ISSN No. 0976-5697

Volume 6, No. 1, Jan-Feb 2015

International Journal of Advanced Research in Computer Science

TECHNICAL NOTE

Available Online at www.ijarcs.info

The Heap Structure and Its Applications

C. R. Kavitha
SLN College of Sciences
Andhra Pradesh, India

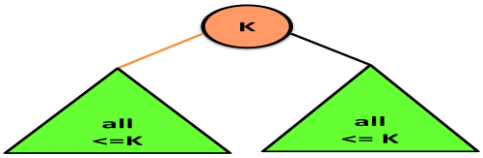
Abstract :A heap is a partially sorted binary tree. Like the binary trees, heaps have a meaning for the Left and Right subtrees. The root of a heap is guaranteed to hold the largest node in the trees; its subtrees contain data that have lesser values. Unlike the binary search tree, however, the smaller nodes of a heap can be placed on either the Right or Left subtree. Therefore, both the Left and Right branches of the tree have the same meaning .Heaps have another interesting facet: They are often implemented in an array rather than a linked list. This implementation is possible because the heap is, by definition, complete or nearly complete. This allows a fixed relationship between each node and its children. There are two factors at work: the time it takes to create a heap by adding each element and the time it takes to remove all of the elements from a heap. Fortunately, we have a guarantee that adding a single element to and removing a single element from a heap both take O(log(n)) time.

Keywords: Heap, Reheapup, Reheapdown, Maxheap, Minheap

I. INTRODUCTION

A Heap is a Binary Tree Structure with the following properties:

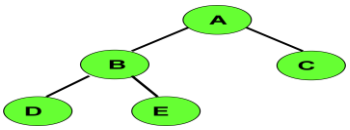
a) The tree is Full (or) Complete Binary Tree.
b) The key value of each node is greater than or equal to the key value in each of its descendents [1].



Like Binary Search Tree, Heaps have two properties are as follows [4]:

a. Structure Property:

A heap is a binary tree that is completely filled, with the possible exception of the bottom level, which is filled from left to right. The following figure shows a complete binary tree.



A complete binary tree of height 'h' has between 2^h and 2^{h-1} node. This implies that the height of complete binary tree is [log N].

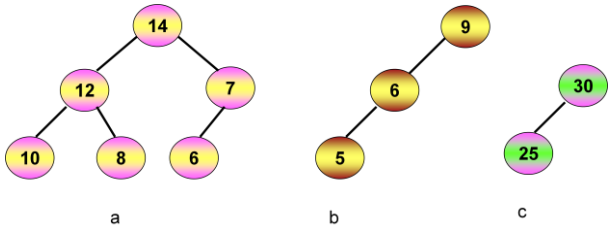
b. Heap Order Property:

The property that allows operation to be performed quickly is the heap-order property. Since we want to find the minimum element, the smallest element should be at the root [4]. The key of the parent node is always smaller than the key of the child node, that is parent < key of child [5].

II. MAX TREE & MIN TREE

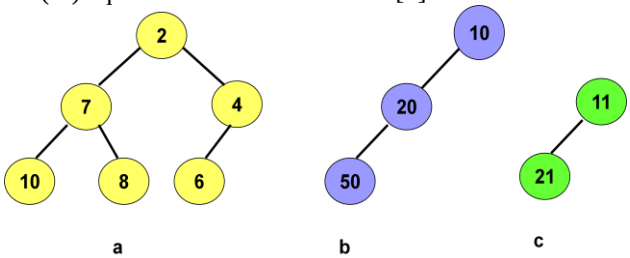
a. Max Tree:

A Max tree is a tree in which, value in each node is greater than (or) equal to those in its children [2].



b. Min Tree:

A Min tree is a tree in which, value in each node is less than (or) equal to those in its children [2].



It's not necessary for a max tree to be binary. Nodes of a max (or) min tree may have an arbitrary number of children.

III. TYPES OF HEAP

a. Max Heap:

A max heap is a max tree that's also a complete binary tree [2].

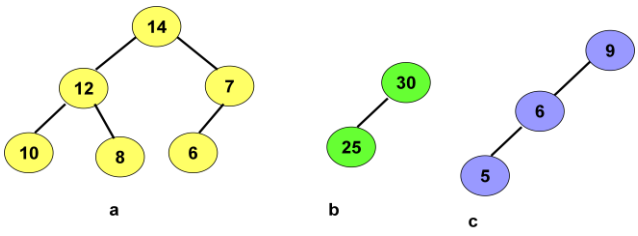


Figure (a) & (b) are Max heap, because it's a complete binary tree Fig (c) is not a Max heap because, it's not a complete binary.

b. Min Heap:

A min heap is a min tree that's also a complete binary tree [2].

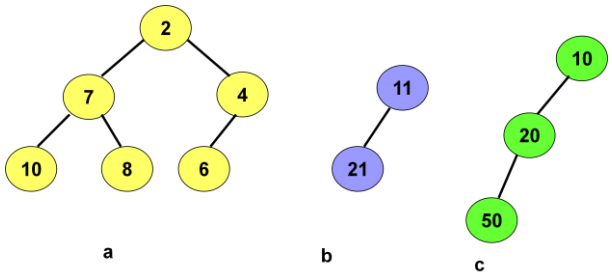


Figure (a) & (b) are Min heap, because both the trees are complete binary trees. Fig (c**)** is not a Min heap because it's not a complete binary.

IV. HEAP OPERATIONS

Two operations: Insert and Delete. To implement the insert and delete operations, we need two basics operations: Reheap Up and Reheap Down.

A. Reheap up (Insertion):

This operation will take place when inserting a new node into the tree [1].

B. Re-heap down (Deletion):

This operation will take place when deleting a node from the tree [1].

a) Reheap UP (Insertion):

The Reheap-Up operation repairs a "broken" heap by floating the last element up the tree until it is in its correct location in the heap. This was showed in the following example.

Step-1

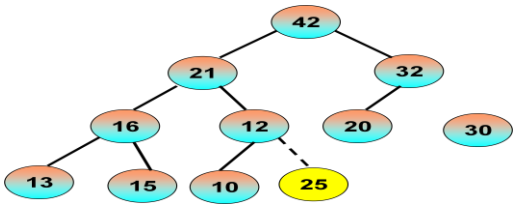


Figure.1 Original Tree: not a heap after insertion of node (25)

Step-2

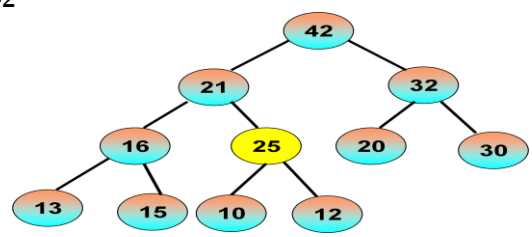


Figure. 2 Node (25) was floating to reach its position

Step-3

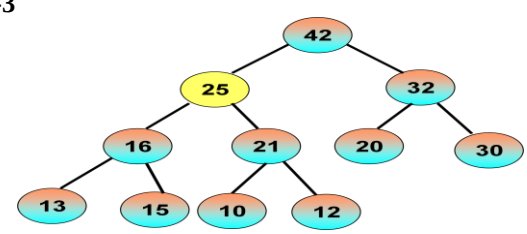


Figure.3 Node (25) was placed in the original position

b) Insertion into a Max Heap:

Let us take the following figure

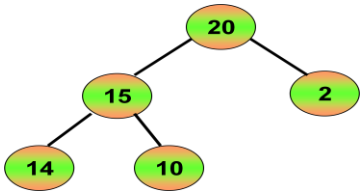


Figure.1

This is the Max Heap with five elements. When an element is added to this heap, the resulting 6th element position is shown in the following figure:

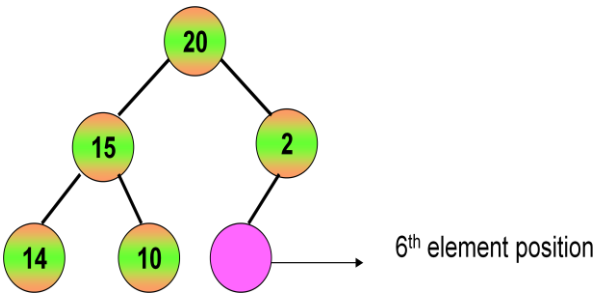


Figure. 2

The insertion can be completed by placing the new element into the new node and then bubbling the new element up the tree (along the path from the new node to the root) until the new element has a parent whose priority is > = of the new element. Suppose if we want to insert the element '1', (in Fig-1), it may be inserted as the left child of the node '2'.

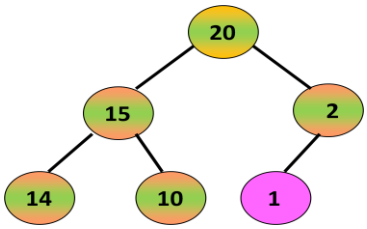


Figure. 3

Instead of inserting the element '1', insert the element '5', (in Fig-1) this element is placed as the left child. But according to the definition of Max heap, the element '2' is moved down to it's left child & 5 is bubbled up one node as follow:

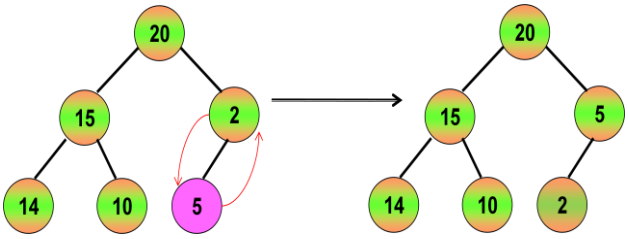


Figure. 4

If we want to insert the element '21', in Fig-1, it's inserted as a left child of '2', so 21 is inserted as a 'ROOT' node & 20 becomes right child of the element 21 and the element '2' become left child of '20'.

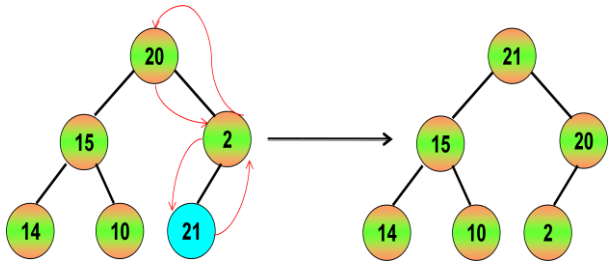


Figure. 5

c) Algorithm for heap insertion:

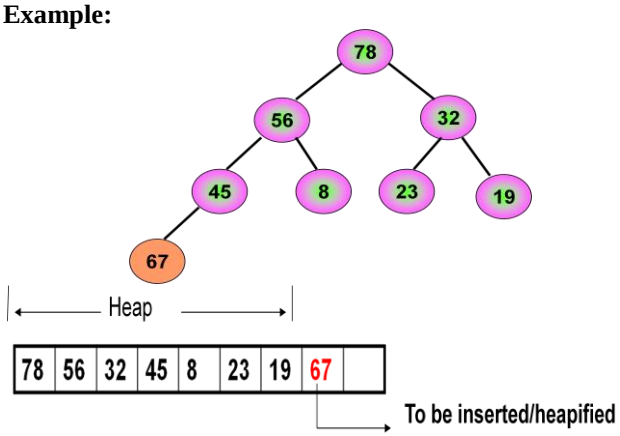
algorithm insertHeap (ref heap<arrayof datatype>
ref last <index>,
insert data into heap)

Precondition : heap is a valid heap structure
Postcondition : data have been inserted into heap
Return : True if successful ; False if array full;

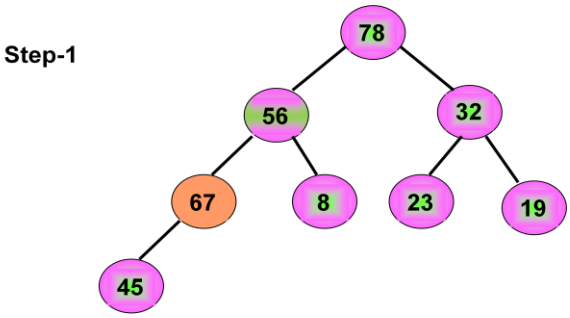
a. if (heap full)
b. end if
c. last = last + 1
d. heap [last] = data
e. reheapup (heap, last)
f. return true

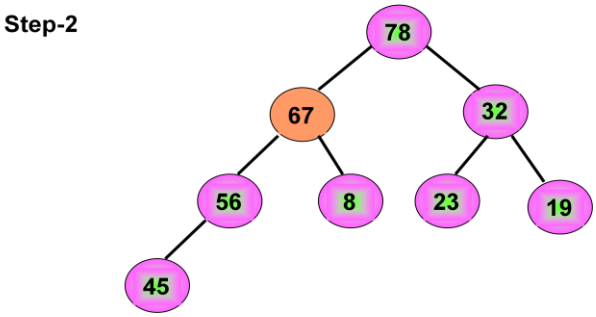
End inserheap

Example:

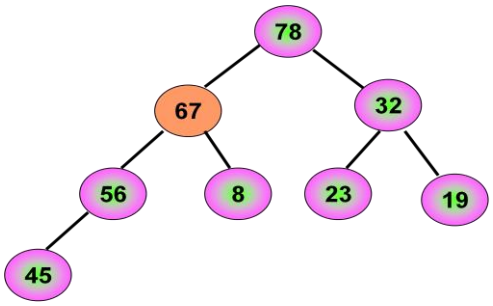


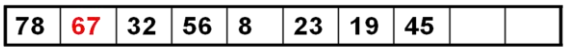
Before insertion (or) Before reheap – up





Step-3 Now the node 67 was placed correctly in its position.





Aftere insertion (or) After reheap – up

d) Reheap Down (Deletion):

ReheapDown repairs a "broken" heap by pushing the root down the tree until it is in its correct position in the heap [1].

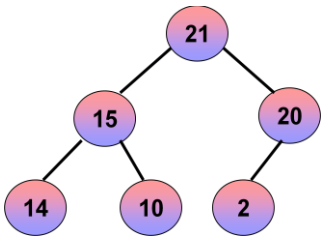
e) Deletion from a Max Heap:

Any node can be deleted from a heap tree. But from the application point of view, deleting the root node has some special importance [3].

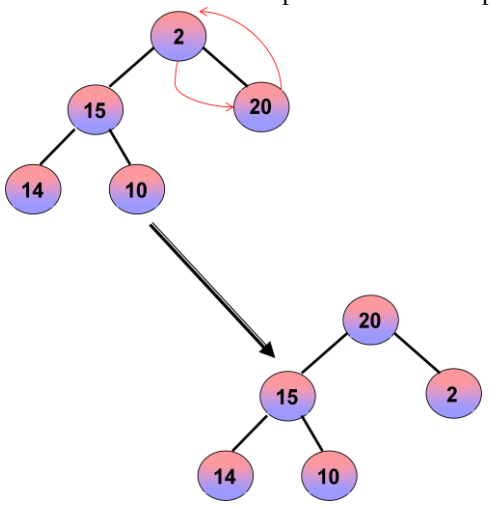
(a). Read the Root Node: When deleting a node from a heap, the most common and meaningful logic is to delete the root. The heap is thus left without a root.

(b). Replace Root node by the last element in the heap tree: To reestablish the heap, move the data in the last heap node to the root and reheapdown.

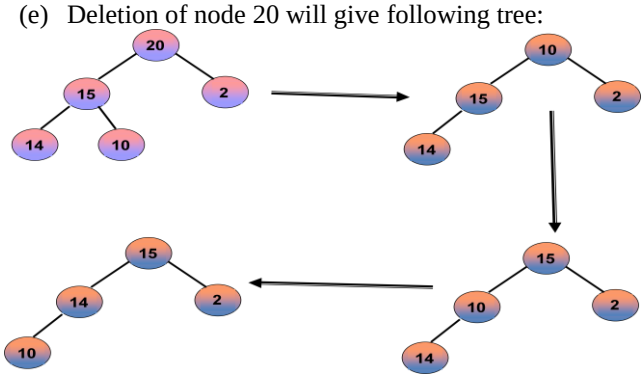
(c). When an element is to be removed from a Max heap, it's taken from the root of the heap. Suppose if we want to delete the element '21', the resulting tree will be as follow**:**



(d) After deleting the root node '21'. The leaf node '2' is placed in the root and reheapdown has to take place



(e) Deletion of node 20 will give following tree:



f) Algorithm for heap deletion:

algorithm deletetHeap (ref heap < array of datatype>,
ref last <index>,
delete root of heap & passes data
back to caller)

Precondition : heap is a valid heap structure,
last is index to last node in heap,

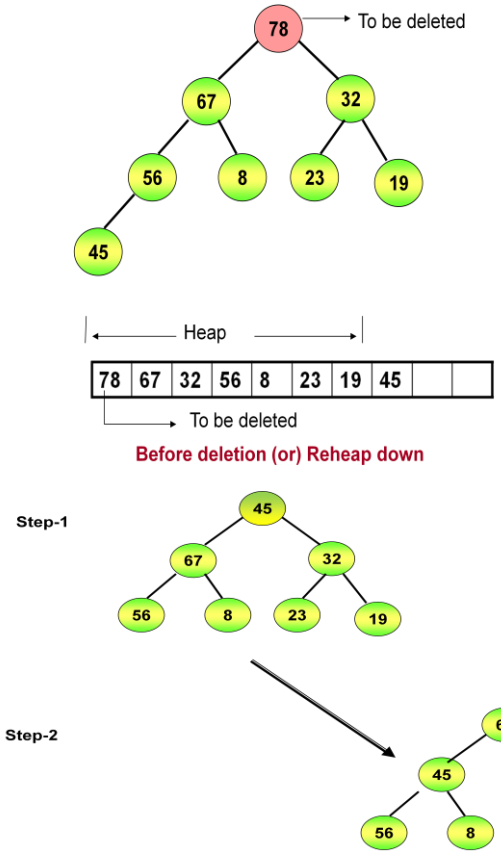
Postcondition : root has been deleted from heap
root data placed in dataout.

Return : True if successful ; False if array full;

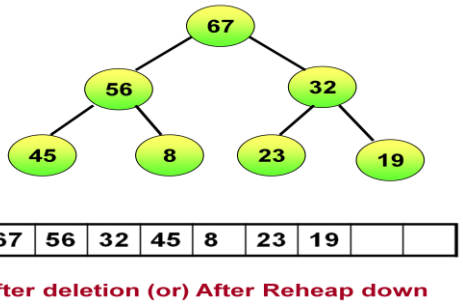
a. if (heap full)
b. return false
c. end if
d. dataout = heap [0]
e. heap [0] = heap [last]
f. reheapdown (heap, 0, last)
g. return true
End deleteHeap

Example:

Delete the node 78



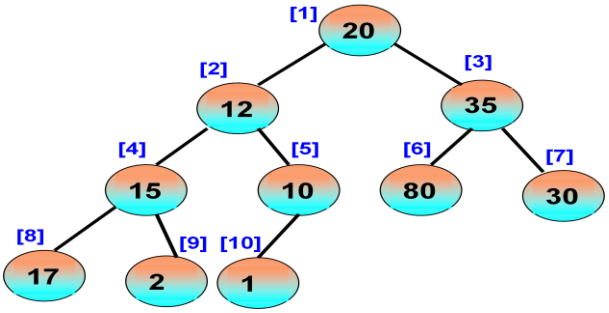
Step-3



V. MAX HEAP INITIALIZATION

a. Let us consider an array a[ ] with n-elements. Assume n=10 and the priority of the elements in a[10] is as follow:
a[20, 12, 35, 15, 10, 80, 30, 17, 2, 1]

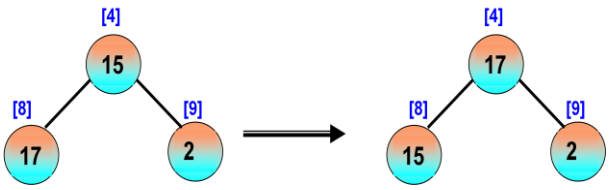
b. Suppose this array elements are arranged in a complete binary tree [2], as shown in the following fig.



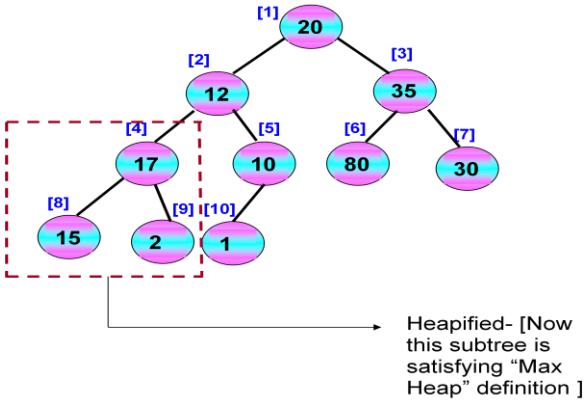
c. This complete binary tree is not a 'Max Heap', To heapify (make into a max heap) the above complete binary tree, use i = n/2 formula

Max heap is a max tree that is also a complete binary tree. But from the above figure, the positions [4] and [8] are not satisfying the definition of Max heap. So in this array position the heapified is going to take place. The position going to be heapified is [8], applying this in

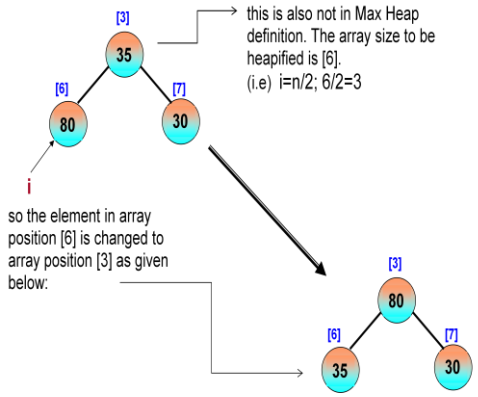
i = n/2; i = 8/2 = 4



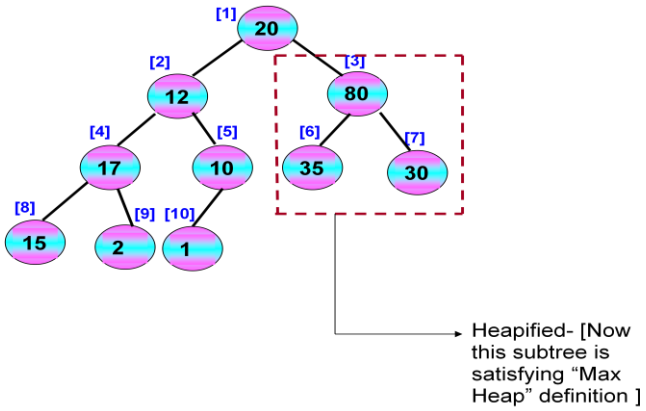
d. The heapified tree is as follow:



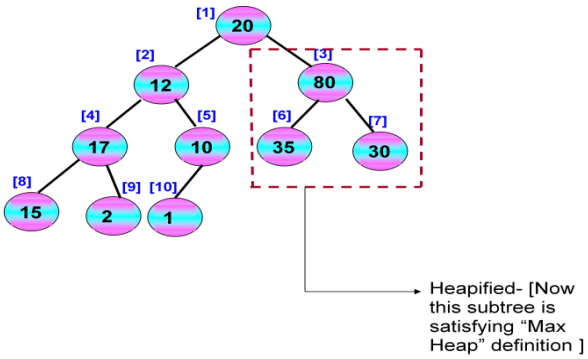
e. Now take the priority of [3,6 and 7] nodes to heapified



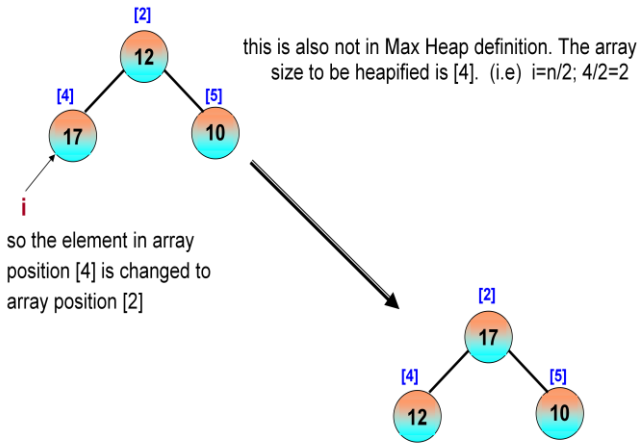
f. The heapified tree is as follow:



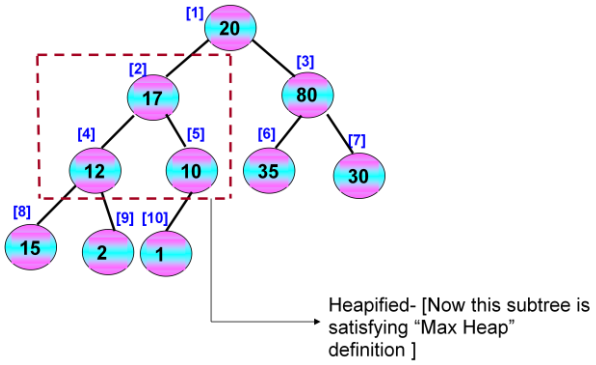
g. The heapified tree of the priority [3,6 and 7] is as follow:



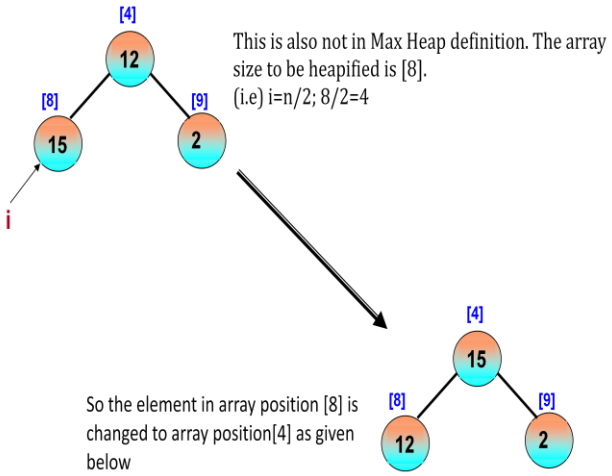
h. Now also, the above fig. is not a Max heap, because



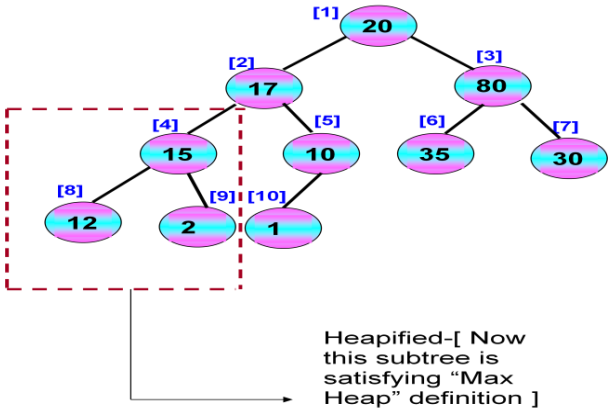
i. The heapified tree is as follow:



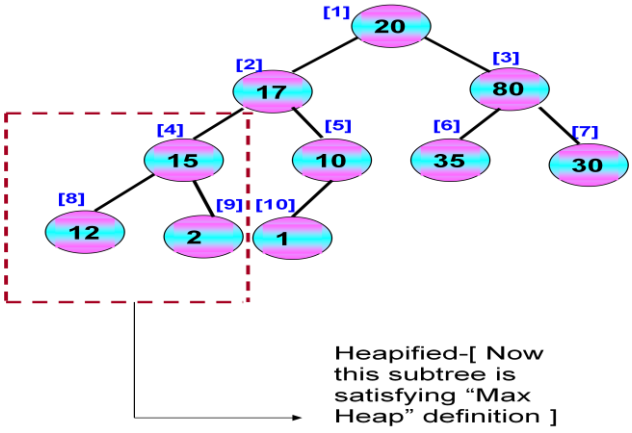
j. Now take



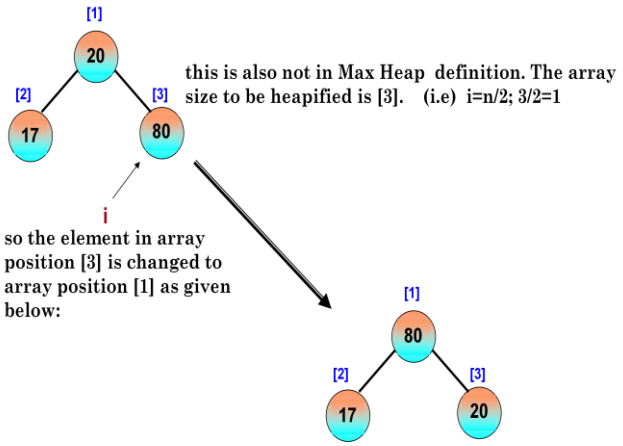
k. The heapified tree is as follow:



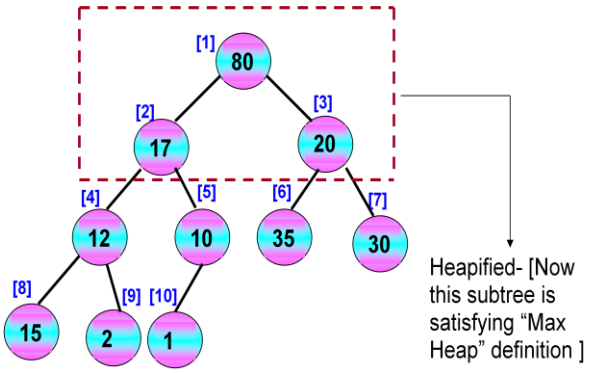
l. The heapified tree is as follow



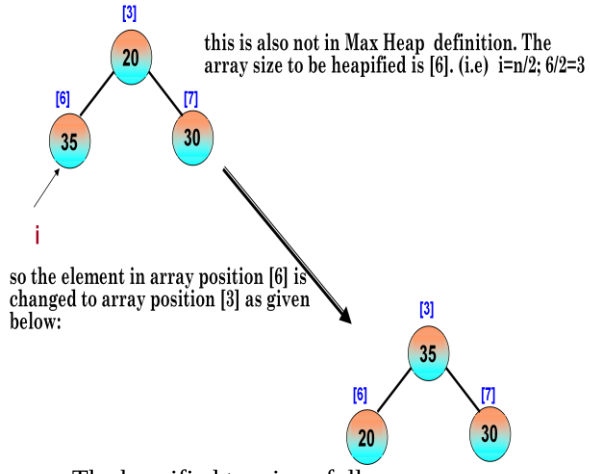
m. Still also the array position[3] to be heapified



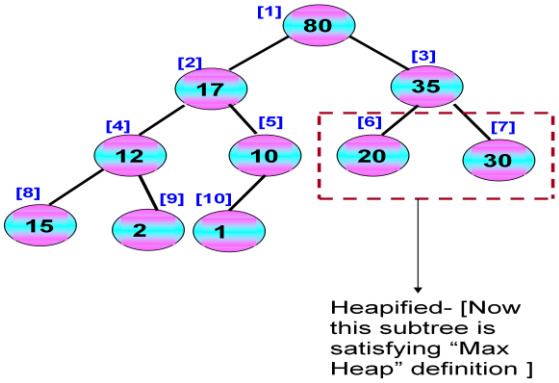
n. The heapified tree is as follow:



o. The array position[6] to be heapified



p. The heapified tree is as follow:



q. According to Max Heap – definition

a) the Root node 80 – is greater than it's children node – 17, 35
b) the Root node 17 – is greater than it's children node – 15, 10
c) the Root node 35 – is greater than it's children node – 20, 30
d) the Root node 15 – is greater than it's children node – 12, 2

VI. APPLICATIONS OF HEAP TREE

There are two main applications of heap trees [1]

a. ***Sorting using Heap Tree (Heap sort):*** The sorting method which is based on heap tree is called Heap Sort". And this is the efficient sorting method.

b. ***Priority Queue implementation using Heap Tree:*** Priority Queue can be implemented using circular array, Linked List. Another simplified implementation is possible using heap tree. The heap, however, can be represented using an array. This implementation is therefore free from the complexities of circular array and Linked List but getting the advantages of simplification of array [6].

a) Sorting using Heap Tree (Heap Sorting):

Any kind of data can be sorted either in ascending order or in descending order using heap tree. This actually comprises of the following steps:

Phase 1: Build a heap tree with the given set of data to sort the data in ascending /descending order, we have to built Max heap / Min heap in step-

Phase 2:(a) Delete the root node from the heap.
(b) Place the last leaf node in the root position.
(c) Rebuild the heap.
(d) Place the deleted node in the output.

Phase 3: Continue Step-2 until the heap tree is empty.

EXAMPLE

Sort the following set of data in Ascending Order

33, 14, 65, 2, 76, 69, 59, 85, 47, 99, 98

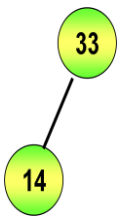
Phase 1

Build a heap tree with the given set of data

Step 1

Step 2

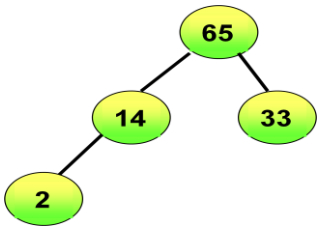
Insert the element '14'



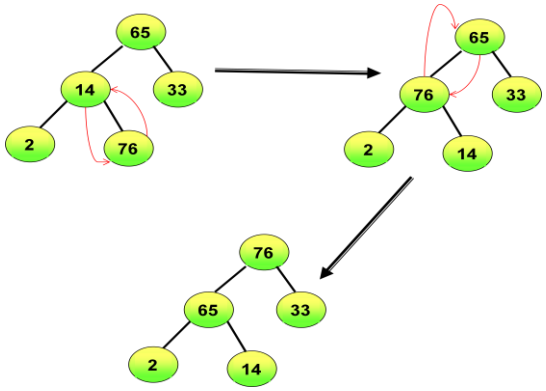
Step 3

Insert the element '65'

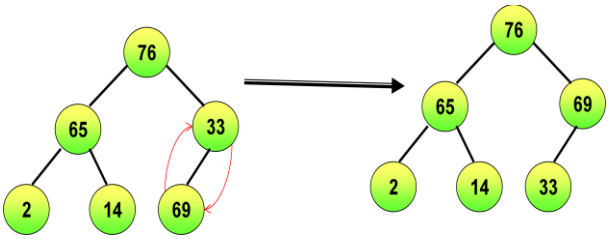
Step 4
Insert the element ‘2’



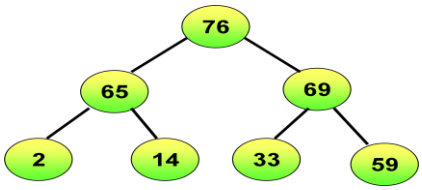
Step 5
Insert the element ’76’



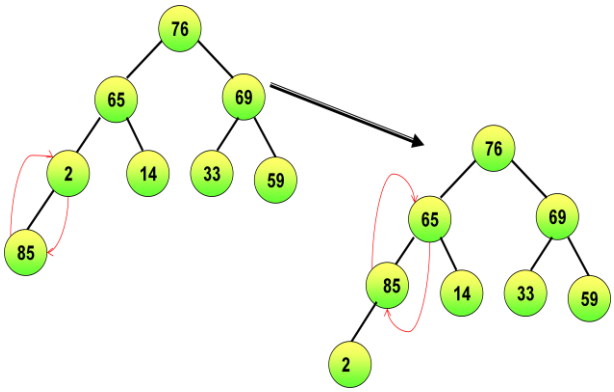
Step 6
Insert the element ‘69’

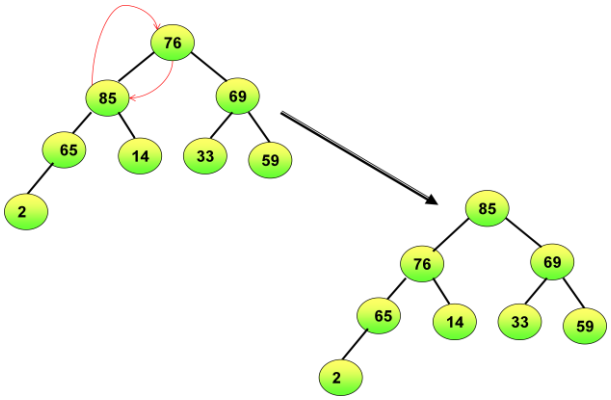


Step 7
Insert the element ‘59’

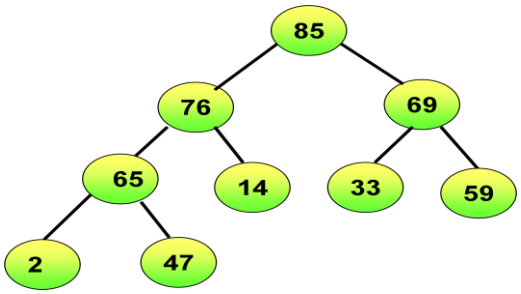


Step 8
Insert the element ‘85’

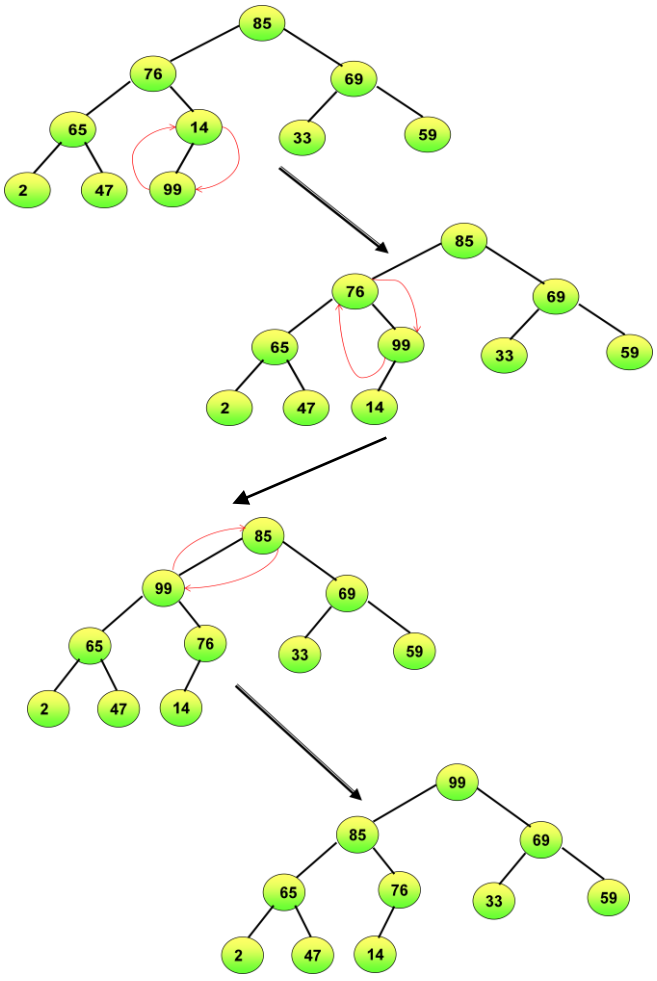




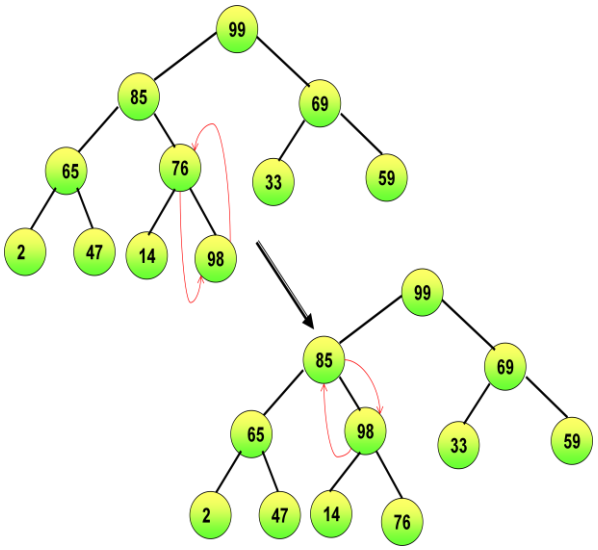
Step 9
Insert the element ’47’



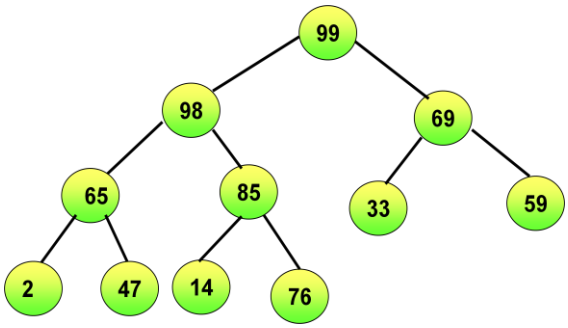
Step 10
Insert the element ’99’



Step 11
Insert the element ’98’



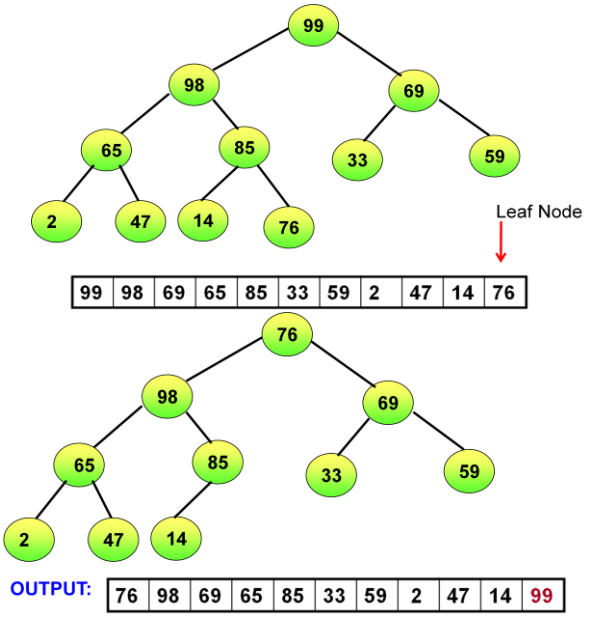
The construction of Heap tree for the given set of data was over. Now move to Phase 2, to carry out the Deletion and Rebuilding of data in the Heap tree



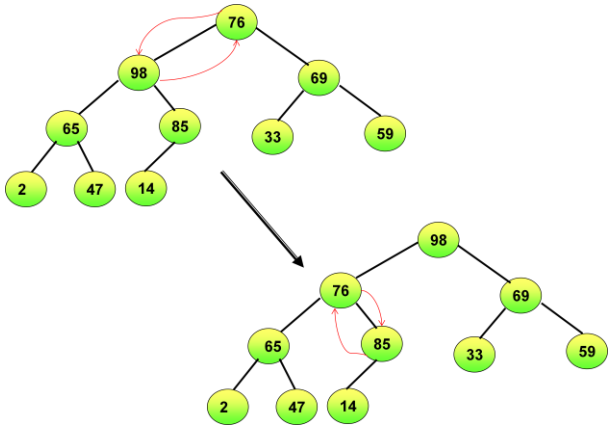
Phase 2: (a) Delete the root node from the heap.
(b) Place the last leaf node in the root position.
(c) Rebuild the heap.
(d) Place the deleted node in the output.

Step 1

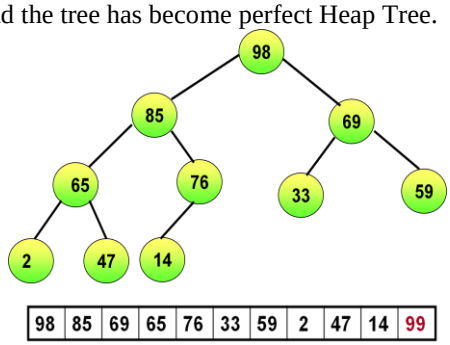
Delete the root node '99' and place it in the output, and place the last leaf node '76' in the root position



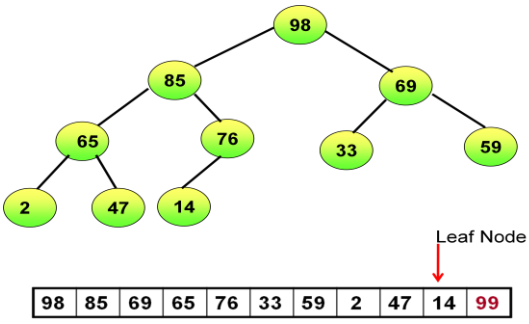
After swapping the root node and the last node, the next step is to rebuild the heap tree. Because after swapping the resultant tree is not a Heap tree, so rebuild the heap tree using "REHEAPDOWN" process as follow:



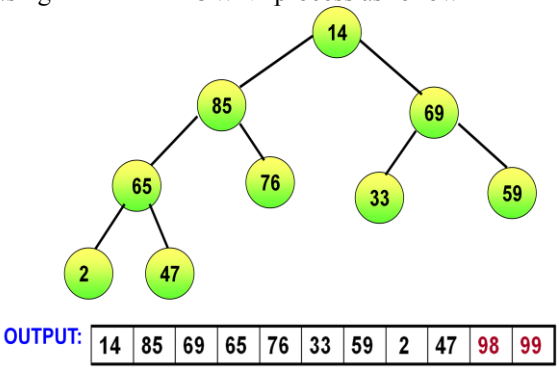
After Reheapdown the leaf node was placed in a correct place. And the tree has become perfect Heap Tree.

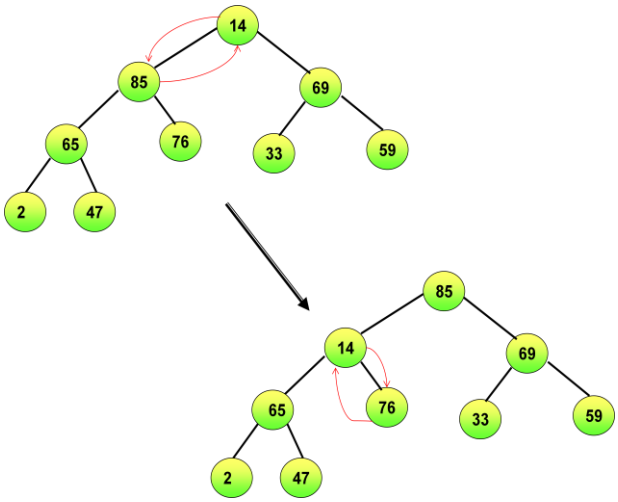


Step 2: Delete the root node '98' and place it in the output , and place the last leaf node '14' in the root position.

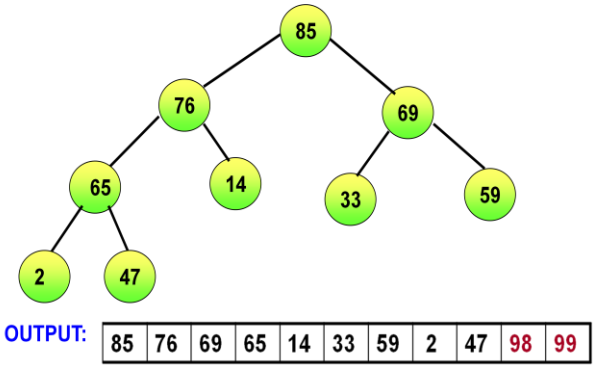


After swapping the root node and the last node, the next step is to rebuild the heap tree. Because after swapping the resultant tree is not a Heap tree, so rebuild the heap tree using "REHEAPDOWN" process as follow



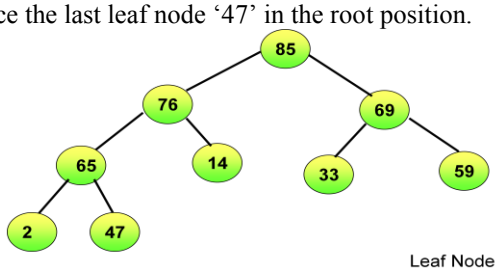


After Reheapdown the leaf node was placed in a correct place. And the tree has become perfect Heap Tree.



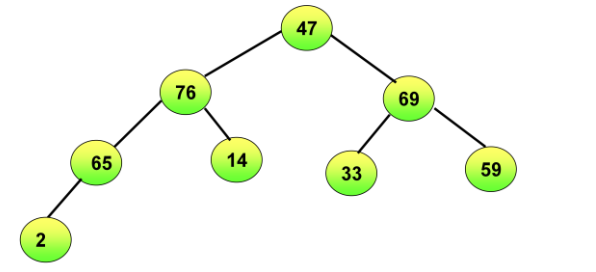
OUTPUT:	85	76	69	65	14	33	59	2	47	98	99

Step 3: Delete the root node '85' and place it in the output and place the last leaf node '47' in the root position.

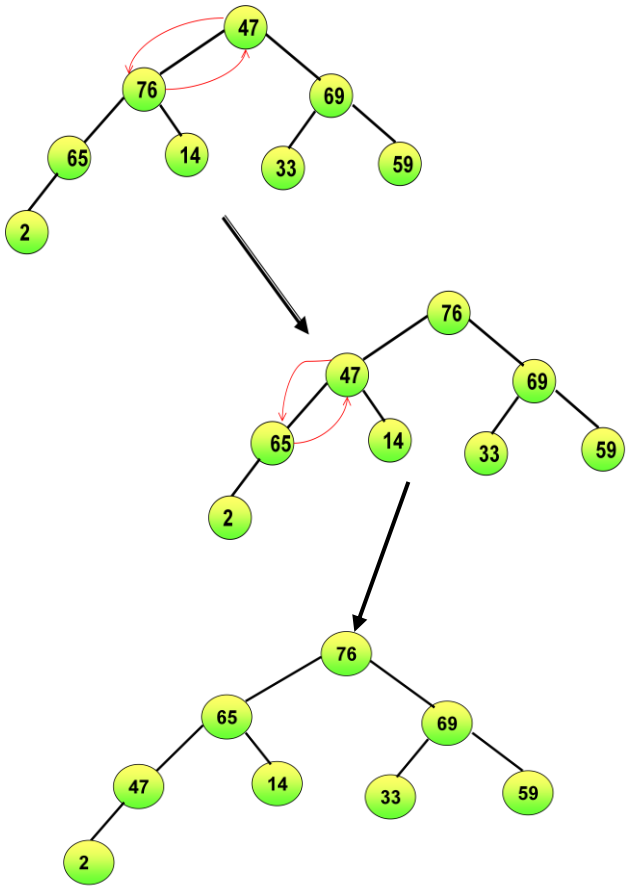


OUTPUT:	85	76	69	65	14	33	59	2	47	98	99

After swapping the root node and the last node, the next step is to rebuild the heap tree. Because after swapping the resultant tree is not a Heap tree, so rebuild the heap tree using "REHEAPDOWN" process as follow:



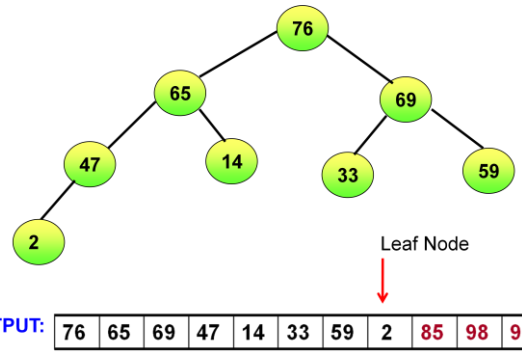
OUTPUT:	47	76	69	65	14	33	59	2	85	98	99



OUTPUT:	76	65	69	47	14	33	59	2	85	98	99

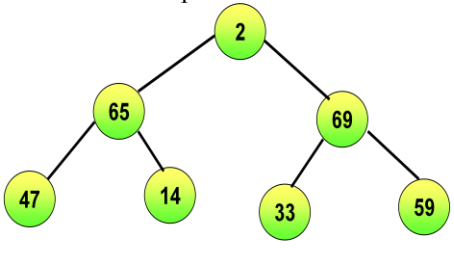
After Reheapdown the leaf node was placed in a correct place. And the tree has become perfect Heap Tree.

Step 4: Delete the root node '76' and place it in the output, and place the last leaf node '2' in the root position.

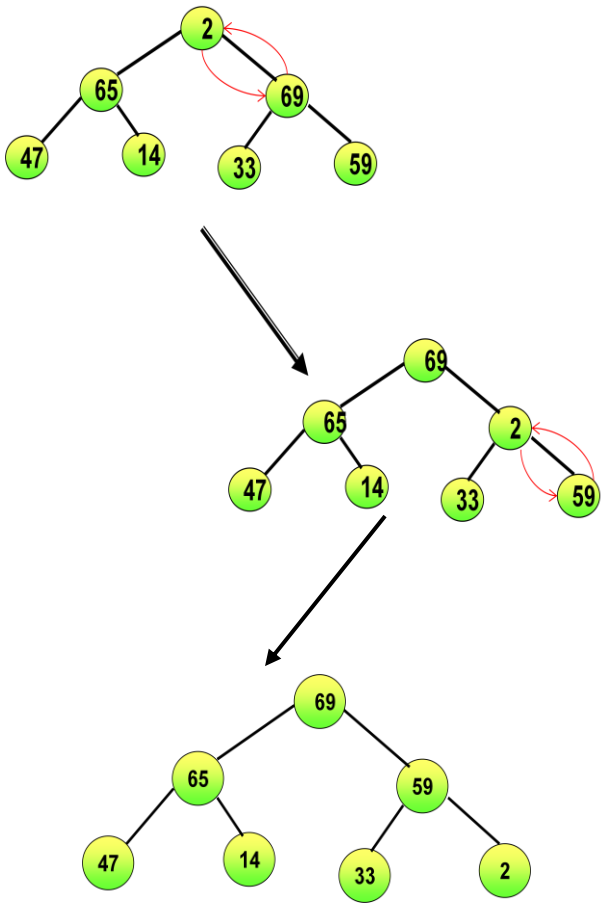


OUTPUT:	76	65	69	47	14	33	59	2	85	98	99

After swapping the root node and the last node, the next step is to rebuild the heap tree. Because after swapping the resultant tree is not a Heap tree, so rebuild the heap tree using "REHEAPDOWN" process as follow:



OUTPUT:	2	65	69	47	14	33	59	76	85	98	99

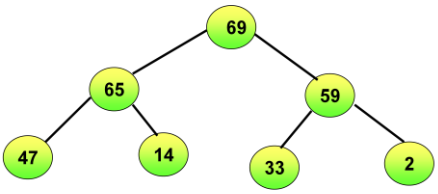


After Reheapdown the leaf node was placed in a correct place. And the tree has become perfect Heap Tree.

OUTPUT:

69	65	59	47	14	33	2	76	85	98	99

Step 5: Delete the root node '69' and place it in the output, and place the last leaf node '2' in the root position.

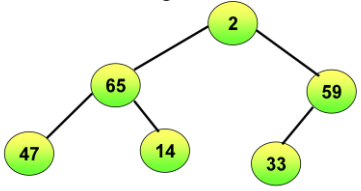


Leaf Node

OUTPUT:

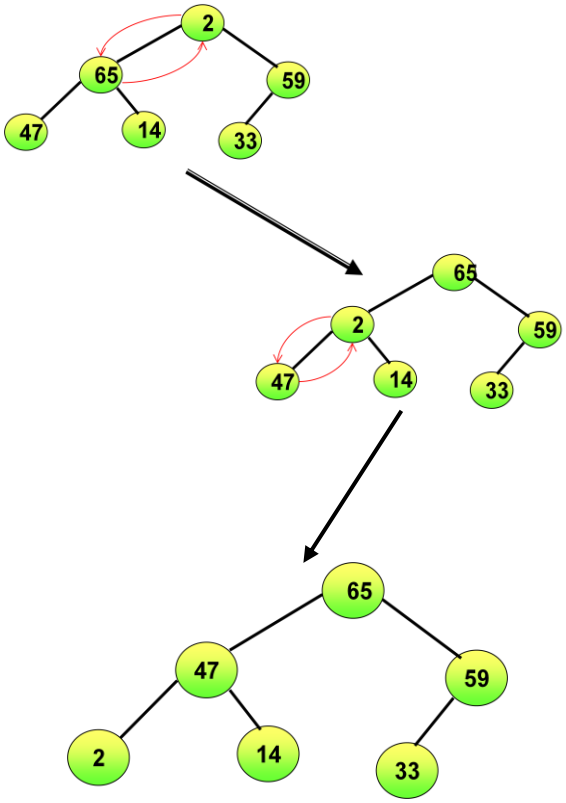
69	65	59	47	14	33	2	76	85	98	99

After swapping the root node and the last node, the next step is to rebuild the heap tree. Because after swapping the resultant tree is not a Heap tree, so rebuild the heap tree using "REHEAPDOWN" process as follow:



OUTPUT:

2	65	59	47	14	33	69	76	85	98	99

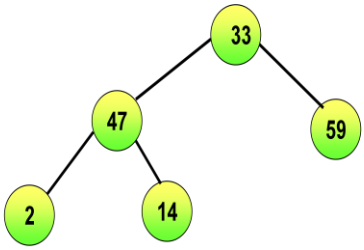


After Reheapdown the leaf node was placed in a correct place. And the tree has become perfect Heap Tree.

