Discussion 4

CSCI 570

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu

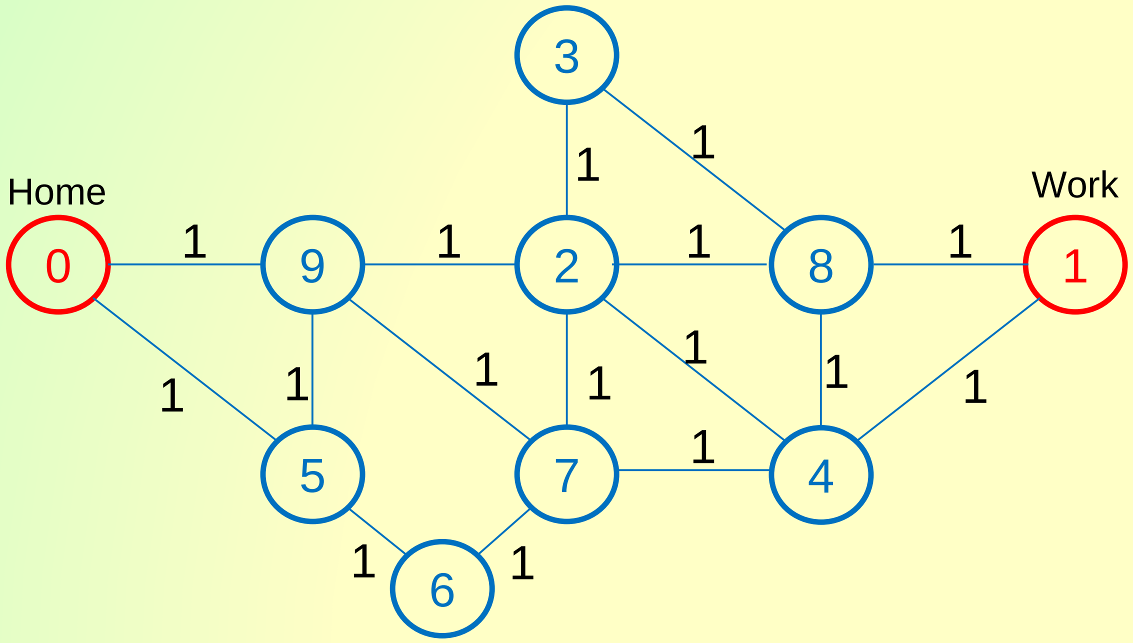
DISCUSSION 4

Outline

- Problems with Solutions

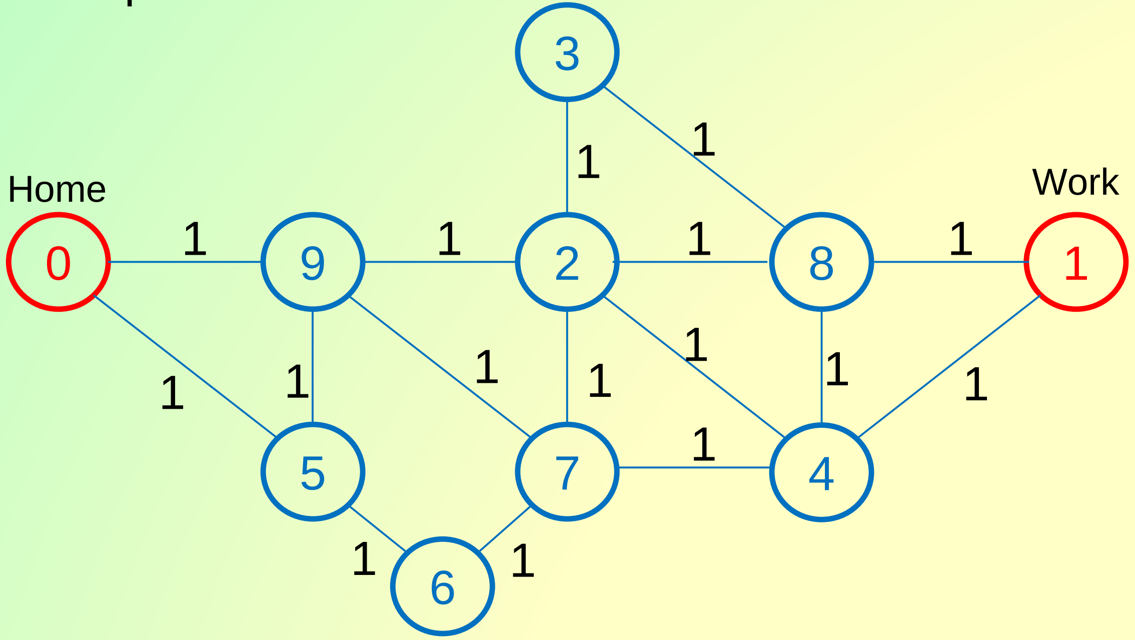
Problem 1

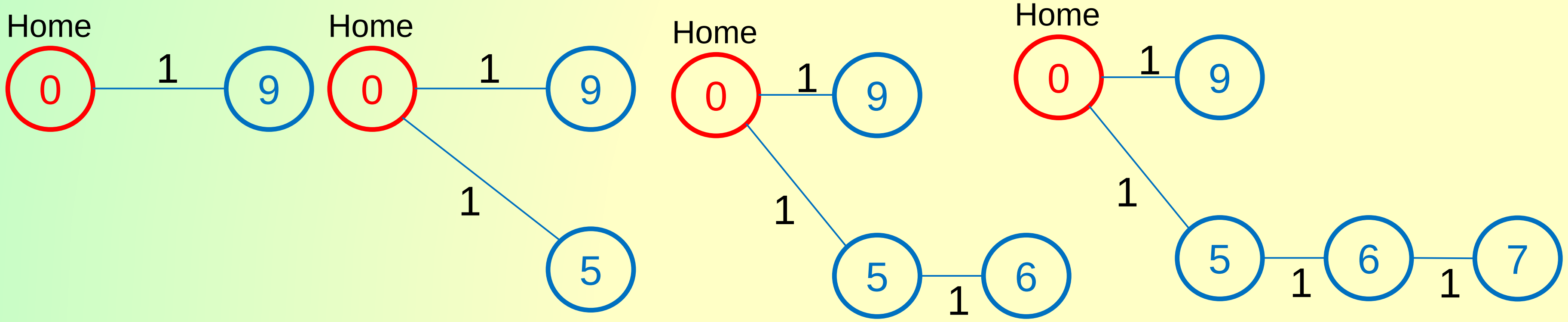
- Hardy decides to start running to work in San Francisco city to get in shape. He prefers a route that goes entirely uphill and then entirely downhill so that he could work up a sweat uphill and get a nice, cool breeze at the end of his run as he runs faster downhill. He starts running from his home and ends at his workplace. To guide his run, he prints out a detailed map of the roads between home and work with k intersections and m road segments (any existing road between two intersections). Each road segment has unit length, and each intersection has a distinct elevation. Assuming that every road segment is either fully uphill or fully downhill, give an efficient algorithm to find the shortest path (route) that meets Hardy's specifications. If no such path meets Hardy's specifications, your algorithm should determine this. Justify your answer.



