NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

Cachar, Assam

B.Tech. IVth Sem

Subject Code: CS206

Subject Name: Algorithms

Submitted By:

Name	: Subhojit Ghimire
Sch. Id.	: 1912160
Branch	: CSE – B

1. What are the minimum and maximum number of elements in a heap of height h?
   ➔ Minimum number of elements: 2^h
   Maximum number of elements: $2^{h+1} - 1$.

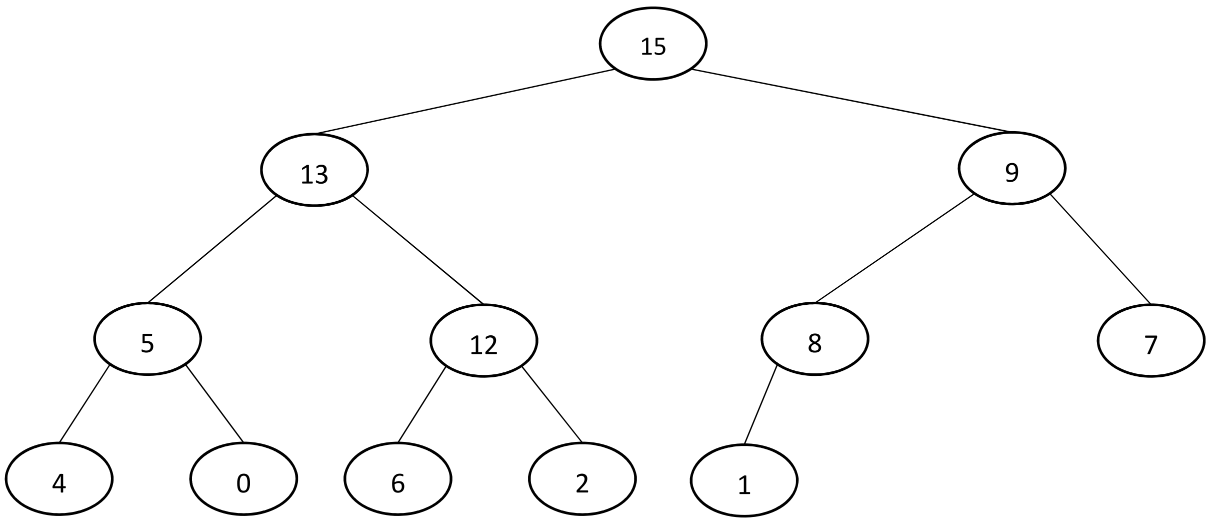
2. Show that, with the array representation for storing an n-element heap, the leaves are the nodes indexed by [n/2] + 1, [n/2] + 2. ..., n.
   ➔ For $\left[\frac{n}{2}\right] + 1$,
   If the left child existed, it would be $= 2\left(\left[\frac{n}{2}\right] + 1\right)$
   $= n + 2$
   $> n$
   i.e., the left child has the index larger than the number of elements in the heap, which is not possible and hence, it proves that the node indexed (n/2 + 1) doesn't have any children. Hence it is a leaf.
   The same holds true for the increasing index terms.
   Hence, verified.

3. Show that the worst-case running time of MAX-HEAPIFY on a heap of size n is Ω (lg n). (Hint: For a heap with n nodes, give node values that cause MAX-HEAPIFY to be called recursively at every node on a simple path from the root down to a leaf.)
   ➔ To show: Worst-case running time of MAX-HEAPIFY on a heap of size n = Ω (lg n)
   Known: The height of a heap with size n = lg n
   If we put a value at the parent node that is less than the elements at the children nodes, the MAX-HEAPIFY will be called recursively until a leaf is reached, which is Θ (lg n) times.
   Hence, it's worst-case running time is Ω (lg n).

