**Chapter 06**

Q1. What are the minimum and maximum numbers of elements in a heap of height h?

Ans. Minimum number of elements would be 2h  and maximum no of element would be 2h+1 -1 . Because in a tree with height (h-1) there are 20 + 21 + 22 +….+ 2h-1 = = 2h -1 elements. Hence in hth layer there arises two cases.

1. Only 1 element is there . (No. of elements will be 2h -1 +1 = 2h )
2. Full binary Tree (No. of elements will be 2h - 1 + 2h = 2h+1 -1)

Q2. Show that an n-element heap has height ⌊lg n⌋.

Ans . InFormal answer:

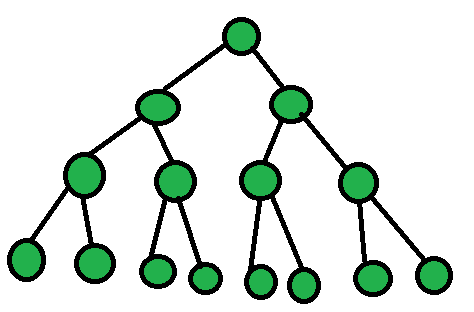
Write n = 2m −1+k where m is as large as possible. Then the heap consists of a complete binary tree of height m − 1, along with k additional leaves along the bottom. The height of the root is the length of the longest simple path to one of these k leaves, which must have length m. It is clear from the way we defined m that m = ⌊lg n⌋.

Formal Proof:

There will be two cases in total

1. Full binary tree
2. Only on element (leftest).

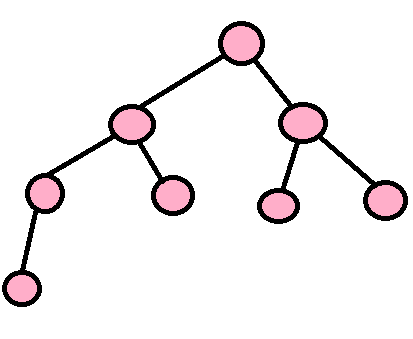
a).



**(Full Binary Tree)**

n = 2h+1  -1 ⇒ lg (n+1) = (h+1) lg2 ⇒ lg (n +1)= (h +1) [lg2 2 = 1] ⇒ h = lg (n+1) -1 --(1)

b). Only left child



**(Only one left as leftest node)**

N = 2h ⇒ lg(n) = h lg 2 ⇒ h = lg(n) --(2)

From 1 and 2 we get

2h ≤ n ≤ 2h+1  -1 ⇒ h ≤ lg ( n) ≤ h +1

By defination of floor function

X = ⌊x⌋ + α

where 0 ≤ α <1

h = ⌊lg n⌋

Q3. Show that in any subtree of a max-heap, the root of the subtree contains the largest value occurring anywhere in that subtree.

Ans. Let us assume that there exist a node in the subtree whose value is greater than the root. If it is not the root then it must have a parent. Since it’s value is greater than it’s parent. We voilated the property of a max-heap. Hence it contradict the fact that it’s max heap. So, root of the subtree has largest element.

Q4. Where in a max-heap might the smallest element reside, assuming that all elements are distinct?

Ans. Min. element will surely be the smallest element. Proof by contradiction.

Let us assume (x- smallest node) is an internal node. So it must be having child/ren . Since in max-heap parent is always greater than it’s child. X will be having smaller children which contradict the fact that x is smallest. Hence x cann’t be internal node and must be lead node.

Q5. Is an array that is in sorted order a min-heap?

Ans. Yes, it is. Because every parent(i) is smaller than its child(2i+1 or 2i).

Eg: {1,2,3,4,5,6} .

1 is smaller than 2(left child) and 3 (right child)

2 is smaller than 4(left child) and 5 (right child)

Q6. Is the array with values {23,17,14,6,13,10,1,5,7,12} is a max-heap?

Ans. No,it’s not. 6’s child must be smaller than 6. But 6 at pos 4 has left and right child at 8 and 9 respectively.

