CS2010 Semester 1 2012/2013

Data Structures and Algorithms II

Tutorial 03 - Heaps

For Week 05 (10 September - 14 September 2012)

Released: Thursday, September 5, 2012

Document is last modified on: September 12, 2012

1 Introduction and Objective

The purpose of this tutorial is to reinforce the concepts of Binary Heap data structure which can be used as Priority Queue. We will also discuss PS2 Subtask 1 during this tutorial.

You can use `http://www.comp.nus.edu.sg/~stevenha/visualization/heap.html` to *verify* the answers of some questions in this tutorial. However during written tests, you have to be able to do this by yourself though.

2 Tutorial 03 Questions

Heaps, Heaps and more Heaps !

Q1. Is the tree shown in Figure 1 below a valid max heap?

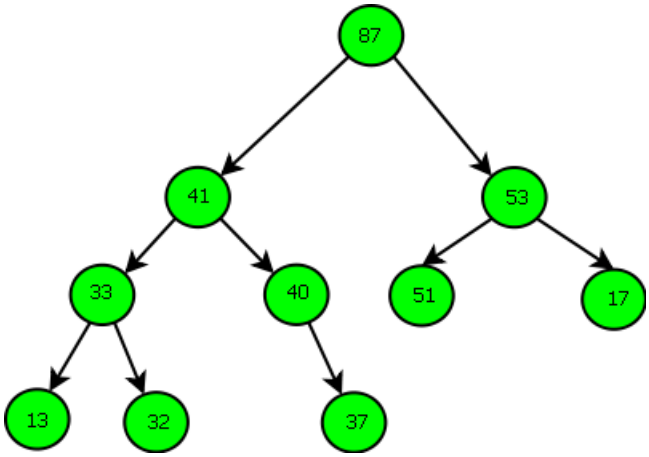


Figure 1: Is this a heap?

Ans: **No**. Because it's not a complete binary tree. If node 37 is the left child rather than the right child of 40, then it is a valid max heap.

Q2a. Show the result of inserting 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, 2 one by one into an initially empty max heap (in another word, execute `BuildHeapSlow(array)` as shown in the lecture).

Ans: See Figure 2.

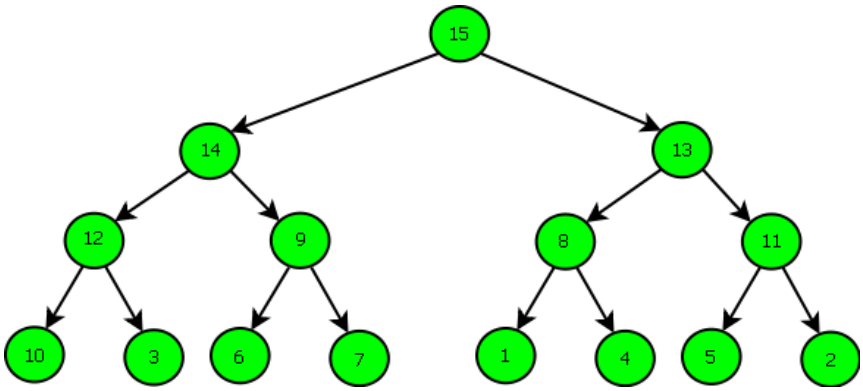


Figure 2:

Q2b.) Show the result of using $O(n)$ `BuildHeap(array)` to build a max heap using the same input.

Ans: See Figure 3. Notice that the two heaps are different for this question, but both are valid. Check your answer with: `www.comp.nus.edu.sg/ stevenha/visualization/heap.html`

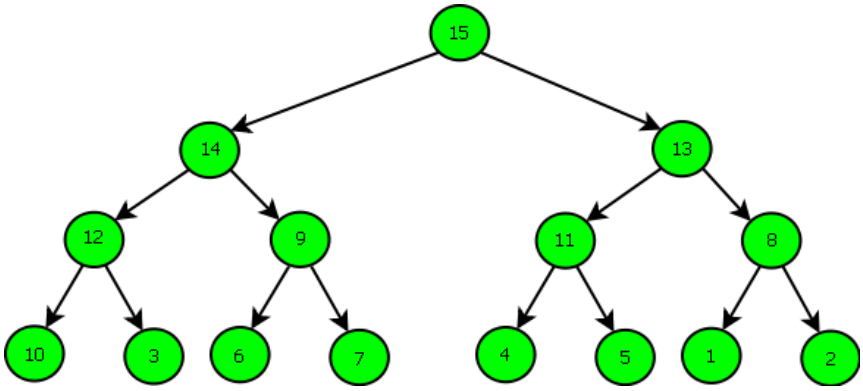
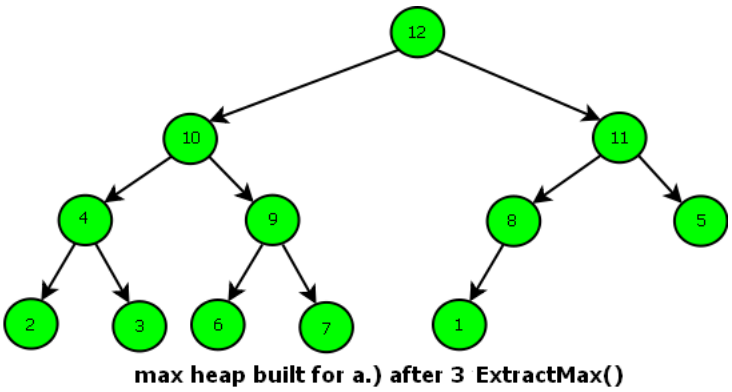


Figure 3:

Q2c.) Show the result of 3 `ExtractMax()` operations on the max heap built in a.) and the one in b.)



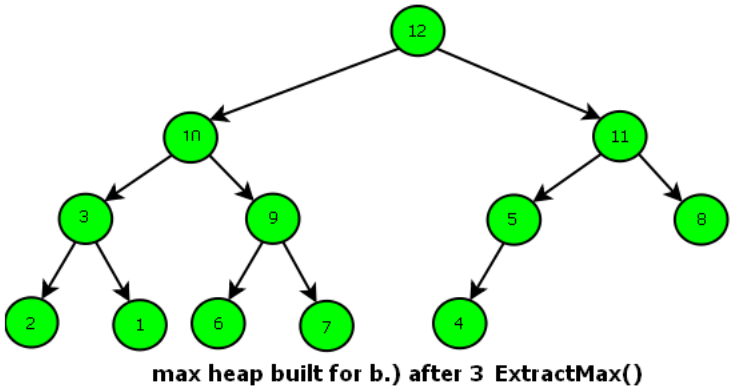


Figure 4:

Q3. What is the minimum and maximum number of comparisons between heap elements required to construct a max heap of 8 elements using the $O(n)$ `BuildHeap(array)`?

Ans: 7 and 11 respectively.

The case for minimum happens when the array to be converted into a max heap already satisfies the max heap property (e.g. try {8,7,6,5,4,3,2,1}). The structure of an 8 node heap is shown below.

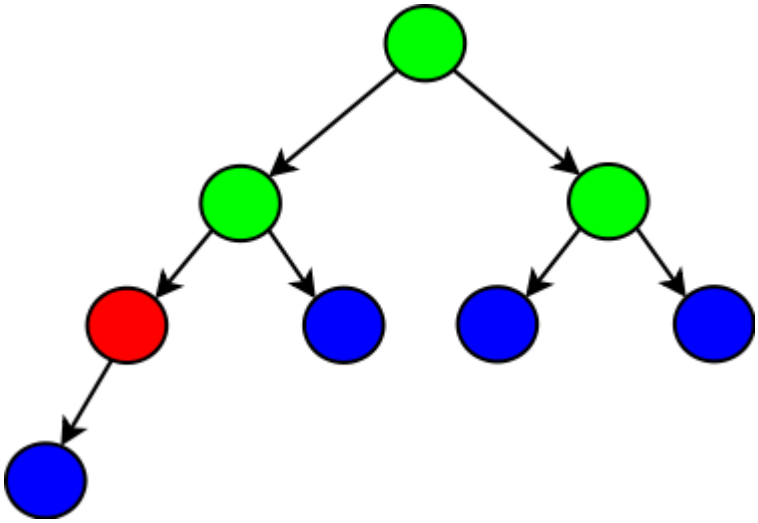


Figure 5:

In this case except for the last internal node (the red node), which only does 1 comparison, the rest of the internal nodes (green nodes) do exactly 2 comparisons (with it's left and right child) so the # of comparisons = 1+3*2 = 7.

The case for maximum happens where we have to call `ShiftDown` at each internal node all the way to the deepest leaf (note: {1,2,3,4,5,6,7,8} does not really produce the maximum number of comparison, try {1,2,3,5,4,6,7,8}).

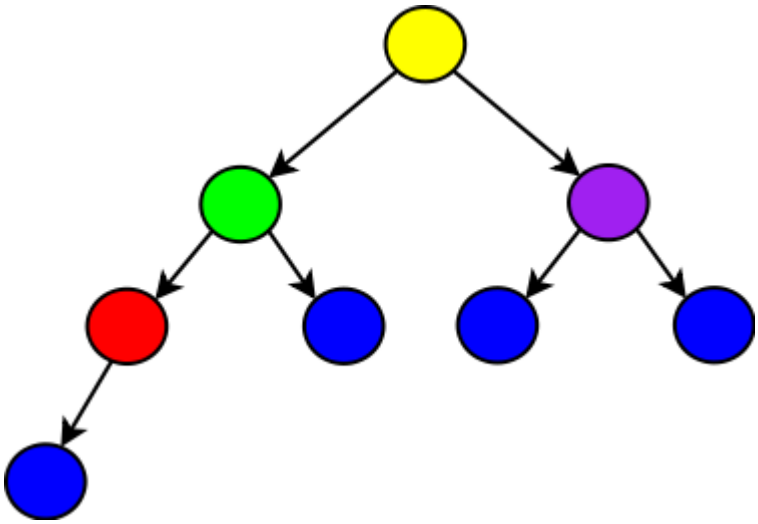


Figure 6:

The red node requires 1 comparison

Then, the purple node requires 2 comparison
The green node requires 2+1 (2 comparison at level 1, 1 comparison one level down at level 2)
Finally, the yellow node requires 2+2+1 (trace the longest path)
Total = 1+2+(2+1)+(2+2+1) = 11.

Q4. What modifications are required so that *both* `ExtractMax()` and `ExtractMin()` can be done in $O(\log n)$ time and every other heap operation retains the same running time?

Classic Answer: Instead of 1 heap, we use 2 heaps (this is called a double ended priority queue). One is a max heap, the other a min heap. Each node has an additional pointer (call it the twin pointer) which points to its twin node from the other heap.

Updating twin pointers: The twin pointer of a node will need to be updated whenever it's twin is shifted up or down the heap. The ShiftUp and ShiftDown method will does have to be modified to take care of the twin pointers. Updating the twin pointer takes a constant time per ShiftUp/ShiftDown, and thus does not affect the complexity of ShiftUp and ShiftDown methods, and by extension every other methods used in heaps.

Initialization: Simply insert the list of items into the 2 heaps, initialize the pointers (in array based implementation of heaps, the twin pointer will simply point to the node in the same index for each heap), and perform BuildHeap on each heap. Time complexity is still $O(n)$.

ExtractMin or ExtractMax: Go to the corresponding heap (min heap if ExtractMin and max heap if ExtractMax) and perform the operation. Now we follow the twin pointer to the other heap. Note that in the other heap, the position of the node that is replaced by the last leaf must itself be a leaf (otherwise the min or max heap property is violated). Now we only need to perform ShiftUp, if necessary, on the replaced node. Time taken is bounded by $O(\log n)$ since ShiftDown and ShiftUp are called once each.

Insert: Insert the new item into both heaps. Make sure twin pointer is set. Perform ShiftUp on each heap. Time needed is still $O(\log n)$, as ShiftUp is only called twice, once on each heap.

Competitive Programmer Answer: Just use a balanced BST to model the Priority Queue. The keys in this bBST are the priority values. Then, to find the minimum/maximum element, we just need to call $O(\log n)$ `findMin()`/`findMax()`. Insertion/Enqueue to bBST is still $O(\log n)$. Notice that we do NOT have to associate `PriorityQueue` ADT to be always a `Binary Heap` data structure!

Q5. Give an algorithm to find all nodes bigger than some value x in a max heap that runs in $O(k)$ time where k is the number of nodes in the output.

Ans: Perform a pre-order traversal of the max heap starting from the root. At each node, check if node key is > x. If yes, output node and continue traversal. Otherwise terminate traversal on subtree rooted at current node, return to parent and continue traversal.

Algorithm 1 findNodesBiggerThanX(node,x)

```
if (node.key > x) then
    output(node.key)
    findNodesBiggerThanX(node.left,x)
    findNodesBiggerThanX(node.right,x)
end if
```

Analysis of time bound required -
The traversal terminates when it encounters that a node's key $\leq x$. In the worst case, it encounters $k * 2$ number of such nodes. That is, each of the left and right child of a valid node (node with key $> x$) are invalid. It will not process any invalid node other than those $k * 2$ nodes. It will process all k valid nodes, since there cannot be any valid nodes in the subtrees not traversed (due to the heap property). Thus the traversal encounters $O(k + 2 * k) = O(k)$ number of nodes in order to output the k valid nodes.

Q6. The *second* largest element in a max heap with more than two elements (all elements are unique) is always one of the children of the root. Is this true? If yes, show a simple proof. Otherwise, show a counter example.

Yes it is true. This can be proven easily by proof of contradiction. Suppose the second largest element is not one of the children of the root. Then, it has a parent that is not the root :O. There will be a violation to max heap property. Contradiction. So, the second element must always be one of the children of the root.

Problem Set 2

Q7. Discussion of PS2 subtask 1.

Ans: Just by using CS1020 knowledge: You can create an array of 100-30+1 = 71 normal queues of Strings to indicate `dilation` 30 to 100. Put `womanName` at the back of the correct queue (at index: `priority - 30`) when she arrives. This is $O(1)$. `GiveBirth()` always remove the woman at the front of the highest numbered queue that is currently not empty and similarly for `Query()`, both are $O(71) = O(1)$. This should be doable with just a few array of queues manipulation.

This trick only works if the priority values only involve small range of integers. It also does not work with `UpdateDilation` due to the need for searching the location of the current priority value of a certain `womanName` that can be anywhere in the 71 normal queues (can be in the middle of one of those normal queues).