Solution 1

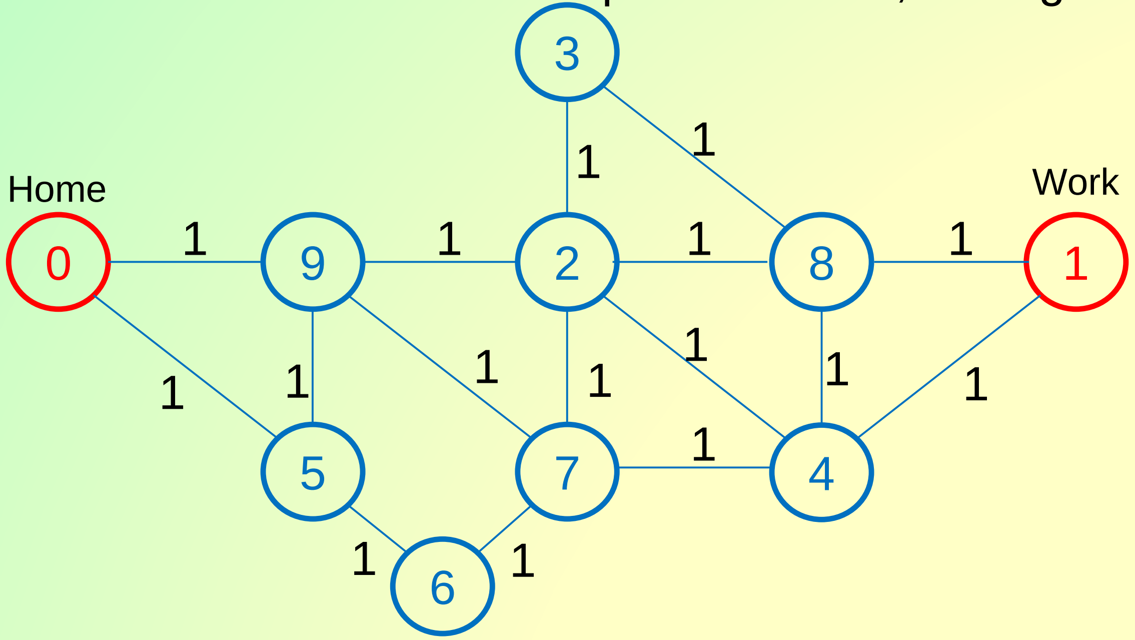
- Let's run Dijkstra's Algorithm from Home only on uphill edges since Hardy only wants to run uphill at first from his home

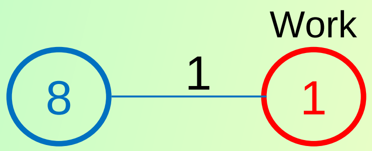


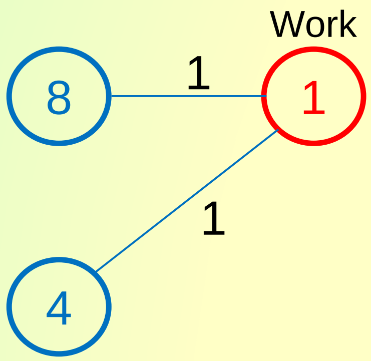


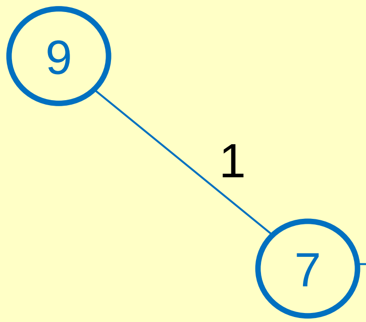
Solution 1

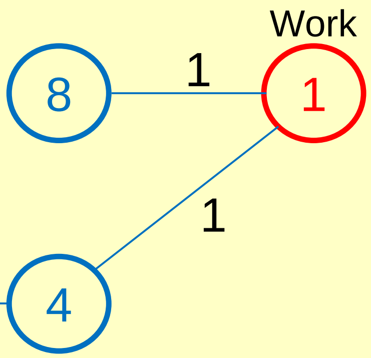
- Now let’s run Dijkstra’s Algorithm from Work only on uphill edges since Hardy only wants to run downhill for the second part of his run, ending at his Work





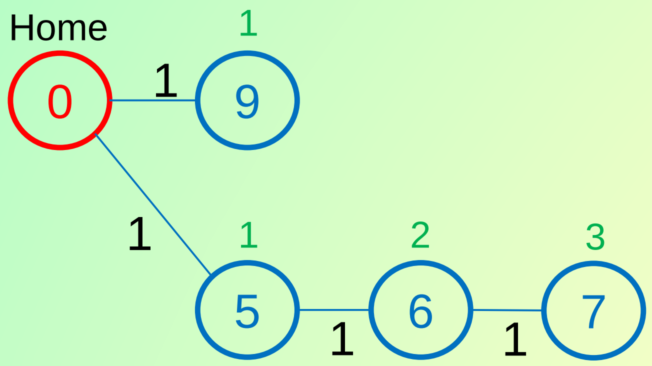


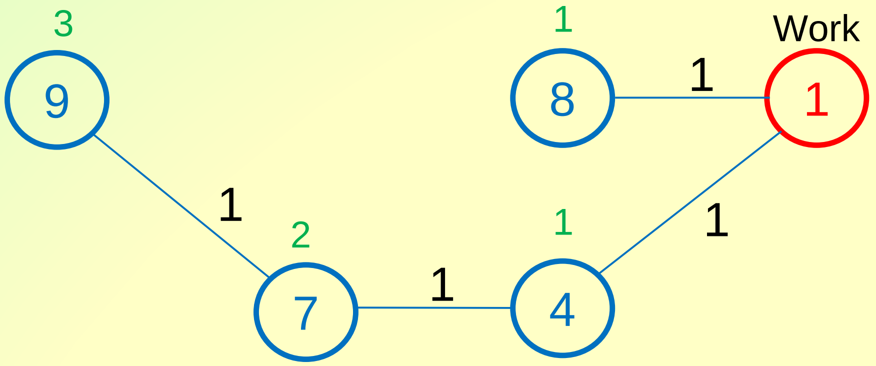




Solution 1

- Find the distances to each node in the shortest path trees that were generated

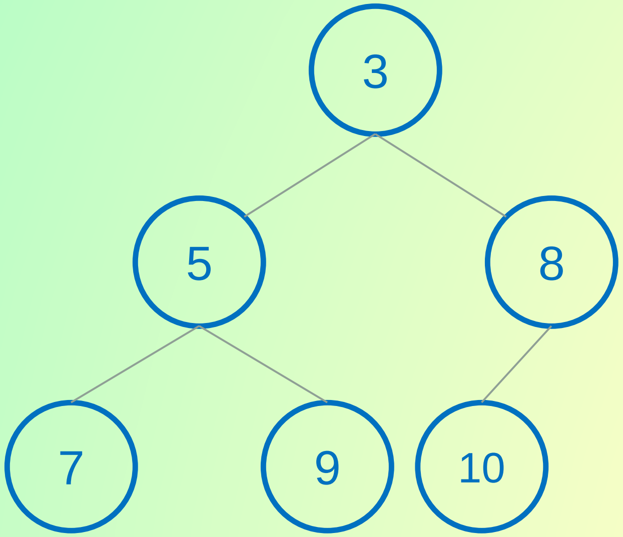


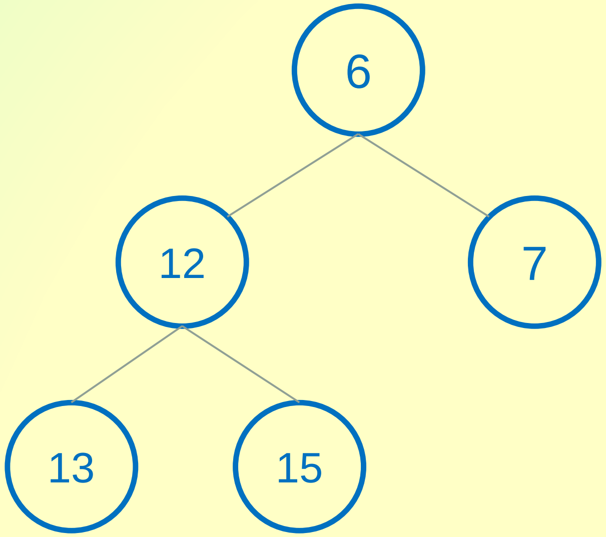


- Find the nodes in common between the two trees, and add the distances together
  - If there are no nodes in common, there is no path that goes completely uphill from Home and completely downhill to Work in the graph
  - If there are nodes in common, the node with the minimum cost will be the node that divides Hardy's uphill route from Home from this downhill route to Work
    - Node 7 is in common, with a cost of 3 from Home and 2 from Work, giving a total cost of 5
    - Node 9 is in common, with a cost of 1 from Home and 3 for Work, giving a total cost of 4, which is the shortest path that is completely uphill from Home and completely downhill to Work with 9 being the apex

Problem 2

- Suppose you have two binary min-heaps, A and B, with a total of *n* elements between them. You want to discover if A and B have a key in common. Give a solution to this problem that takes time O(n lgn) and explain why it is correct. Give a brief explanation for why your algorithm has the required running time.
  - Here are two sample min-heaps.

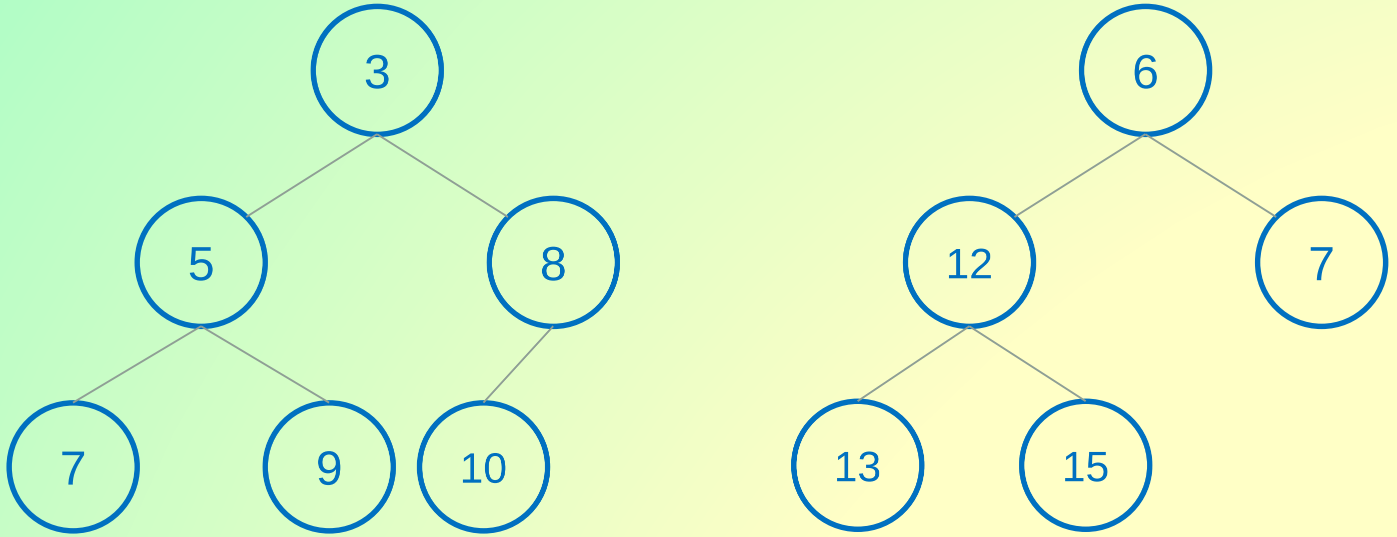




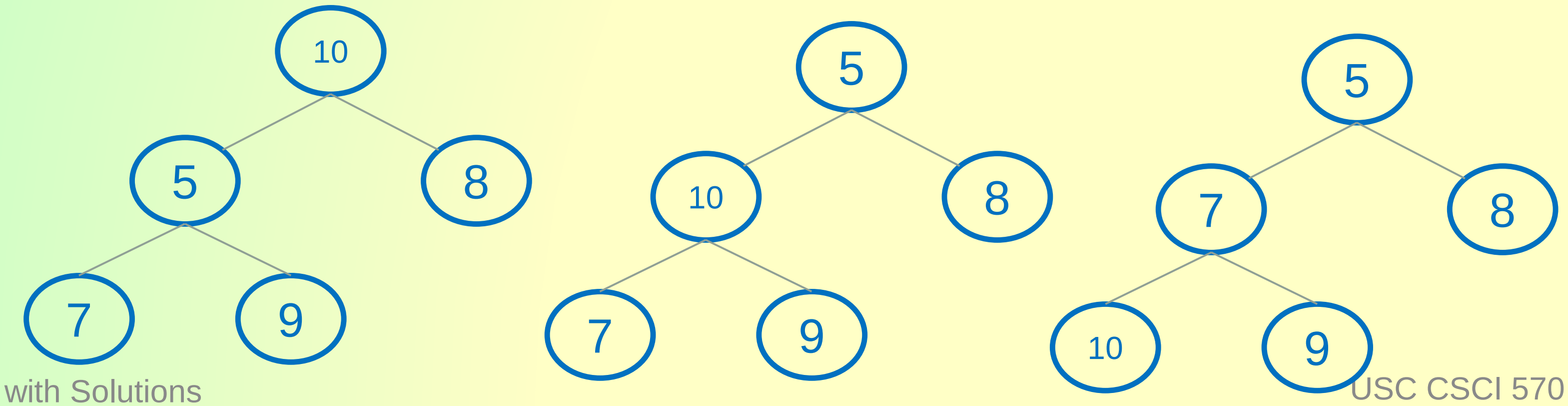
Solution 2

- Compare the root of each min-heap. If they match, stop.
- If they don't match, remove the smaller-valued root
  - We remove the smaller-valued root because we know that value isn't in the other min-heap. There is a chance the larger-valued root is in the other min-heap. Why?
  - When you delete the root, move one of the leaves to the root and call MIN_HEAPIFY
- Repeat until a match is found or one of the min-heaps is empty
  - MIN_HEAPIFY runs in O(lgn), and in the worst case, we have to run MIN_HEAPIFY on all n nodes, giving O(nlgn)

