华东师范大学计算机科学与技术学院上机实践报告

课程名称：数据结构	**年级**：2022 级	**上机实践成绩**：
指导教师：金健	**姓名**：岳锦鹏	**上机实践时间**：2 学时
上机实践名称：第八章作业	**学号**：10213903403	**上机实践日期**：2023/12/29

一、 实验目的

1. 使用第八章的知识解决排序的综合问题。

二、 实验内容

1. 给定一个序列，使用快速排序将其从小到大进行排列。
2. 给定一个序列，构造一个大根堆。

三、 实验原理

1. 程序设计原理。

四、 实验步骤

1. 问题抽象
2. 编写程序
3. 调试程序
4. 完善总结

五、 调试过程、结果和分析

1. 两道题分别对应了快速排序和堆排序。
2. 快速排序要注意好起点和终点的位置选取，作为比较标准的元素放在最前面，开始位置为数组的开始，开始元素为这个元素后面的一个元素，结束元素为数组末尾的元素。需要维护两个指针，分别从开始往后移动和从末尾往前移动，用来标记下一个要交换的位置。最后这两个指针交汇时的位置就是比较标准的元素需要放置的位置了。
3. 堆排序要注意不同的操作，构造堆时从最后一个非叶子节点开始往前逐个下沉，当然也可以看做向一个空的堆中不断插入来完成构造；插入操作是先插入到末尾然后上浮；弹出操作是将堆顶的元素弹出，之后把末尾的元素放到对顶再下沉。
4. 下沉操作比上浮操作更复杂一点，因为要比较父节点和左右两个子节点，把最大的子节点换上了，而上浮操作只需要和父节点比较就行。
5. 快速排序的输入：
   20
   41 40 19 8 5 5 17 32 39 32 98 95 68 56 60 59 67 94 77 98

输出：

5 5 8 17 19 32 32 39 40 41 56 59 60 67 68 77 94 95 98 98

6. 大根堆的输入：

20

41 40 19 8 5 5 17 32 39 32 98 95 68 56 60 59 67 94 77 98

输出：

逐个插入后，逐个弹出输出的结果为：

98 98 95 94 77 68 67 60 59 56 41 40 39 32 32 19 17 8 5 5

一次性初始化后，逐个弹出输出的结果为：

98 98 95 94 77 68 67 60 59 56 41 40 39 32 32 19 17 8 5 5

六、 总结

1. 在大根堆排序时，如果不使用弹出，而是每次把堆顶的元素和堆末尾的元素交换，之后将堆的长度减一，这样操作完成后就会形成升序排列。如果每次把堆顶的元素弹出，并且输出，这样操作完成后就会形成降序排列。这里使用的是弹出方式。

七、 附件

1. 第八章作业 1.cpp

```cpp
#include <iostream>
#include <cassert>

using namespace std;
#define N 1000
int a[N];
int *_one_process(int *_begin, int *_end){
    int temp = *_begin;
    int len = _end - _begin;
    int *left = _begin, *right = _end;
    for (int *p = _begin + 1; p <= _end; p++) {
        if (*p <= temp) {
            swap(*left++, *p);
        } else {
            swap(*right--, *p);
        }
    }
    assert(left == right);
    *left = temp;
    return left;
}

void _fastsort(int *_begin, int *_end) {
    if (_begin >= _end)
```

```
25          return;
26      int *middle = _one_process(_begin, _end);
27      _fastsort(_begin, middle - 1);
28      _fastsort(middle + 1, _end);
29  }
30
31  void fast_sort(int *a, int len) {
32      _fastsort(a, a + len - 1);
33  }
34
35  int main()
36  {
37      int n;
38      cin >> n;
39      for (int i = 0 ; i < n; i++) cin >> a[i];
40      fast_sort(a, n);
41      for (int i = 0; i < n; i++) cout << a[i] << " ";
42      cout << endl;
43      return 0;
44  }
```

2. 第八章作业 2.cpp

```
1   // 20:50
2   // 22:31
3
4   #include <iostream>
5
6   const int N = 1000;
7
8   class BigHeap {
9   private:
10      int *_internal_array;
11      int len = 0;
12      bool external = false;
13
14      int get_left_child(int index) {
15          return this->_internal_array[index * 2];
16      }
17
18      int get_right_child(int index) {
```

```
19          return this->_internal_array[index * 2 + 1];
20      }
21
22      int get_parent(int index) {
23          return this->_internal_array[index / 2];
24      }
25      void adjust_up(int index) {
26          while (index > 1 && this->_internal_array[index] >
            ↪  this->get_parent(index)) {
27              std::swap(this->_internal_array[index],
                ↪  this->_internal_array[index / 2]);
28              index /= 2;
29          }
30      }
31
32      void adjust_down(int index) {
33          while (index * 2 <= this->len) {
34              int target = index;
35              if (this->_internal_array[target] <
                ↪  this->get_left_child(index))
                ↪  // 如果左节点更大
36                  target = index * 2;
                    ↪  // 左节点换上来
37              if (index * 2 + 1 <= this->len &&
                ↪  this->_internal_array[target] <
                ↪  this->get_right_child(index)) // 如果右节点更大
38                  target = index * 2 + 1;
                    ↪  // 右节点换上来
39              if (index == target)
40                  break;
41              // 如果满足了当前节点比左右子节点大，就不用继续下沉了
42
43              std::swap(this->_internal_array[index],
                ↪  this->_internal_array[target]);
44              index = target;
45          }
46      }
47      // 好像不太对，算了不用了，叶子节点直接判断就行了
48      int get_last_non_leaf_index(int len) {
49          int pow_of_2 = 1;
50          int times = 0;
51          while (pow_of_2 < len) {
```

```
52              pow_of_2 *= 2; // 也可以写成 pow_of_2 <<= 2;
53              times++;
54          }
55          return times;
56      }
57
58  public:
59      BigHeap() { // 构造函数，在初始化时被调用
60          this->_internal_array = new int[N];
61      }
62      BigHeap(int *external_array, int len) { // 也是构造函数，但是使用外部的
        ↪  数组初始化，数组要从 1 开始
63          this->_internal_array = external_array;
64          this->len = len;
65          for (int i = this->len; i >= 1; i--) {
66              adjust_down(i);
67          }
68          this->external = true;
69      }
70      ~BigHeap() { // 析构函数，在对象被删除时被调用
71          if (!this->external)
72              delete this->_internal_array;
73      }
74
75      int get_length() {
76          return this->len;
77      }
78
79      void insert(int num) {
80          this->_internal_array[++this->len] = num;
81          int index = this->len;
82          adjust_up(index);
83      }
84
85      int pop() {
86          int result = this->_internal_array[1];
87          this->_internal_array[1] = this->_internal_array[this->len--];
88          adjust_down(1);
89          return result;
90      }
91  };
92
```

```cpp
int a[N];
int len;

int main() {
    std::cin >> len;
    for (int i = 1; i <= len; i++)
        std::cin >> a[i];

    BigHeap big_heap_internal;
    for (int i = 1; i <= len; i++) {
        big_heap_internal.insert(a[i]);
    }
    std::cout << " 逐个插入后，逐个弹出输出的结果为: " << std::endl;
    while (big_heap_internal.get_length() > 0) {
        std::cout << big_heap_internal.pop() << " ";
    }
    std::cout << std::endl;

    BigHeap big_heap_external(a, len);
    std::cout << " 一次性初始化后，逐个弹出输出的结果为: " << std::endl;
    while (big_heap_external.get_length() > 0) {
        std::cout << big_heap_external.pop() << " ";
    }
    std::cout << std::endl;
    return 0;
}
```