**Tutorial of Quick Sort and Heap Sort**

**Q1**. Given the pseudo code of Quick Sort in lecture note 07, for the array ***A***= {3, 8, 4, 0, 11, 1, 9, 12, 7, 6, 13, 2, 5, 14, 10}.

(1) if we call ***MEDIAN3*** function, what the array will be like?

(2) In the 1st iteration, after we call ***PARTITION*** function (Lec 07, page 26), what the array will be like, and what’s the value of ***i***and ***j***?

**Solution:**

**(1)** The array will be {3, 8, 4, 0, 11, 1, 9, 14, 7, 6, 13, 2, 5, 10, 12}, and the ***pivot*** = 10. Note that ***pivot*** here is value, not index.

**(2)** The array will be{3, 8, 4, 0, 5, 1, 9, 2, 7, 6, 10, 14, 11, 13, 12}, i=10, j=9. Note that ***i*** and ***j*** here are indexes, not values.

**Q2**. If we use Quick Sort algorithm to sort an array, ***A***, with elements are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (These array elements may be arranged in an unordered manner).

(1) Try to analyze the algorithm complexity of the best case for ***A***?

(2) Try to analyze the algorithm complexity of the worst case for ***A***?

(3) Analyze the algorithm complexity of average case.

**Solution:**

As shown in our Lecture notes (Lec 07, page 28), for Quick Sort Algorithm, we need:

* Divide
  1. Pivot selection: O(1)

For pivot selection, we use ***MEDIAN3*** method to realize that, the cost is O(1).

* 1. Partitioning: O(n)

In the PARTITION process, we need to compare each element with the pivot and exchange the position between A[i] and A[j], so the cost is O(n)

* 1. Recursive calls: T(q) + T(n-q-1)

For the recursive calls “*QUICKSORT(A, left, q-1)*” and “*QUICKSORT(A, q+1, right)*”, the cost is T(q) + T(n-q-1).

* Conquer and Combine: O(1)

All in all, the recurrence relation of quick sort algorithm complexity is T(n)=T(q)+T(n-q-1)+O(n)

(1) The best case should be that in each iteration, the pivot chosen is exactly the median of the sub-array, thus each partition can reduce the number of elements that need to be sorted as much as possible. For example, we have found 3 as pivot for the sub-array of ***A***, {0,4, 5, 3, 2, 1, 6}

In this case, the original array ***A*** (with size ***n***) can be evenly divided into two sub-arrays ***A1*** and ***A2*** of the same length ***n/2***. Thus, we have:

T(n) = 2T(n/2) + n

= 2[2T(n/22) + n/2] + n

= 22T(n/22) + 2n

= 23T(n/23) + 3n

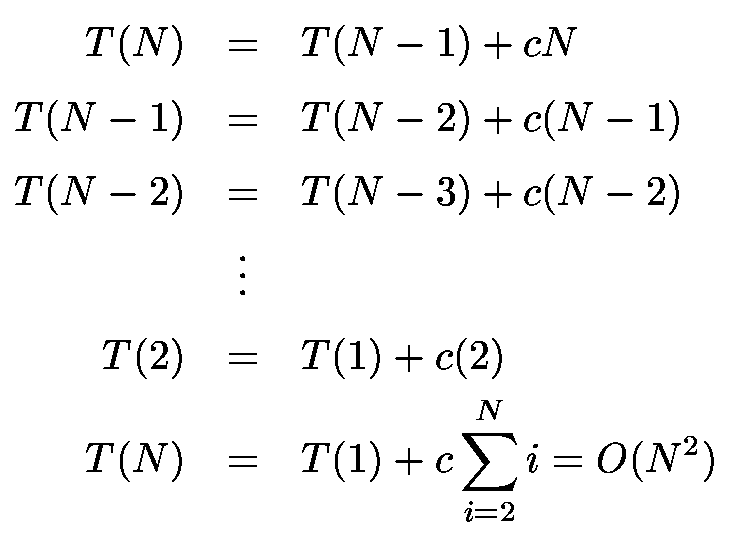
= 2iT(n/2i) + i\*n

Let i=log(n),

= nT(n/n) + n\*log(n)

= O(n\*log(n))

(2) The worst case should be that in each iteration, the pivot we chose are always the biggest or smallest element of the array. In this case the length of sub-array, ***A1***, will be ***0*** and the length of another sub-array will be ***n-1*** (minus 1 because the pivot is also one element of the array). Thus, we have:



**(3)** For the average case, it should be the average case of which pivot can be placed any position of the array, then the recurrence relation will be: . Solve it, and we can get the algorithm complexity is O(nlogn).