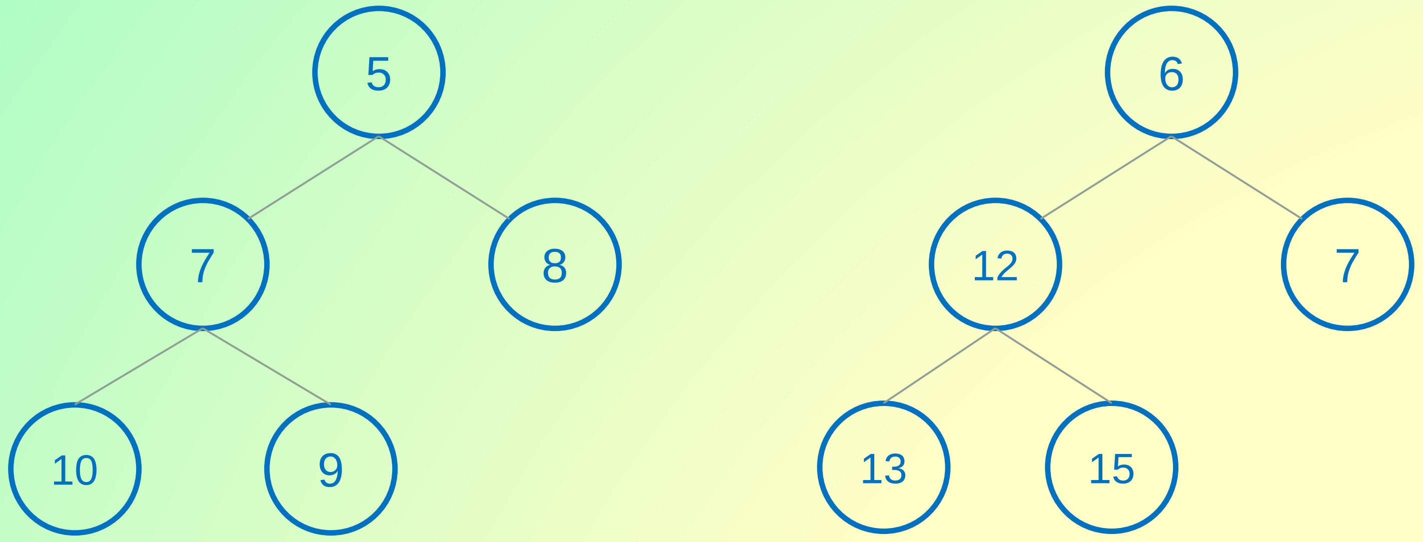
Solution 2



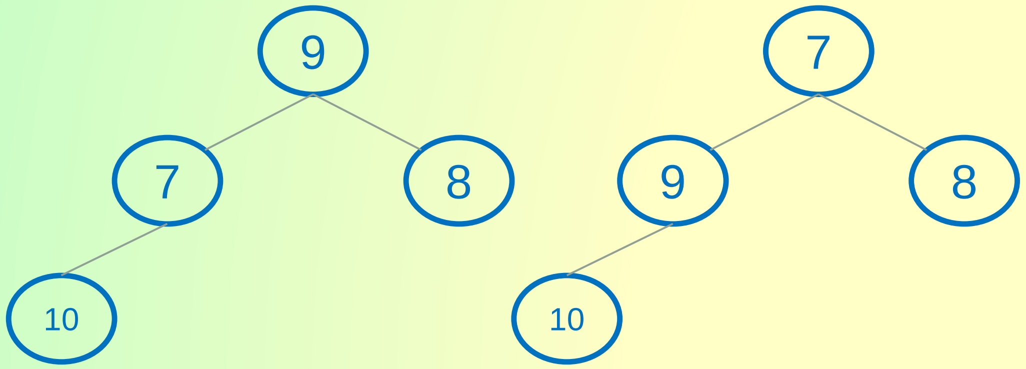
- Compare 3 and 6
  - Remove 3, replace it with 10, and 10 floats down



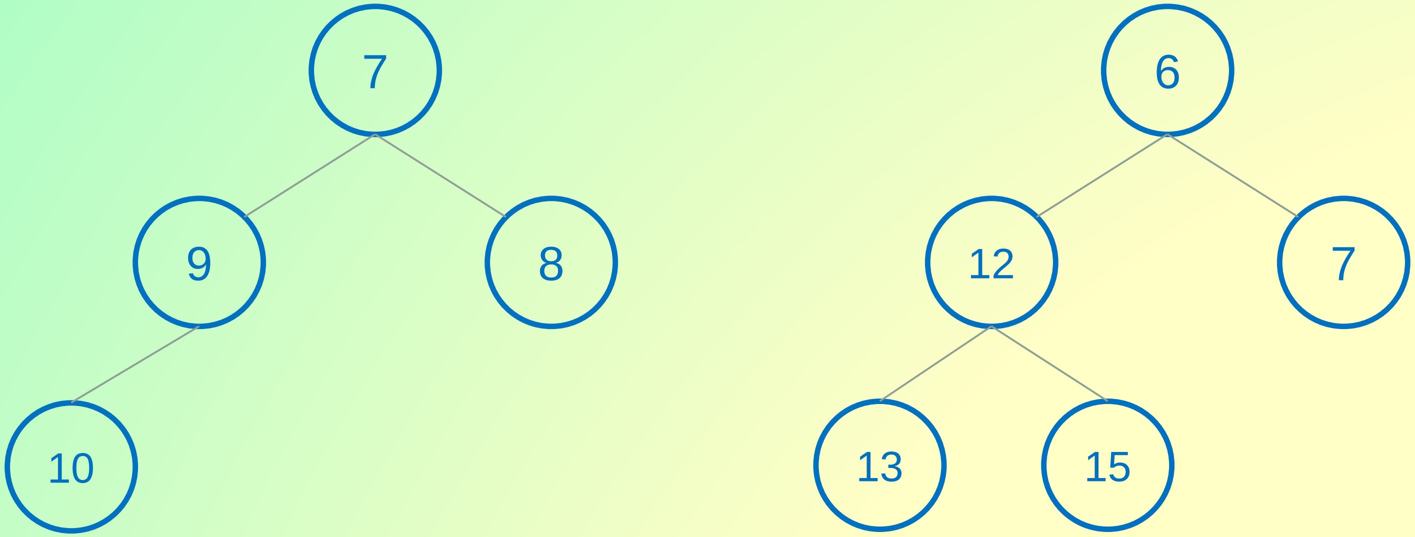
Solution 2



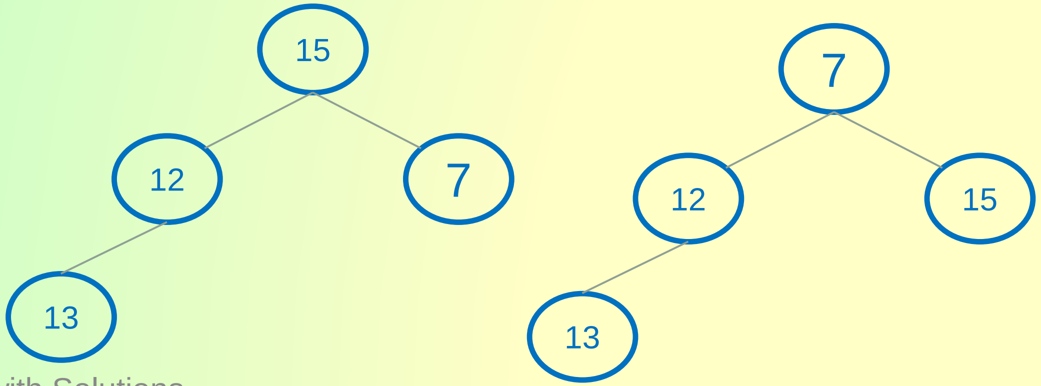
- Compare 5 and 6
  - Remove 5, replace it with 9, and 9 floats down



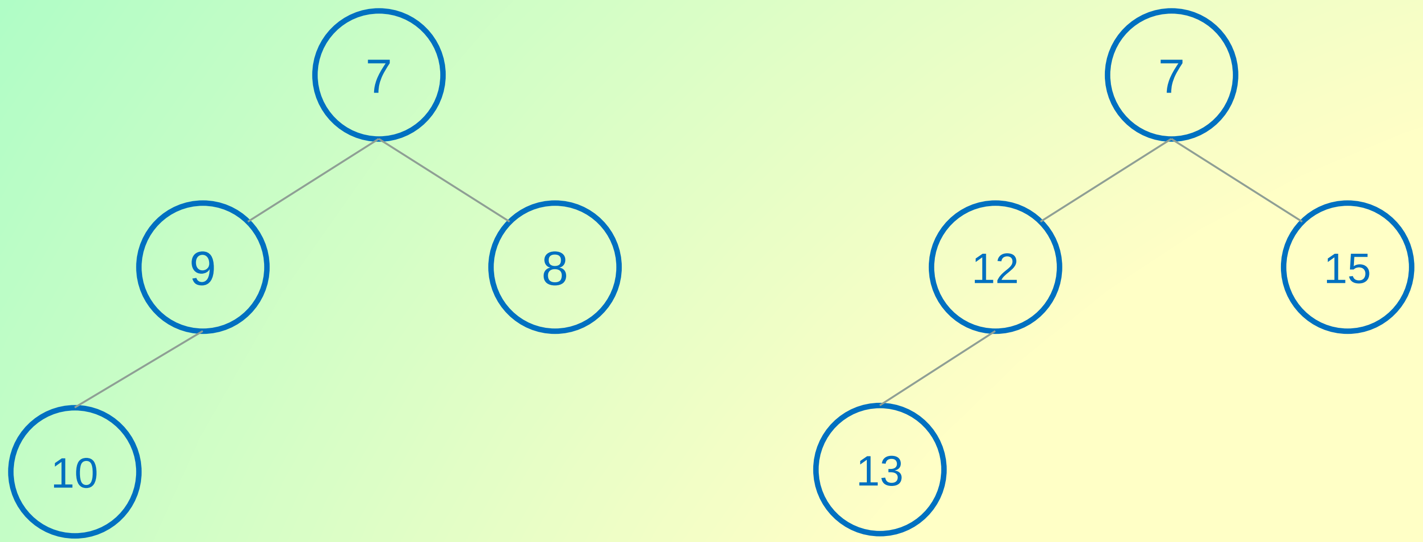
Solution 2



- Compare 7 and 6
  - Remove 6, replace it with 15, and 15 floats down



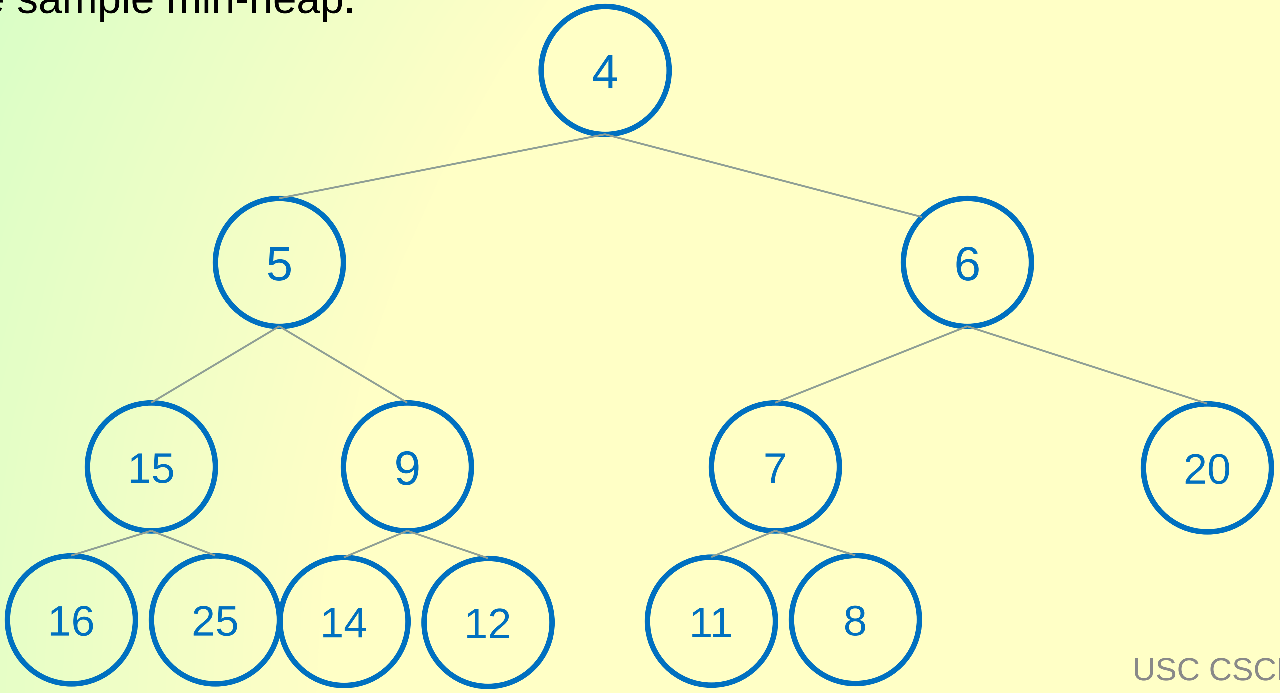
Solution 2



- Compare 7 and 7
    - Stop because we found a match

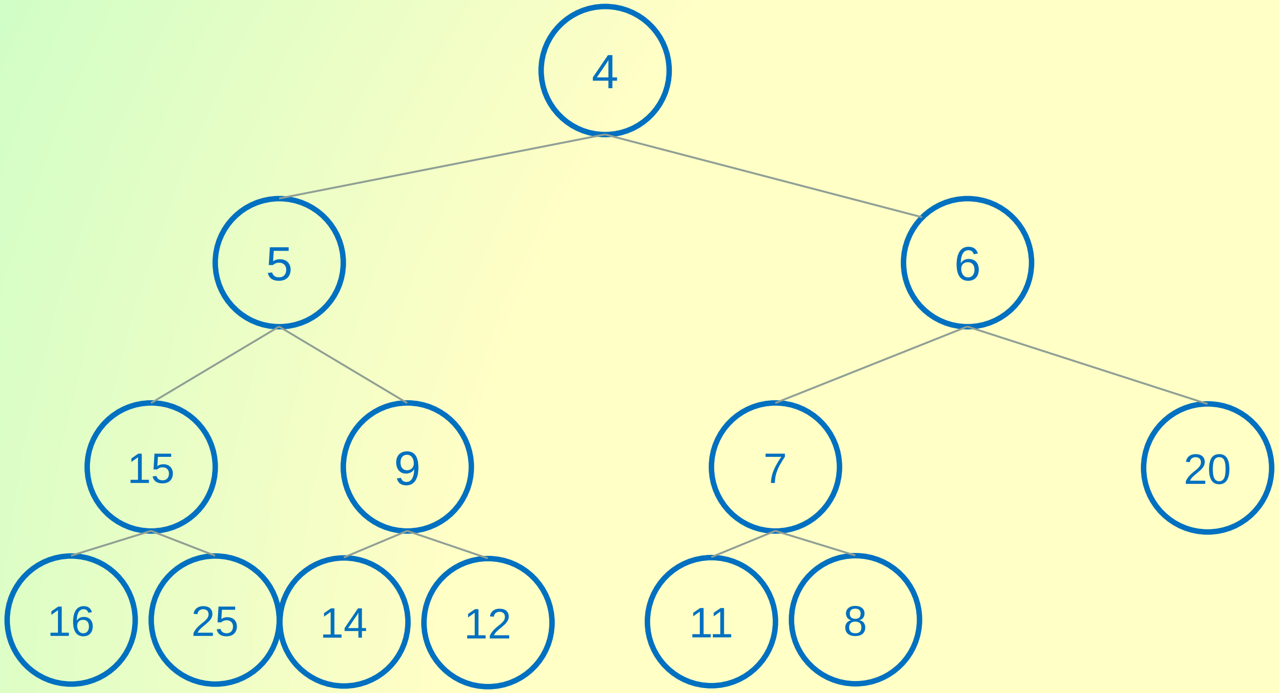
Problem 3

- Let *H* be a binary min-heap storing *n* keys. Give an efficient algorithm to report all keys in *H* that are less than or equal to a given value *x* (which may or may not be in the heap). Ideally, your algorithm should run in time *O(k)*, where *k* is the number of values returned from the heap. Note that this is independent of *n*.
- For example, if your heap is {4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8} and *x*=7, your algorithm should output 4, 5, 6 and 7 (not necessarily in that order).
  - Here is the sample min-heap.



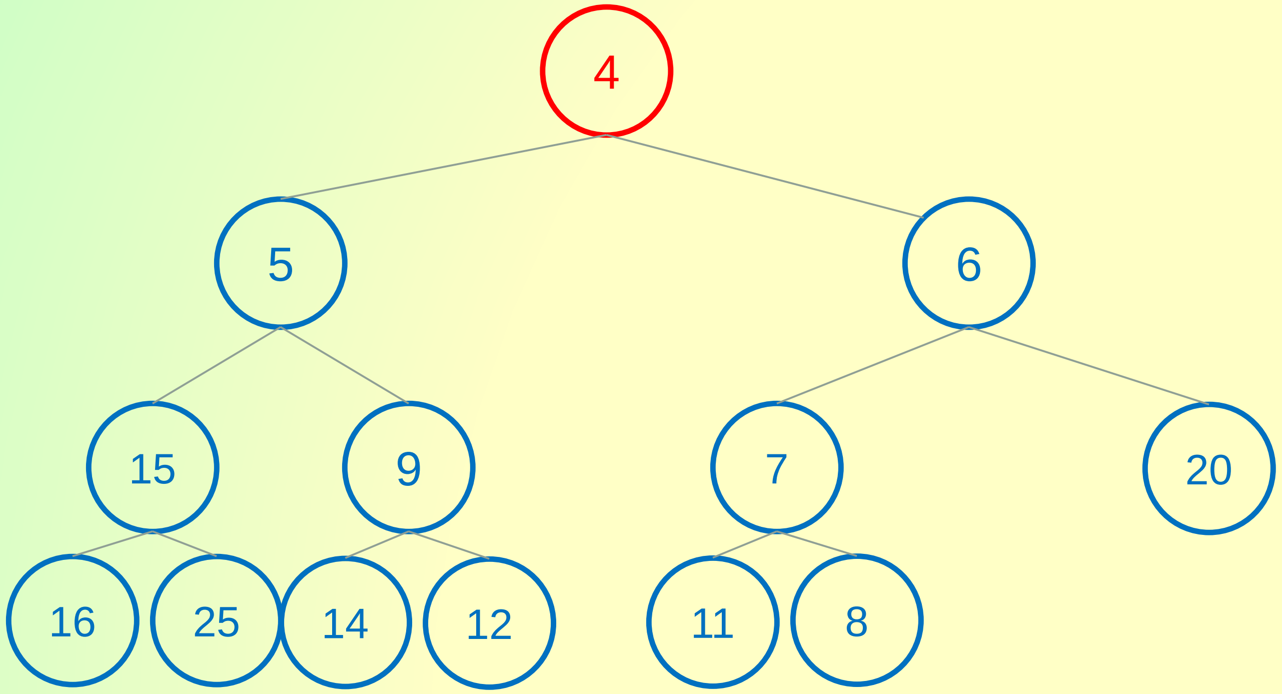
Solution 3

- If the top element is greater than x, stop.
- Otherwise, report the top element and recursively call this on each child heap
  - Each call takes O(1) time and will be called at most O(k) times recursively
  - We will have called the function a constant number of times more than k to check the children of an element we are reporting as being less than x, with a worst case of k+1 additional comparisons, giving O(k + k+1) = O(k)



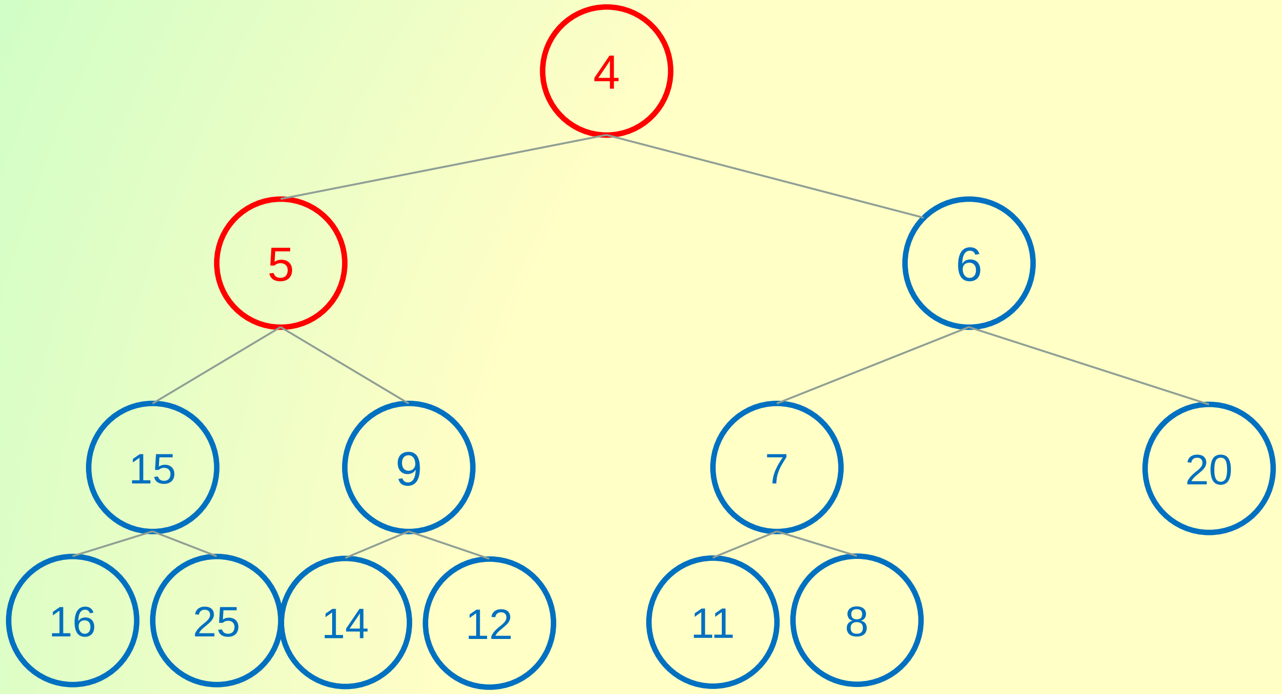
Solution 3

- Take x=7
- Compare 4 with 7 and return 4
- Recursively do this for each of the child nodes, 5 and 6



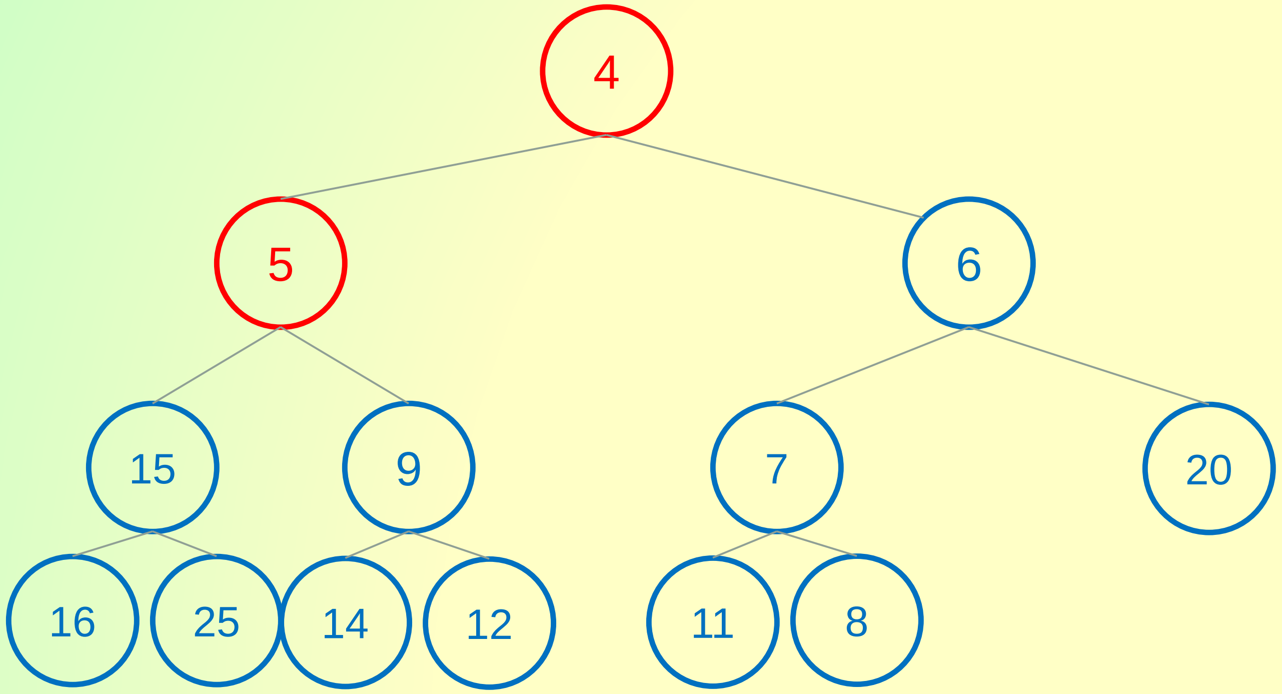
Solution 3

- With x=7, compare 5 with 7 and return 5
- Recursively do this for each of the child nodes, 15 and 9



