20161607\_沈志立\_2班\_第七周作业

7.8

在最差的情况下，栈的深度为n；

把较大的partition放在第一位，然后是较小的partition，所以较小的partition可以先得到处理；

可以减为log n；

7.9

2 6 4 0 1 3 5 7 首先，把0设为轴值

2 6 4 7 1 3 5 0 第一次交换结束

0 6 4 7 1 3 5 2 第一分割结束，并把1设为轴值

0 6 4 7 2 3 5 1 第二次交换结束

0 1 4 7 2 3 5 6 第二分割结束，并把2设为轴值

0 1 4 7 6 3 5 2 第三次交换结束

0 1 2 7 6 3 5 4 第三分割结束，并把3设为轴值

0 1 2 7 6 4 5 3 第四次交换结束

0 1 2 3 6 4 5 7 第四分割结束，并把4设为轴值

0 1 2 3 6 7 5 4 第五次交换结束

0 1 2 3 4 7 5 6 第五分割结束，并把5设为轴值

0 1 2 3 4 5 7 6 第六次交换结束

0 1 2 3 4 5 6 7 第六分割结束，并把6设为轴值

0 1 2 3 4 5 7 6 第六次交换结束

0 1 2 3 4 5 6 7 第六分割结束

7.16

（a）代码：

1. **void** sort3(ELEM A){
2. **if**(A[1]<A[0])
3. **if**(A[2]<A[0])
4. **if**(A[2]<A[1])       //321
5. swap(A[0],A[2]);
6. **else**{               //312
7. swap(A[0],A[1]);
8. swap(A[1],A[2]);
9. }
10. **else**{                   //213
11. swap(A[0],A[1]);
12. }
13. **else**
14. **if**(A[2]<A[1])
15. **if**(A[2]<A[0]){      //231
16. swap(A(0),A(2));
17. swap(A(1),A(1));
18. }
19. **else**                //132
20. swap(A[1],A[2]);
21. **else**     v              //123
22. }

最好情况下：两次比较

最差情况下：三次比较

平均情况下：（3+3+2+3+3+2）/6=8/3

(b)因为5的全排列有120种情况，写成代码就太多了，所以说一下思路：

首先先从这5个数里选择前3个数，这选出来的3个数按照第一问排好序

其次第4个数与排好的3个数中的第2个比较，再与第1个和第3个比较，共需要2次比较就能到达正确的位置

最后第五个数与排好的4个数中的第2个比较，就会有再与第1个比较或者再与第3个比较的过程，其中最好的情况只需要2次，最差的情况需要比较3次

最好情况下：六次比较

最差情况下：八次比较

平均情况下：8/3+2+12/5=106/3

(c)同理可得

最好情况下：十三次比较

最差情况下：十七次比较

平均情况下：十六次比较

7.17

首先拿出30000bit空间去按顺序标记序列的关键码，之后再使用分配排序，输出时只输出数值。

7.18

1 不排序 平均复杂度n

2 不排序 平均复杂度n

3 不排序 平均复杂度n

4 使用快速排序 平均复杂度nlogn

5 使用快速排序 平均复杂度nlogn+n

7.20

代码：

1. LList<int> mergesort(LList<int> inlist) { LList<int> templist[2];
2. **if** (inlist.length() <= 1)
3. **return** inlist; inlist.setStart();
4. int curr = 0;
6. **while** (!inlist.isEmpty()) {
7. int item;
8. inlist.remove(item);
9. templist[curr].append(item);
10. curr = (curr + 1) % 2;
11. }
13. mergesort(templist[0]);
14. mergesort(templist[1]);
16. templist[0].setFirst();
17. templist[1].setFirst();
18. **while** (!templist[0].isEmpty() ||
19. !templist[1].isEmpty()) {
20. **if** (templist[0].isEmpty()) {
21. templist[1].remove(item);
22. inlist.append(item);
23. }
24. **else** **if** (templist[1].isEmpty()) {
25. templist[0].remove(item);
26. inlist.append(item);
27. }
28. **else** **if** (templist[0].currValue() < templist[1].currValue()) {
29. item = templist[0].remove();
30. inlist.append(item);
31. }
32. **else** {
33. item = templist[1].remove();
34. inlist.append(item);
35. }
36. }
37. **return** inlist;
38. }

7.22

任何基于比较的搜索算法都可以类比为判定树。这棵树会有n个叶节点，就至少有logn层。所以基于比较的搜索算法在最差的情况下为logn