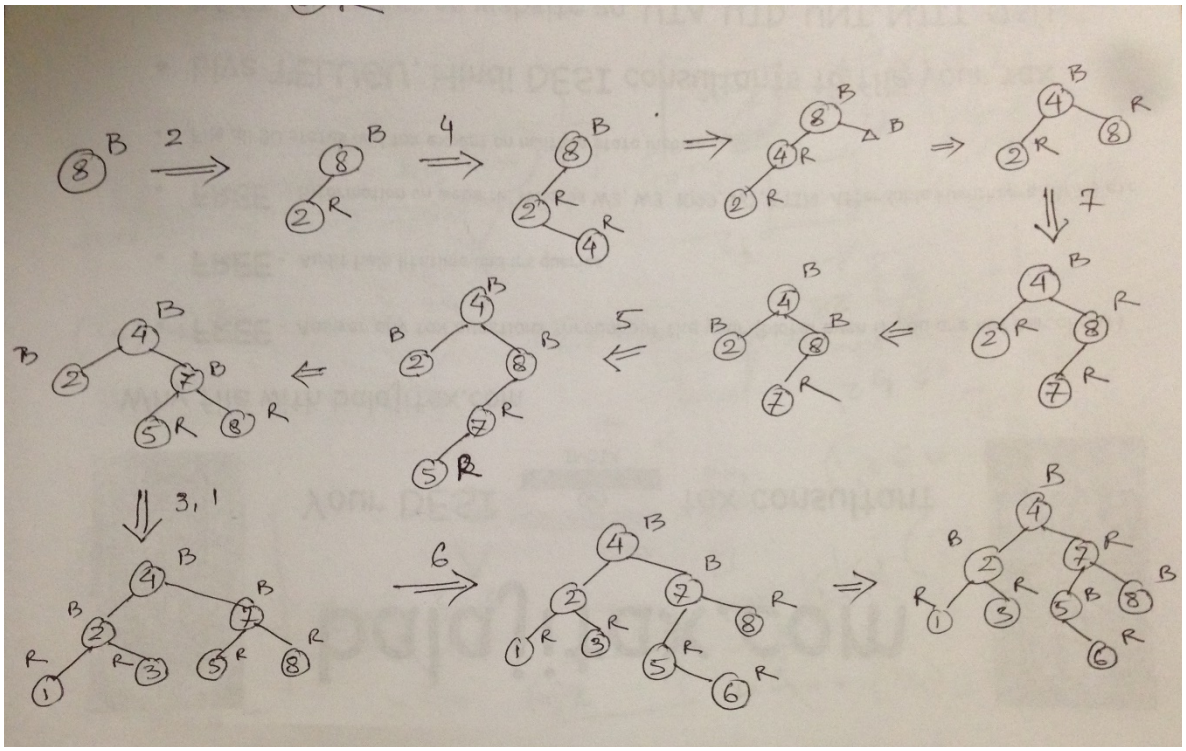
1. Given a heap and a number k, design an efficient algorithm that outputs the top-k largest element from the heap. What is the running time of the algorithm? Can you design an algorithm that runs faster than O(klogn)?

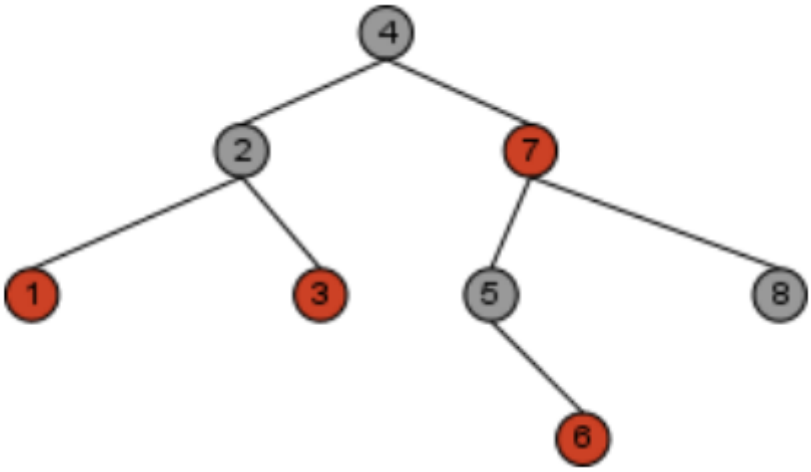
Answer

We can apply Delete Max operation k times in a Max_Heap to get the top-k-largest elements. It will take O(klogn) time. However, using a second heap, we can do this in a faster way. Here goes the description.