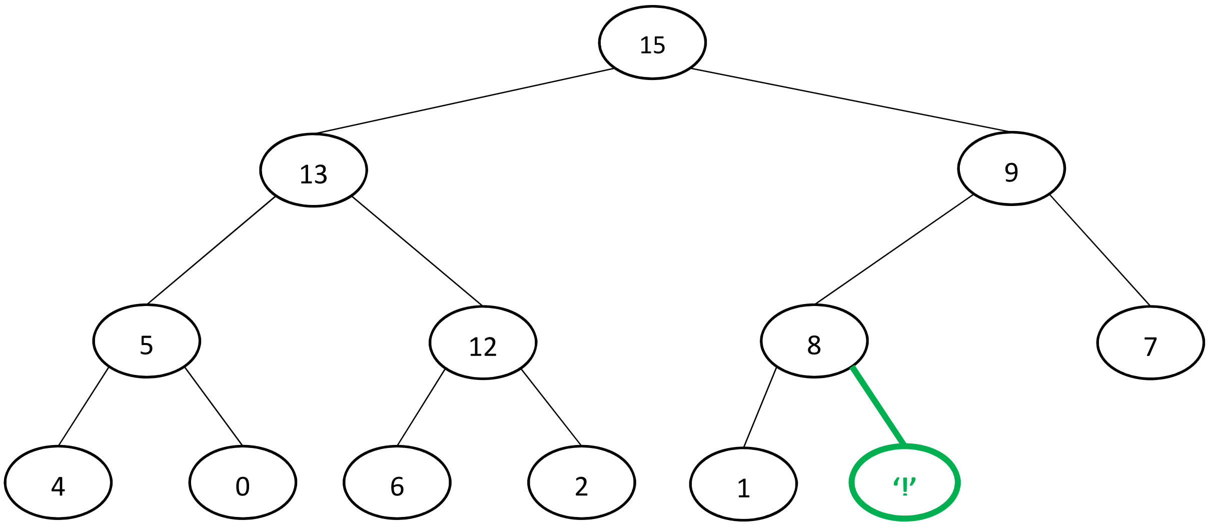
4. Why do we want the loop index i in line 2 of BUILD-MAX-HEAP to decrease from [A.length/2] to 1 rather than increase from 1 to [A.length/2]?
   ➔ The BUILD-MAX-HEAPIFY functions under the assumption that the children nodes are both the max-heaps to their own children elements.
   So, if we loop the index i from 1 to [A.length/2] in, for example, the array A [] = {10, 5, 4, 20, 3}, then the element 5 will swap with the element 20 at second iteration to make the new array as {10, 20, 5, 4, 3} and end the process as it reaches the [A.length/2] iterations. In this case, the children node element 20 is still larger than the parent node element 10, which defies the property of MAX-HEAP.
   However, if we decrease from [A.lenght/2] to 1, it will first swap 5 with 20 in the first iteration, and then 10 with 20 in the second (or final, in this case) iteration, making the final array as {20, 10, 4, 5, 3} which is the required max heap.

5. Show that there are at most [n/2^{h+1}] nodes of height h in any n-element heap.
   ➔ For a heap with n-elements, the leaves are indexed as:
   $$\left[\frac{n}{2}\right]+1,\left[\frac{n}{2}\right]+2,\left[\frac{n}{2}\right]+3,\ldots,n$$
   This mean, there are $\left[\frac{n}{2}\right]$ number of leaves in any heap of size n.
   Proof by induction,
   Let, n_h be the number of nodes at height h
   For, h = 0,
   $n_0 = \left[\frac{n}{2}\right]$, which is true
   Assuming, it holds true for h – 1.
   To verify: $n_h = \frac{n}{2^{h+1}}$
   When n_{h-1} is even,
   $$n_h = \left[\frac{n_{h-1}}{2}\right]$$
   When n_{h-1} is odd,
   $$n_h = \left[\frac{n_{h-1}}{2}\right]+1 > \left[\frac{n_{h-1}}{2}\right]$$
   This gives,
   $$n_h \le \left[\frac{1}{2}\left[\frac{n}{2^{(h-1)+1}}\right]\right] = \left[\frac{1}{2}\frac{n}{2^h}\right] = \frac{n}{2^{h+1}}$$
   Hence, verified.

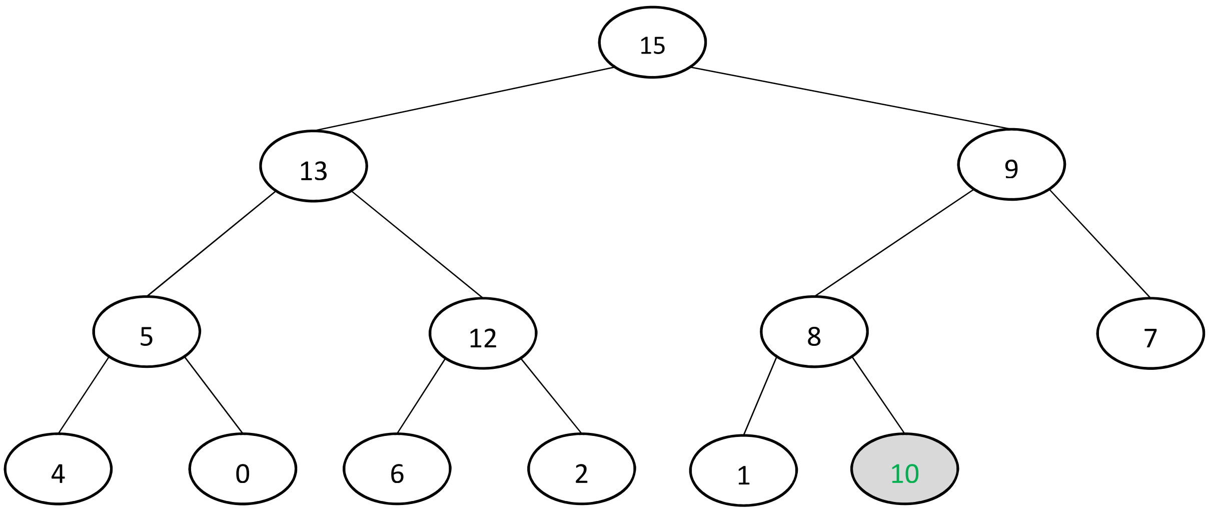
6. Illustrate the operation of MAX-HEAP-INSERT (A, 10) on the heap A = (15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1).
   ➔ Max Heap A = (15, 13, 9, 5, 12, 8, 7, 4)



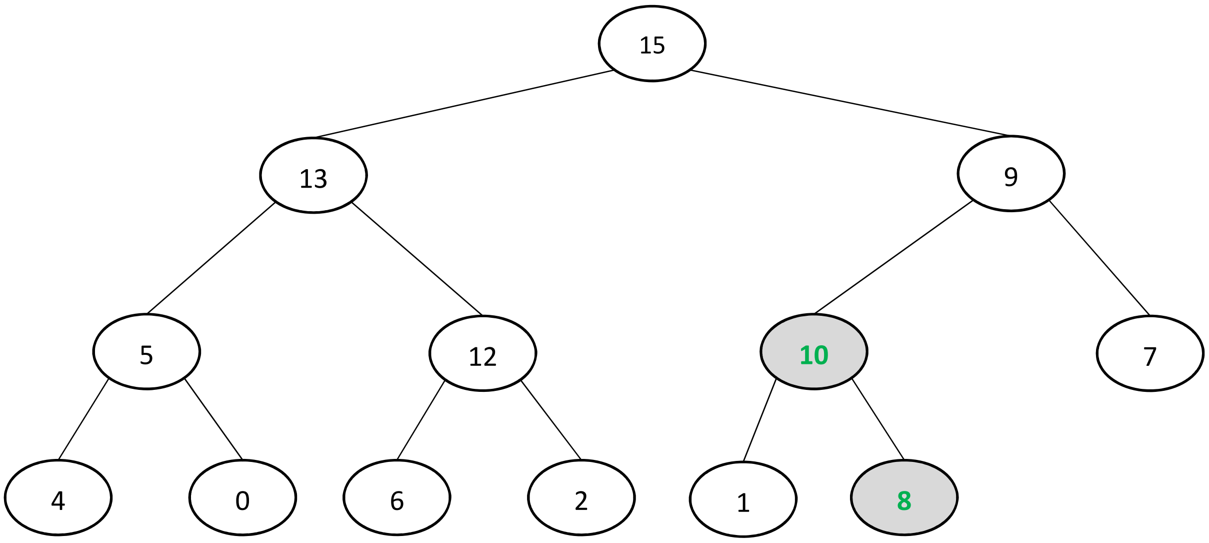
Calling MAX-HEAP-INSERT (A, 10) and assigning a node with undefined value