At 9th pos 7 (right child) is greater than its parent.

Q7. 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/2⌋3 …….n.

Ans. To justify this claim it is suffice to show that element indexed by {⌊n/2⌋+1, ⌊n/2⌋+2,⌊n/2⌋3 …….n} have no child. Suppose that we have a x in this range. This is it’s child would be located at 2x and 2x+1 . But as you may notice that both these indices are out of bound ({2⌊n/2⌋+2) and don’t fall inside the A.heapsize. Hence x can’t belong to this bound. It mean within the bound no element has any child. So they must be leaf nodes.

Q1. Using Figure 6.2 as a model, illustrate the operation of MAX-HEAPIFY(A,3) on the array {27,17,3,16,13,10,1,5,7, 12,4,8,9,0}.

Ans.

MAX-HEAPIFY(A,3)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 27 | 17 | ***3*** | 16 | 13 | 10 | 1 | 5 | 7 | 12 | 4 | 8 | 9 | 0 |

↓

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 27 | 17 | 10 | 16 | 13 | ***3*** | 1 | 5 | 7 | 12 | 4 | 8 | 9 | 0 |

↓

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 27 | 17 | 10 | 16 | 13 | 9 | 1 | 5 | 7 | 12 | 4 | 8 | ***3*** | 0 |

Q2. Starting with the procedure MAX-HEAPIFY, write pseudocode for the procedure MIN-HEAPIFY(A,i), which performs the corresponding manipulation on a min-heap. How does the running time of MIN-HEAPIFY compare to that of MAX-HEAPIFY?

Ans.

Min-Heap pseudocode. (For C++ code see file)

MIN-HEAPIFY(A, i)

l = LEFT(i)

r = RIGHT(i)

if l ≤ A.heap-size and A[l] < A[i]

smallest = l

else smallest = i

if r ≤ A.heap-size and A[r] < A[smallest]

smallest = r

if smallest != i

exchange A[i] with A[smallest]

MIN-HEAPIFY(A, smallest)

The running time of MIN-HEAPIFY is the same as that of MAX-HEAPIFY.

Q3. What is the effect of calling MAX-HEAPIFY(A,i) when the element A[i] is larger than its children?

Ans. The array remains unchanged since the if statement on line line 8 will be false.

Q4. What is the effect of calling MAX-HEAPIFY(A,i) for i > A.heap-size/2?

Ans. The array remains unchanged. If i > heapsize.2 then it’s left child and right child will be out of bound. Hence they must be leaf node. And hence they don’t voilate any MAX-HEAPIFY property. Also statement conditions on lines 3 and 6 of the algorithm will never be satisfied. Therefore largest = i so the recursive call will never be made and nothing will happen.

Q5. The code for MAX-HEAPIFY is quite efficient in terms of constant factors, except possibly for the recursive call in line 10, which might cause some compilers to produce inefficient code. Write an efficient MAX-HEAPIFY that uses an iterative control construct (a loop) instead of recursion.

Ans. Pseudocode (For C++ code see file)

MAX-HEAPIFY(A, i)

while true

l = LEFT(i)

r = RIGHT(i)

if l ≤ A.heap-size and A[l] > A[i]

largest = l

else largest = i

if r ≤ A.heap-size and A[r] > A[largest]

largest = r

if largest == i

return

exchange A[i] with A[largest]

i = largest

Q6. Show that the worst-case running time of MAX-HEAPIFY on a heap of size n is ⌊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.)

Ans. Consider the heap resulting from A where A[1] = 1 and A[i] = 2 for 2 ≤ i ≤ n. Since 1 is the smallest element of the heap, it must be swapped through each level of the heap until it is a leaf node. Since the heap has height ⌊lg n⌋, MAX-HEAPIFY has worst-case time Ω(lg n).

Eg: {5,6,6,6,6,6,6,6,6,6….}. Consider this array. Since array root is smaller 5 whill slip down to bottom of tree.

Q1.