**Q3**. If we use the pseudo code of ***INSERTION*** function to construct a max heap for the array ***A***= {3, 8, 4, 0, 11, 1, 9, 12, 7}.

(1) Show each step of ***INSERTION*** (especially for the changes of the heap)

(2) Based on the final step of (1), if we use post-order traversal to traverse the heap, what will be the result?

(3) Show each step of ***DELETEMAX*** (including the changes of heap and the array ***A***)

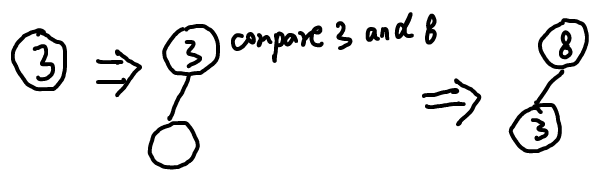
**Solution:**

(1)

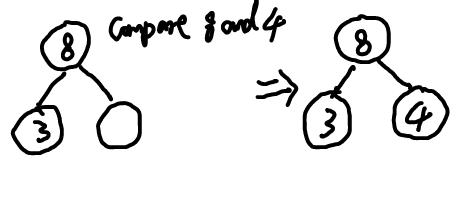
Insert 3:



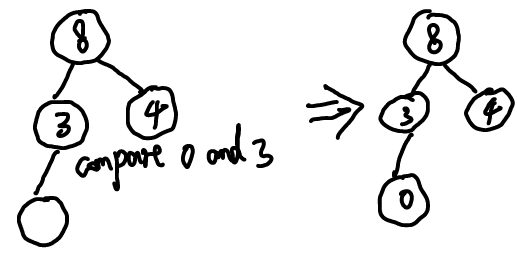
Insert 8:



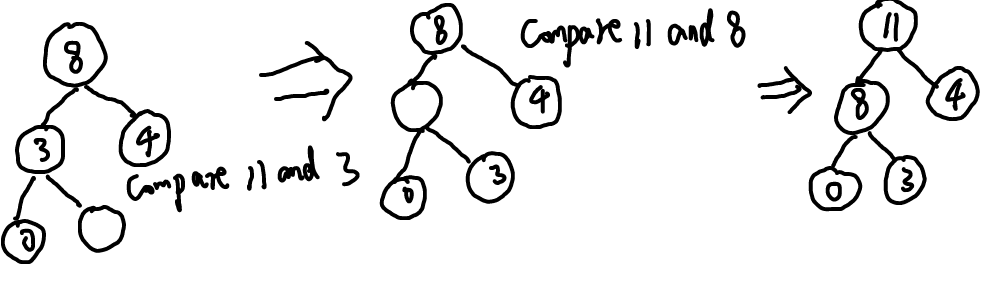
Insert 4:



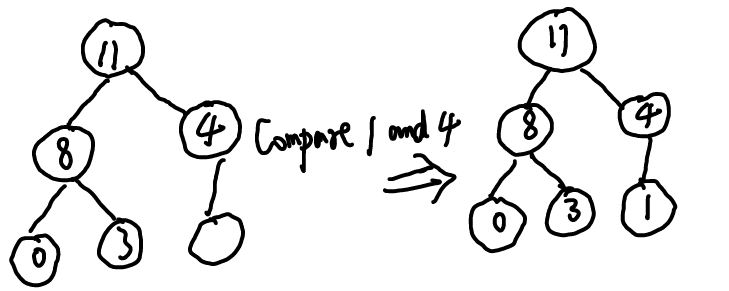
Insert 0:



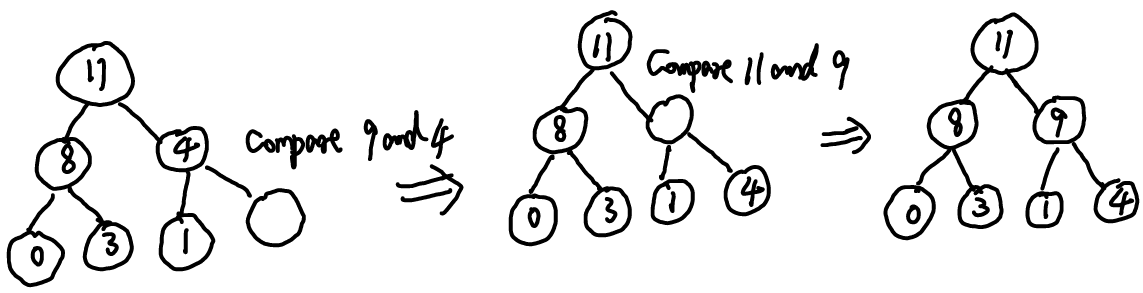
Insert 11:



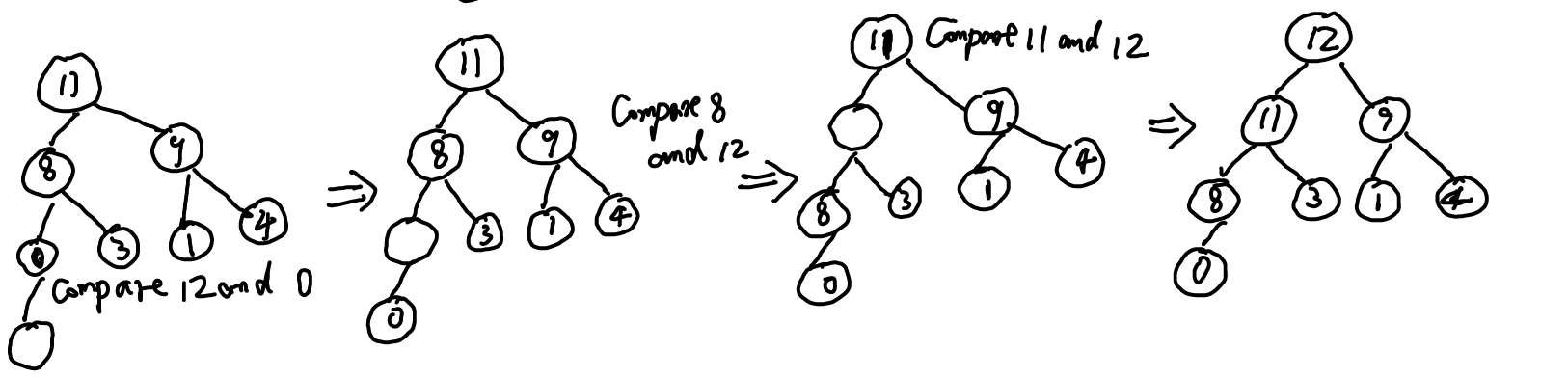
Insert 1:



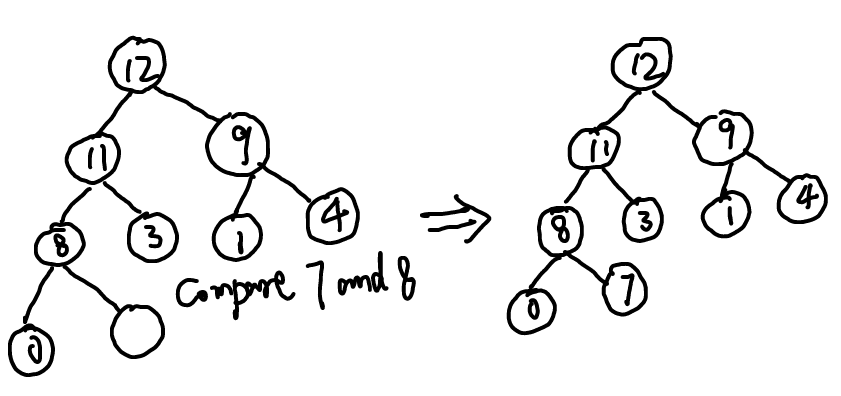
Insert 9:



Insert 12:



Insert 7:



(2) For the last step of the binary-max-heap, we use post-order traversal, we can get:

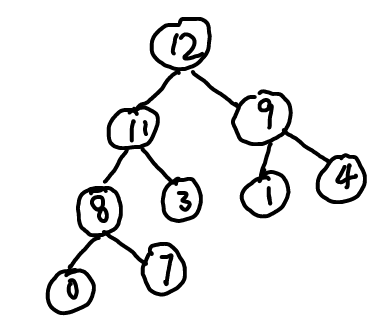
0,7,8,3,11,1,4,9,12

(3) We execute ***DELETEMAX*** step by step

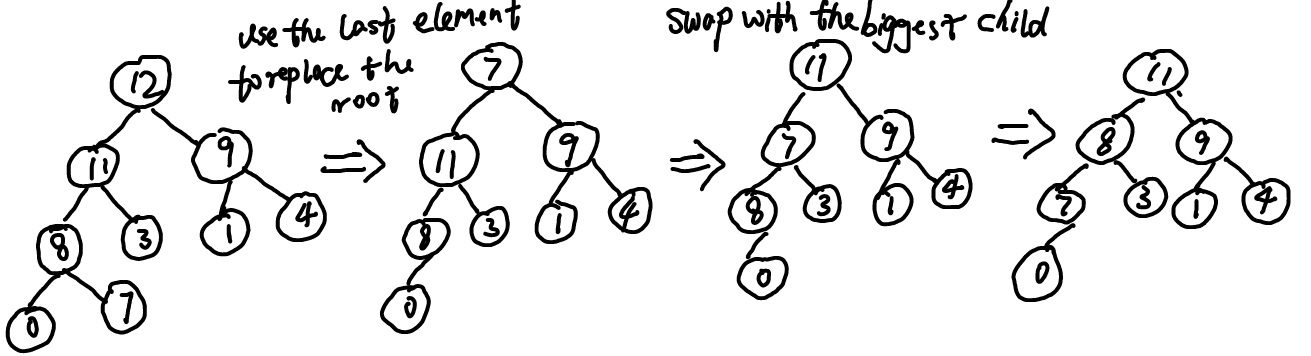
Before deletion, the content of array ***A*** is:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 8 | 4 | 0 | 11 | 1 | 9 | 12 | 7 |

And the heap is:



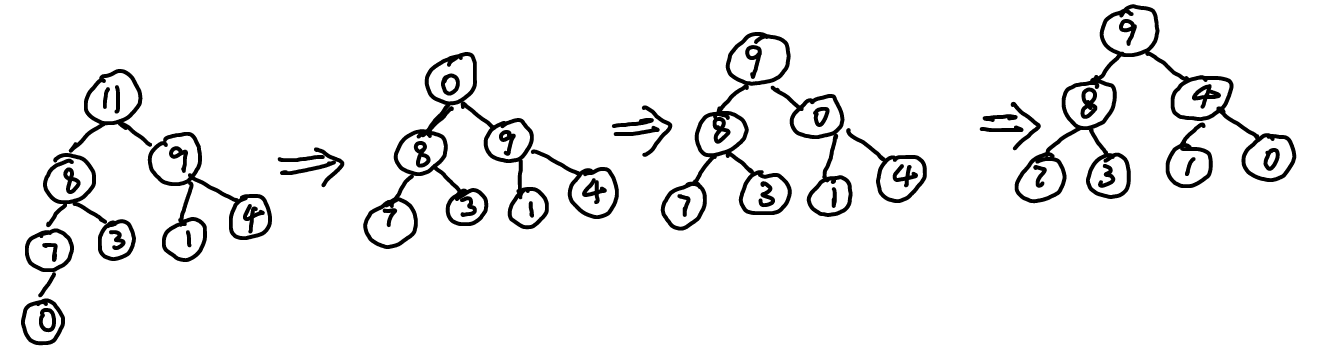
(a) the 1st step of deletion:



And the array is:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 8 | 4 | 0 | 11 | 1 | 9 | 12 | **12** |

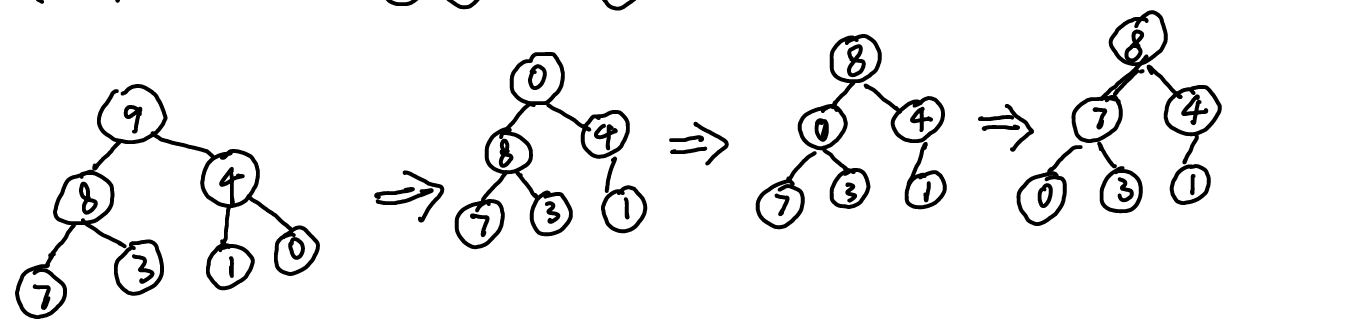
The 2nd step of deletion:



And the array is:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 8 | 4 | 0 | 11 | 1 | 9 | **11** | **12** |

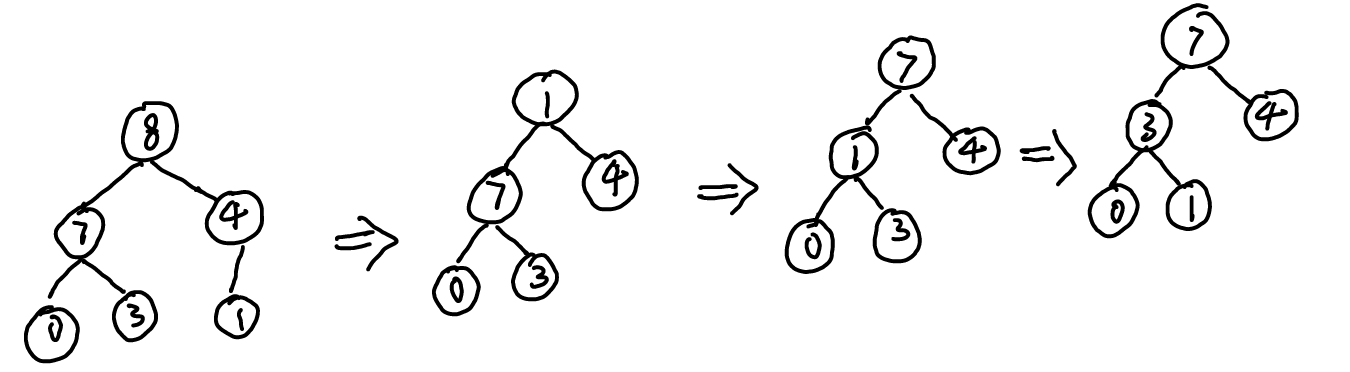
The 3rd step of deletion:



And the array is:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 8 | 4 | 0 | 11 | 1 | **9** | **11** | **12** |

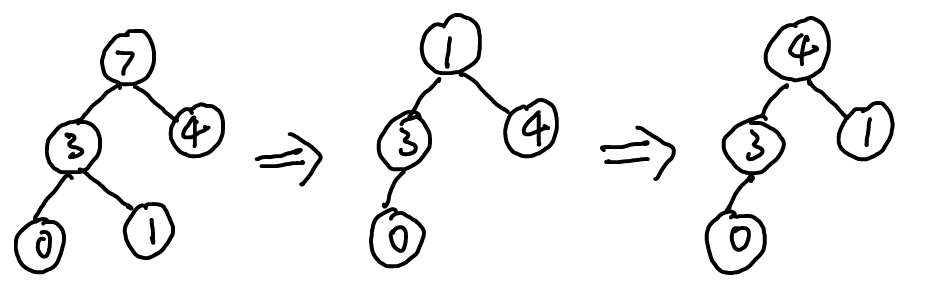
The 4th step of deletion:



And the array is:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 8 | 4 | 0 | 11 | **8** | **9** | **11** | **12** |

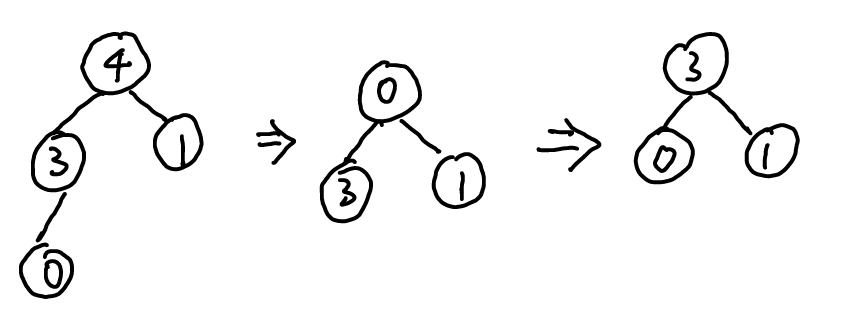
The 5th step of deletion:



And the array is:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 8 | 4 | 0 | **7** | **8** | **9** | **11** | **12** |

The 6th step of deletion:



And the array is:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 8 | 4 | **4** | **7** | **8** | **9** | **11** | **12** |

Please try the last 3 steps of deletion by yourself, and the changes of the array will be as follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 8 | **3** | **4** | **7** | **8** | **9** | **11** | **12** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | **1** | **3** | **4** | **7** | **8** | **9** | **11** | **12** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **3** | **4** | **7** | **8** | **9** | **11** | **12** |

Done.