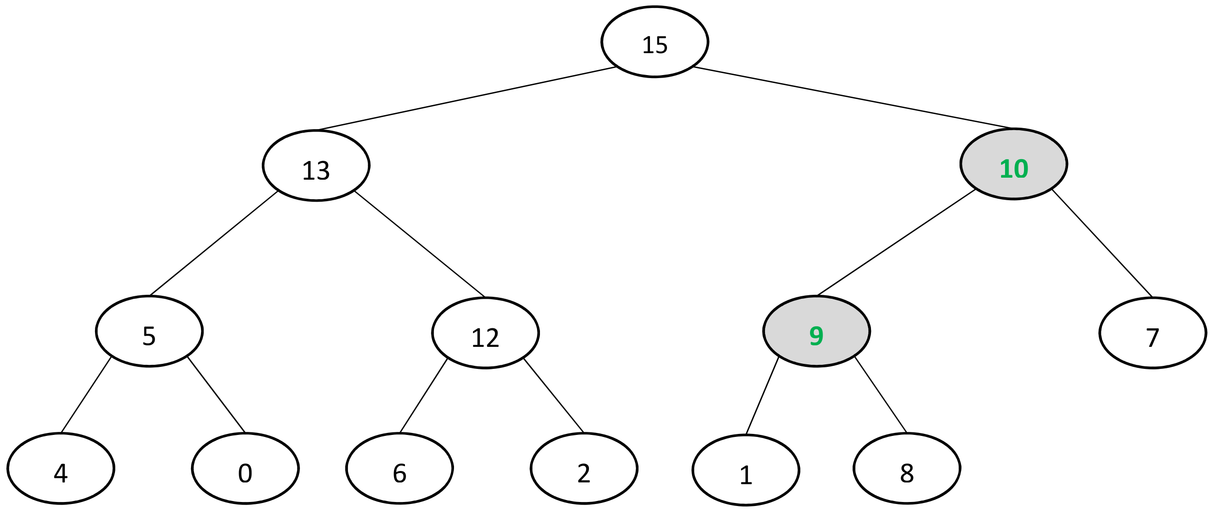
Updating the newly assigned node with value of 10



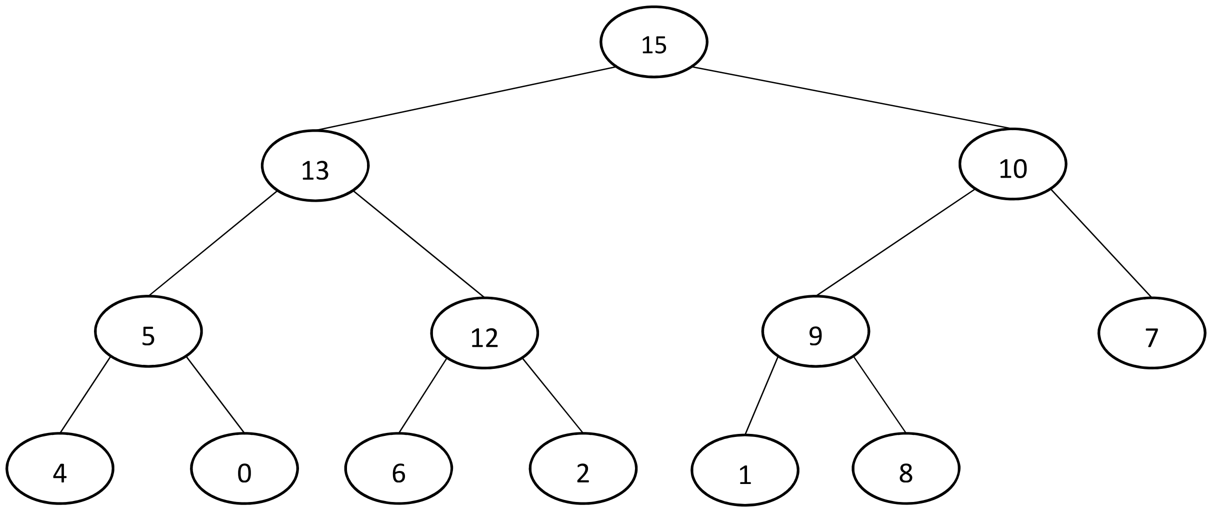
MAX-HEAPIFY Iteration I: (Swapping 8 with 10)



MAX HEAPIFY Iteration II: (Swapping 9 with 10)



Termination



7. Give an O(n lg k)-time algorithm to merge k sorted lists into one sorted list, where n is the total number of elements in all the inputs lists. (Hint: Use a min-heap for k-way merging.)

➔ Algorithm:

1. Construct min heap from heads of k lists in each.
2. Find next element in sorted array and find the minimum element.
3. Add next element from shorter list (from where minimum element was taken) to heap.

Finding the next element in the sorted list takes the maximum of O(lg k) time, hence to find the whole list of n elements, it takes total of O(n lg k) total steps.