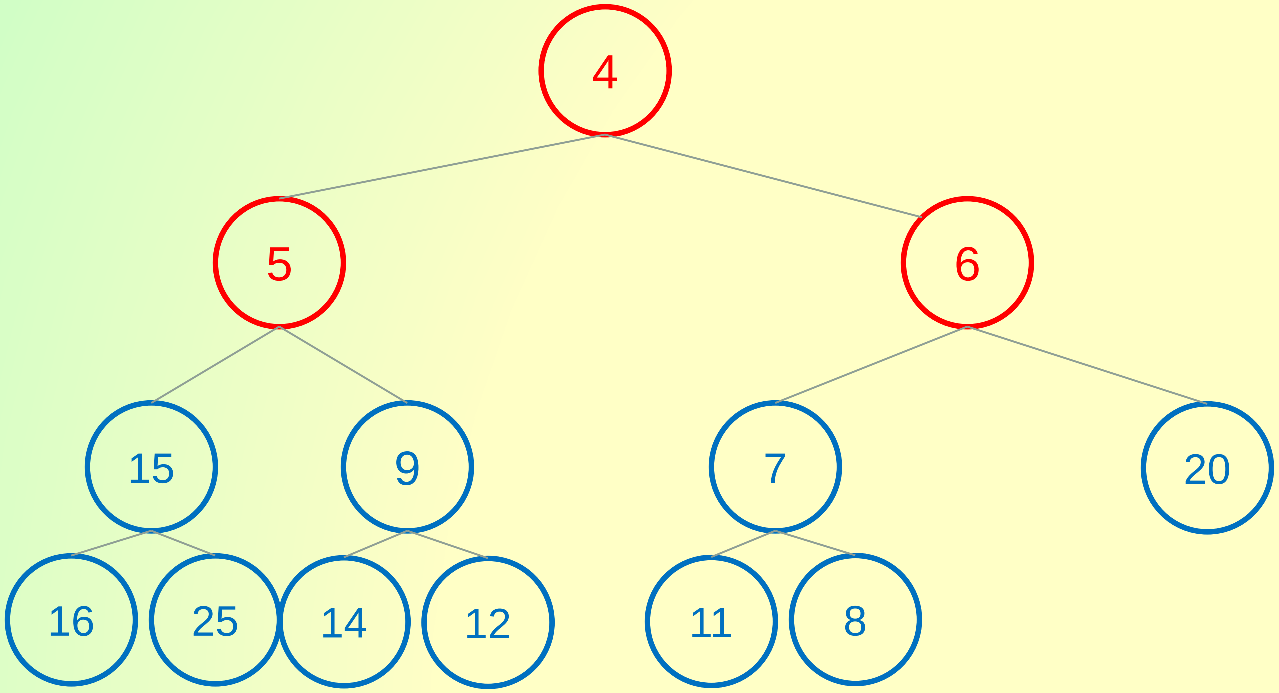
Solution 3

- With x=7, compare 15 with 7 and stop since 15 > 7
- With x=7, compare 9 with 7 and stop since 9 > 7



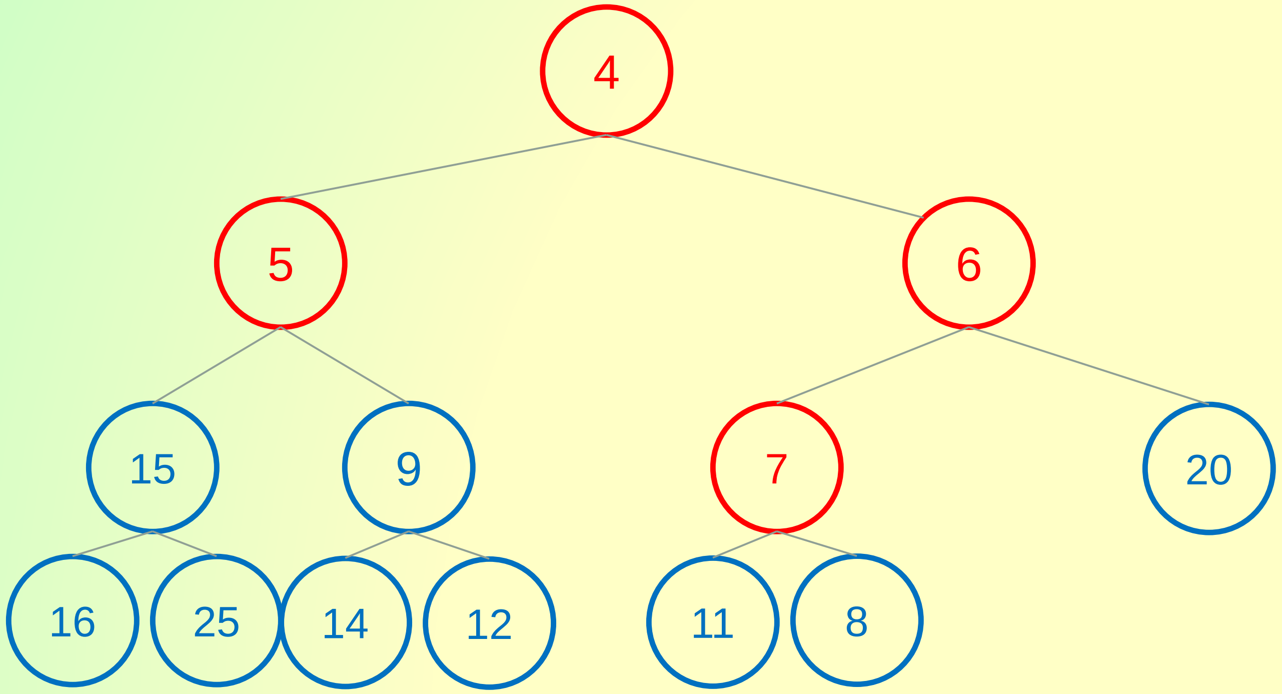
Solution 3

- With x=7, compare 6 with 7 and return 6
- Recursively do this for each of the child nodes, 7 and 20



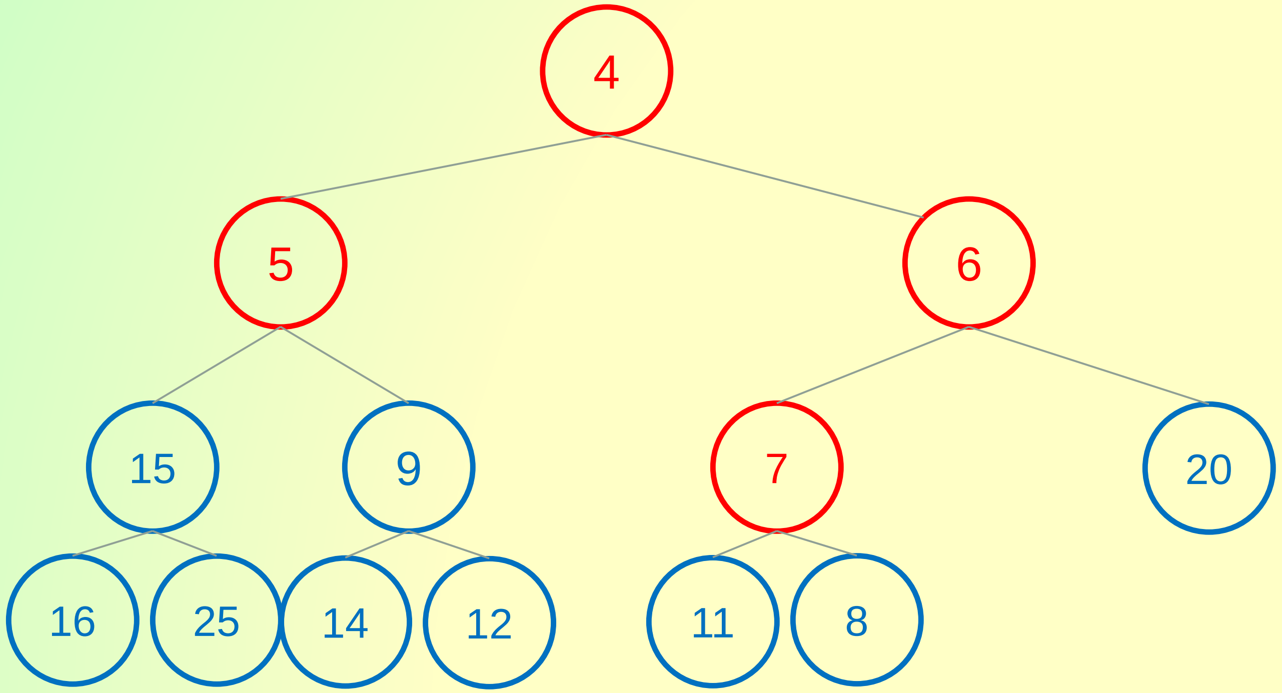
Solution 3

- With x=7, compare 7 with 7 and return 7
- Recursively do this for each of the child nodes, 11 and 8



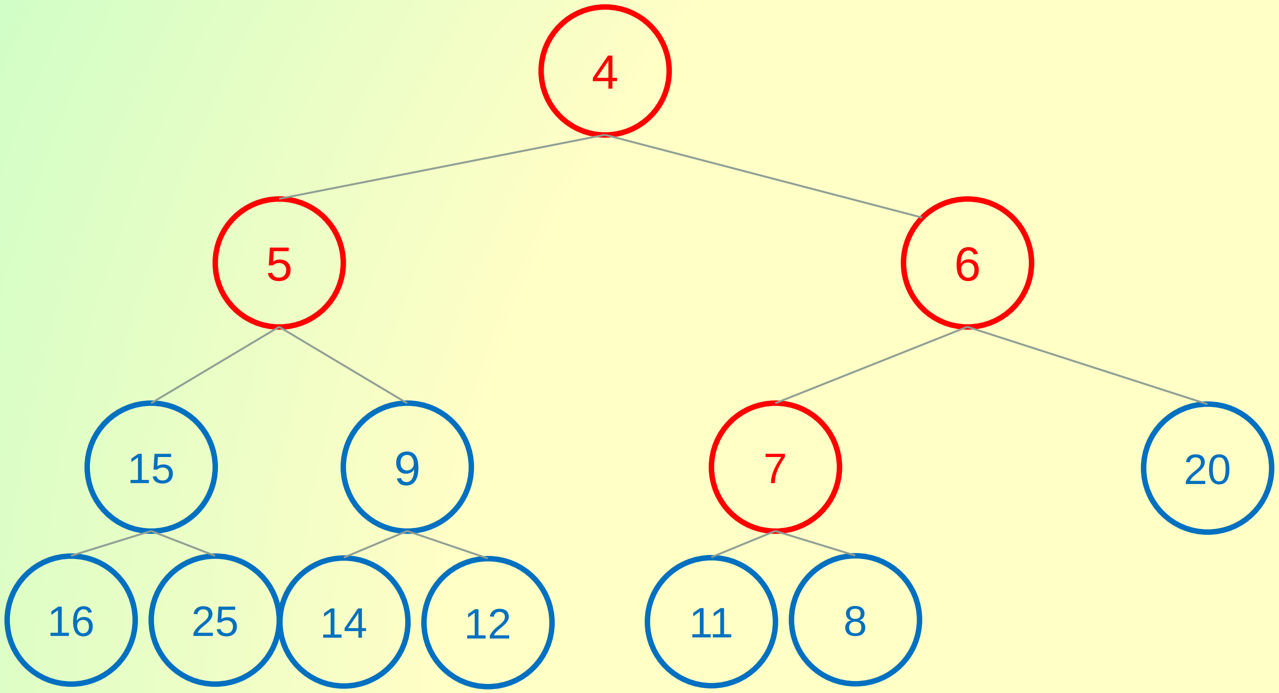
Solution 3

- With x=7, compare 11 with 7 and stop since 11 > 7
- With x=7, compare 8 with 7 and stop since 8 > 7



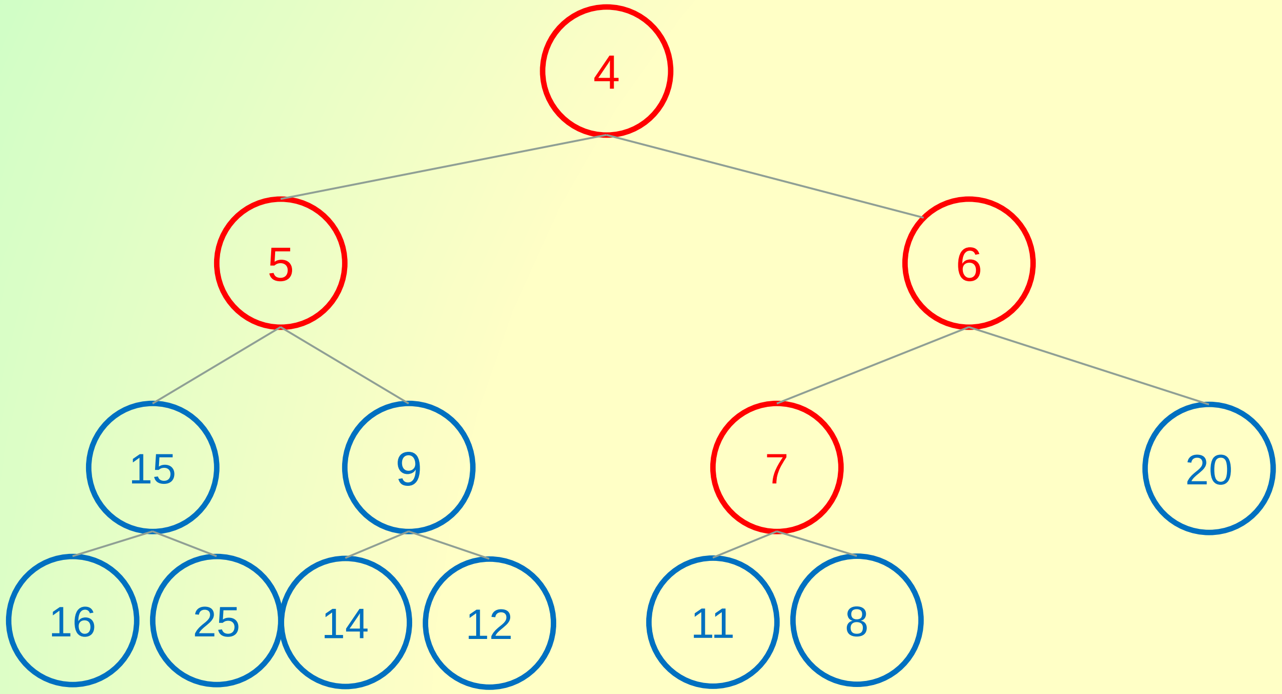
Solution 3

- With x=7, compare 20 with 7 and stop since 20 > 7



Solution 3

- We returned the values 4, 5, 6, 7, which makes k=4
- We compared x=7 to 4, 5, 6, 15, 9, 7, 20, 11, 8, which is 9 values
  - That means that we compared 5 values that we did not return, which is k+1



Solution 3

- Proof
  - Best case is when the value of x is larger than all of the values in the heap
    - Therefore, we make no additional comparisons, requiring exactly k total comparisons
  - The other case would be when we don't return all of the nodes, so there will be some comparison that are made on nodes that are not returned. We will use a proof by induction to show that the total number of comparisons will be k + k+1 in the worst case
  - Base Case – The base case is when k=1, which is when we are only going to be returning the root (k=1, since all of the other nodes in the binary min-heap will be larger than that). We will need to check both the left child and right child to ensure they are bigger than our value of x, giving us 2 additional comparisons (which is k+1).
  - Assumption – assume when we return k nodes, we make k+1 additional comparisons in the worst case
  - Induction – Assume if we return k+1 nodes, we make k+2 additional comparisons in the worst case. The worst case occurs when each node we return has 2 children that are not returned. However, some of the nodes we return may have 2 children that are returned as well. To maximize the number of nodes that need to be checked but not returned, consider the case where we return all of the nodes through level (h-1, where h is the height) of the binary min-heap. There are 2^{h-1} -1 nodes that are returned through level h-1 and 2^{h-1} nodes that are **not** returned at level h. With our assumption of returning k+1 nodes, that means that k+1 = 2^{h-1} -1, giving k+2 = 2^{h-1}, which is the total number of additional comparisons in the worst case as we just stated.