Copy the root of the given Max_Heap [let's call it X] and insert it into a new Max_Heap [let's call it Y, this is empty at the beginning]. Apply Delete Max in Y. This will give the first max element.
Copy the first max element's children from X and insert into Y. Apply Delete Max in Y. This will give the second max element.
Copy the second max element's children from X and insert into Y. Apply Delete Max in Y. This will give the third max element.
Do this K times.
Each time, we are inserting two elements from X into Y and deleting one element from Y. So, the maximum number of elements in Y is (k+1). So, the height of Y will be $\log_2 k$. As, we are doing 2k number of insertion and k number of deletion, the order should be $O(k\log_2 k)$.
[2*k + 1 – k = (k + 1)]

2. Insert the following numbers into an initially empty red-black tree: 8 2 4 7 5 3 1 6. (To solve this problem correctly, you will need to refer to the text-book for details about red-black trees in addition to the main ideas discussed in class).





4. Instead of binary heaps, suppose you had to implement ternary heaps (i.e., where each node has up to three children). Explain how you would implement such heaps using arrays, and how you can determine child and parent pointers. What are the advantages/disadvantages of ternary heaps over binary heaps?

The children of element A[i] are, A[3*i], A[3*i+1] and A[3*i+2]. The parent of element A[i] is, A[floor(i/3)]. Advantage of a ternary heap is, it will have $\log_3 n$ height. It is $\log_3 2$ order smaller than the binary heap height, $\log_2 n$. Disadvantage is, in *heapify/delete_min* operation, it will take 2 comparisons to get the minimum of 3 children while swapping for maintaining heap property.

5. Can you use a binary search tree to simulate heap operations? What are the advantages/disadvantages of doing so?

Insert: BST has its own insert function like Heap. But the disadvantage is, in the worst case, it may take O(n) time unlike $O(\log_2 n)$ in Heap.

Find min: In Heap, we can find the min in O(1) time. In BST, the leftmost node contains the minimum number. So, starting from the root, we will need O(h) time to find the min where in the worst case h = n and in the best case h = $\log_2 n$. However, there is a better way to find the min in BST. We can keep an extra pointer which will always point to the leftmost node of BST. This pointer will be updated when an insert or delete is made. By this way, we can simulate find min in O(1) time.

Delete min: In Heap, Delete min takes $O(\log_2 n)$ time in the worst case. Unlike that, in BST, it will take O(1) time to find the min and O(h) time[in worst case, h = n; best case, h = $\log_2 n$] to change the structure of the tree.

6. Design an algorithm to convert a given binary search tree into a heap efficiently. What is the running time of your algorithm?

BST is implemented using array data structure. We can apply Build_Heap on that array and get the Heap in O(n) time. If it was not implemented using array, we can traverse the BST in O(n) time and get it into an array.

7. Assume a heap is arranged such that the largest element is on the top. Suppose you are given two heaps S and T, each with m and n elements respectively. What is the running time of an efficient algorithm to find the $10th$ largest element of S U T?

Solved in Class.

8. What is minimum possible number of nodes in a red-black tree which contains two black nodes from every root-to-leaf path?

3

9. Suppose we define a red-black tree where along each path the number of black nodes is the same, and there cannot be more than two consecutive red nodes.

I. What is the ratio of the longest possible path length to the shortest possible path length in this tree? **3**

II. For a tree that has b black nodes along each path, what ratio of the maximum possible number of nodes to the minimum possible number of nodes in the tree? **9 or 2^{3b}-1/2^{b}-1**

10. Suppose you implement a heap (with the largest element on top) in an array. Consider the different arrays below; determine which of these arrays that cannot possibly be a heap:

I. 7 6 5 4 3 2 1, ***Yes***

II. 7 3 6 2 1 4 5, ***Yes***

III. 7 6 4 3 5 2 1, ***Yes***

IV. 7 3 6 4 2 5 1, ***No***

V. 7 4 6 3 2 1 5, ***Yes***

11. Can you use any of the previously studied data structures (e.g. heaps, red-black trees) for the Union-Find problem? Explain your answer.

12. Suppose we are working with a Union-Find data structure as described in class (this time path compression is allowed for Find operations). Node z is at a distance of 4 from the root r of its tree (i.e.4 edges away). We now perform a Find on z.

a) By how much will the degree of r change? Explain your answer. (3)

b) By how much will the degree of z change? Explain your answer. (0)

13. What is the worst-case performance of the Union-Find data structure we discussed (with union by rank and path-compression)?

14. Suppose you are given a set S = {1, 2, 3,……9}. Show the steps of Union-Finding by union by size and union by rank (no path compression) for the sequence of following operations: Union (1,2), Union (3,4), Union (2, 4), Union(5,6), Union (6, 7), Union (7, 8), Union (8, 9), union (4, 9).

There are multiple solutions in this problem. One of the solutions is: