
    string prefix = "./inputs/input_";
    string suffix = ".txt";

    for (int i = 1; i <= 5; ++i) { // for every scale
        scale *= 10;

        // clear
        for (int m = 0; m < 5; ++m)
            total_time[m] = 0;

        for (int j = 0; j < 10; ++j) { // for every file in same scale
            string filename = prefix + to_string(scale) + '_' + to_string(j) + suffix;

            // read file
            ifstream input(filename);
            int n = 0;
            input >> n;
            for (int m = 0; m < n; ++m)
                input >> arr[m];

            for (int k = 0; k < 5; ++k) { // for every sort way
                memcpy(temp, arr, sizeof(int) * scale);
                auto start = chrono::system_clock::now();

                fp[k](temp, scale);

                auto end = chrono::system_clock::now();
                auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
                total_time[k] += double(duration.count()) * chrono::microseconds::period::num / chrono::microseconds::period::den;
            }
        }

        cout << "In scale " << scale << '\n';
        for (int m = 0; m < 5; ++m) {
            cout << sort_type[m] << ": " << total_time[m] / 10 << " s\n";
        }
        cout << endl;
    }

    return 0;
}

#endif // DEBUG
```

make_input.cpp

```cpp
#include "./src/sort.hpp"

using namespace std;

int main() {
    int scale = 1;

    for (int i = 1; i <= 5; ++i) {
        scale *= 10;

        string title = "./inputs/input_" + to_string(scale) + '_';
        string suffix = ".txt";
        default_random_engine e;
        e.seed(time(0));
        for (int j = 0; j < 10; ++j)
            My::make_int_to_file(title + to_string(j) + suffix, scale, e);
    }

    return 0;
}
```

sort.hpp

```cpp
#ifndef _SORTS_HPP_
#define _SORTS_HPP_

#include <iostream>
#include <functional>
#include <random>
#include <limits>
```

```cpp
#include <ctime>
#include <string>
#include <fstream>

namespace My {

 /**
  * @brief generate random integer
  * @param dest output stream destination
  * @param length generated number count
  * @note separated with space (aka char ' ')
 */
 void make_int(::std::ostream& dest, int length, ::std::default_random_engine& e) {
  ::std::uniform_int_distribution<int> u(INT32_MIN, INT32_MAX);
  for (int i = 0; i < length; ++i) {
   dest << u(e) << ' ';
  }
 }

 /**
  * @brief generate random integer to file
  * @param path file path
  * @param length generated number count
  * @note output: first line the <length>, second line numbers
 */
 void make_int_to_file(const ::std::string& path, int length, ::std::default_random_engine& e) {
  ::std::ofstream file(path);
  file << length << '\n';
  make_int(file, length, e);
  file.close();
 }

 template <typename Elem, class CmpFunc>
 void selection_sort(Elem* arr, int len, CmpFunc cmp) {
  for (int i = 0; i < len; i++) {
   int l_index = i;
   for (int j = i + 1; j < len; j++)
    if (cmp(arr[j], arr[l_index]))
     l_index = j;
   ::std::swap(arr[i], arr[l_index]);
  }
 }
```

```

template <typename Elem>
void selection_sort(Elem* arr, int len) {
 selection_sort(arr, len, ::std::less<Elem>());
}

template <typename Elem, class CmpFunc>
void insertion_sort(Elem* arr, int len, CmpFunc cmp) {
 for (int i = 1; i < len; i++) {
  Elem temp = arr[i];
  int j = i - 1;
  while (cmp(temp, arr[j]) && j >= 0) {
   arr[j + 1] = arr[j];
   --j;
  }
  arr[j + 1] = temp;
 }
}

template <typename Elem>
void insertion_sort(Elem* arr, int len) {
 insertion_sort(arr, len, ::std::less<Elem>());
}

template <typename Elem, class CmpFunc>
void merge_sort(Elem* arr, int l, int r, CmpFunc cmp) {
 if (l >= r)
  return;
 int mid = l + (r - l) / 2;

 merge_sort(arr, l, mid, cmp);
 merge_sort(arr, mid + 1, r, cmp);

 Elem* temp = new Elem[r - l + 1];
 int i = l, j = mid + 1, total = 0;
 while (i <= mid && j <= r) {
  if (cmp(arr[i], arr[j]))
   temp[total++] = arr[i++];
  else
   temp[total++] = arr[j++];
 }

 if (i <= mid)
  memcpy(temp + total, arr + i, (mid - i + 1) * sizeof(Elem));
  if (j <= r)
   memcpy(temp + total, arr + j, (r - j + 1) * sizeof(Elem));
  memcpy(arr + l, temp, (r - l + 1) * sizeof(Elem));

  delete[] temp;
 }

template <typename Elem>
void merge_sort(Elem* arr, int l, int r) {
 merge_sort(arr, l, r, ::std::less<Elem>());
}

template <typename Elem>
void merge_sort(Elem* arr, int len) {
 merge_sort(arr, 0, len - 1);
}

template <typename Elem, class CmpFunc>
void quick_sort(Elem* arr, int l, int r, CmpFunc cmp) {
 Elem pivot = arr[l + (r - l) / 2];
 int i = l, j = r;

 do {
  while (cmp(arr[i], pivot))
   ++i;
  while (cmp(pivot, arr[j]))
   --j;
  if (i <= j)
   ::std::swap(arr[i++], arr[j--]);
 } while (i <= j);

 if (l < j)
  quick_sort(arr, l, j, cmp);
 if (i < r)
  quick_sort(arr, i, r, cmp);
}

template <typename Elem>
void quick_sort(Elem* arr, int l, int r) {
 quick_sort(arr, l, r, ::std::less<Elem>());
}

template <typename Elem>
void quick_sort(Elem* arr, int len) {
```

```cpp
    quick_sort(arr, 0, len - 1);
  }

  template <typename Elem, class CmpFunc>
  void heapify (Elem* arr, int len, int father, CmpFunc cmp) {
    int child = father * 2 + 1;
    // sink down process
    while (child < len) {
      // get the "maximum" child (depend on the cmp func)
      if (child + 1 < len && cmp(arr[child], arr[child + 1]))
        child++;

      // if father is "bigger" than child (depend on the cmp func)
      if (cmp(arr[child], arr[father]))
        // sink down over
        return;
      else {
        // swap father with the "bigger" child (depend on the cmp func)
        ::std::swap(arr[child], arr[father]);
        // child become father
        father = child;
        child = father * 2 + 1;
      }
    }
  }

  template <typename Elem, class CmpFunc>
  void heap_sort(Elem* arr, int len, CmpFunc cmp) {
    for (int i = (len - 1) / 2; i >= 0; i--)
      heapify(arr, len, i, cmp);

    for (int i = len - 1; i >= 0; i--) {
      ::std::swap(arr[0], arr[i]);
      heapify(arr, i, 0, cmp);
    }
  }

  template <typename Elem>
  void heap_sort(Elem* arr, int len) {
    heap_sort(arr, len, ::std::less<Elem>());
  }

} // namespace My
```

```

#endif // _SORTS_HPP_
```