8. Use the substitution method to prove that the recurrence T (n) = T (n – 1) + Θ (n) has the solution T (n) = Θ (n^2) as claimed at the beginning of Section 7.2.
   - ➔ By the definition of Θ, we can say that there exists c_1 and c_2 such that,

     $$c_1 n \leq \Theta(n) \leq c_2 n$$

     From the equation, we make an assumption,

     $$c_1 n^2 \leq T(n) \leq c_2 n^2$$

     This give, T (n) = T (n – 1) + Θ (n)

     $\leq c_1 (n-1)^2 + c_2 n$, taking Θ (n) as $c_2 n$

     $= c_1 n^2 - 2c_1 n + c_1 + c_2 n$

     $\leq c_1 n^2$, because, $2c_1 > c_2$; $n \geq \frac{c_1}{(2c_1 - c_2)}$

     $= \Theta(n^2)$

     Hence, proved.

9. When RANDOMIZED-QUICKSORT runs, how many calls are made to the random number generator RANDOM in the worst case? How about in the best case? Give your answer in terms of Θ-notation.
   - ➔ In both the cases, quicksort makes Θ (n) calls.
     Explanation:
     RANDOMIZED-QUICKSORT at the worst case,

     $$T(n) = T(n-1) + 1 = n = \Theta(n)$$

     RANDOMIZED-QUICKSORT at the worst case,

     $$T(n) = 2\,T\left(\frac{n}{2}\right) + 1 = \Theta(n)$$

10. Show that in the recurrence,

    $$T(n) = \max_{0 \leq q \leq n-1} \big(T(q) + T(n-q-1)\big) + \theta(n)$$

    $$T(n) = \Omega(n^2)$$

    - ➔ By the definition of Θ, we can say that there exists c_1 and c_2 such that,

      $$c_1 n \leq \Theta(n) \leq c_2 n$$

      From the equation, we make an assumption

      $$c_1 n^2 \leq T(n) \leq c_2 n^2$$

      This gives, T (n) $= \max_{0 \leq q \leq n-1} \big(T(q) + T(n-q-1)\big) + \theta(n)$

      $\geq \max_{0 \leq q \leq n-1} \left(cq^2 - 2q + c(n-q-1)^2 - 2n - 2q - 1\right) + \theta(n)$

      $= c \max_{0 \leq q \leq n-1} \left(q^2 + (n-q-1)^2 - \frac{(2n+4q+1)}{c}\right) + \theta(n)$

      $= cn^2 - c(2n - 1) + \Theta(n)$

      $= cn^2 - 2cn + 2c$

      $\geq cn^2 - 2n$, for $c \leq 1$

      $= \Omega(n^2)$

11. Prove that COUNTING-SORT is stable.
    - ➔ Supposing, positions i and j with $i < j$ both contain some element k.
      Considering, lines 10 through of counting sort, where output array is to be constructed.
      Since, $j > i$, so the loop will check A [j] before checking A [i]. When this happens, the algorithm places A [j] in position m = C [k] of B.
      As C [k] is decremented in line 12, and never incremented again, it can be said that when the 'for' loop checks A [i], at that time, C [k] < m.
      This places A [i] in an earlier position of the output array.
      Hence, this proves the stability.

12. Use induction to prove that radix sort works. Where does your proof need the assumption that the intermediate sort is stable?
    - ➔ Loop Invariant (Induction) method to prove the working of radix sort:
      1. Initialisation: The array is sorted in the last $i – 1$ digits at the beginning of the for loop. This ensure that the array is trivially sorted on the last 0 digits.
      2. Maintenance: After the sorting operation on the ith digit, the array will be sorted on the last i digits. The elements in with varying digit in the ith position are ordered accordingly.
      3. Termination: When the iteration $i = d + 1$, the loop ends, sorting the loop on d digits.

      In the maintenance step, the assumption that the intermediate sort is stable is applied.

13. Show how to sort n integers in the range 0 to $n^3 – 1$ in O(n) time.
    - ➔ Firstly, Convert every integers in the list of n integers to base n. Secondly, Radix Sort the converted integer list. This ensure that every number will have at most of $\log_n n^3 = 3$ digits, so that there will only be the requirement to have 3 passes.
      For each pass, there are n possibilities which can be assigned, which will be sorted using counting sort to sort each digit in O(n) time.

14. Explain why the worst-case running time for the bucket sort is $\Theta(n^2)$. What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time O(n lg n)?
    - ➔ In the worst case, bucket sort would contain all the n items in the single bucket in reverse order, which will then use insertion sort technique to sort it. As insertion sort takes $\Theta(n^2)$, bucket sort takes the same.
      The insertion sort algorithm in the bucket sort can be replaced with either merge sort algorithm or heap sort algorithm, which would make the worst-case running time O(n lg n).