OUTPUT:

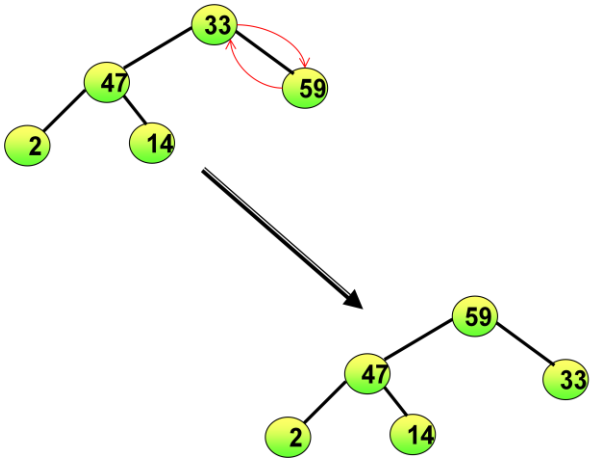
65	47	59	2	14	33	69	76	85	98	99

After swapping the root node and the last node, the next step is to rebuild the heap tree. Because after swapping the resultant tree is not a Heap tree, so rebuild the heap tree using "REHEAPDOWN" process as follow:

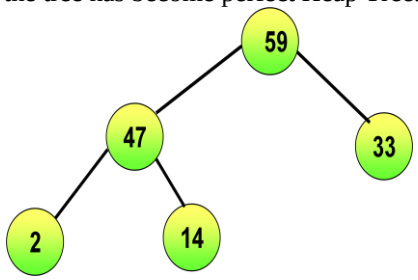


OUTPUT:

33	47	59	2	14	65	69	76	85	98	99



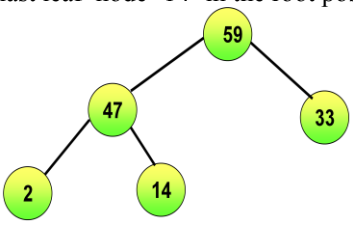
After Reheapdown the leaf node was placed in a correct place. And the tree has become perfect Heap Tree.



OUTPUT:

59	47	33	2	14	65	69	76	85	98	99

Step 7: Delete the root node '59 and place it in the output, and place the last leaf node '14' in the root position.

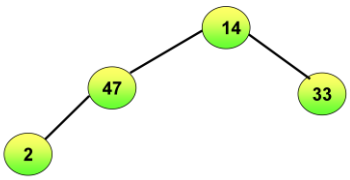


Leaf Node

OUTPUT:

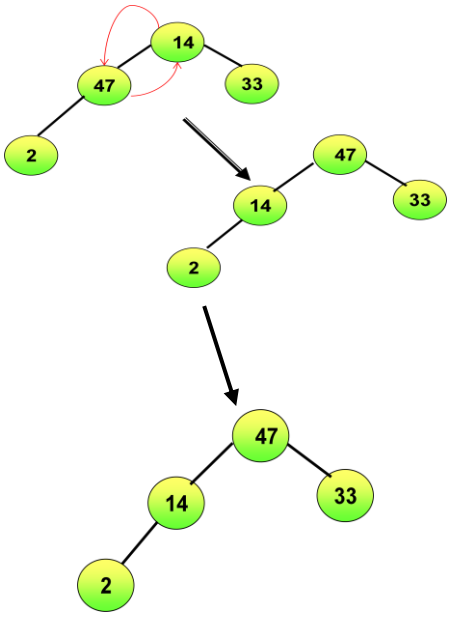
59	47	33	2	14	65	69	76	85	98	99

After swapping the root node and the last node, the next step is to rebuild the heap tree. Because after swapping the resultant tree is not a Heap tree, so rebuild the heap tree using "REHEAPDOWN" process as follow:



OUTPUT:

14	47	33	2	59	65	69	76	85	98	99

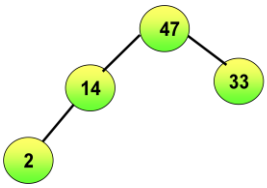


After Reheapdown the leaf node was placed in a correct place. And the tree has become perfect Heap Tree.

OUTPUT:

47	14	33	2	59	65	69	76	85	98	99

Step 8: Delete the root node '47' and place it in the output, and place the last leaf node '2' in the root position.

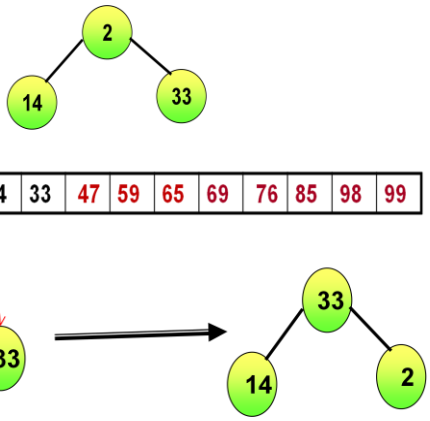


Leaf Node

OUTPUT:

47	14	33	2	59	65	69	76	85	98	99

After swapping the root node and the last node, the next step is to rebuild the heap tree. Because after swapping the resultant tree is not a Heap tree, so rebuild the heap tree using "REHEAPDOWN" process as follow:

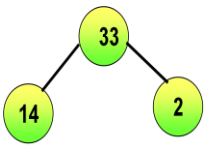


After Reheapdown the leaf node was placed in a correct place. And the tree has become perfect Heap Tree.

OUTPUT:

33	14	2	47	59	65	69	76	85	98	99

Step 9: Delete the root node '33' and place it in the output, and place the last leaf node '2' in the root position.

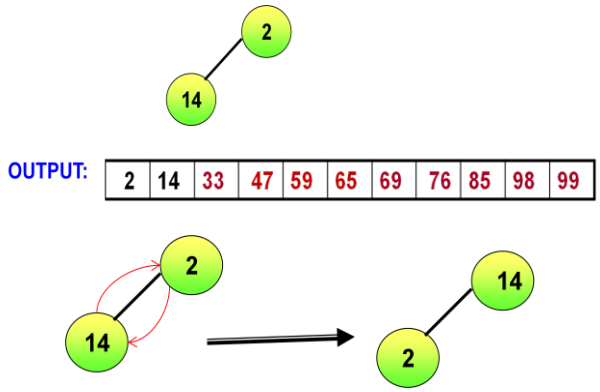


Leaf Node

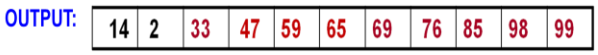
OUTPUT:

33	14	2	47	59	65	69	76	85	98	99

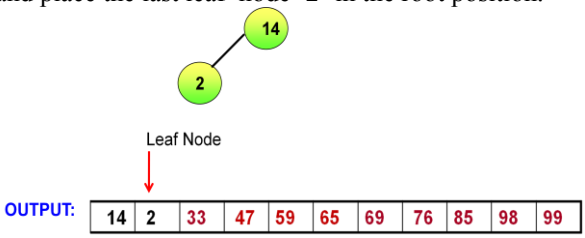
After swapping the root node and the last node, the next step is to rebuild the heap tree. Because after swapping the resultant tree is not a Heap tree, so rebuild the heap tree using "REHEAPDOWN" process as follow:



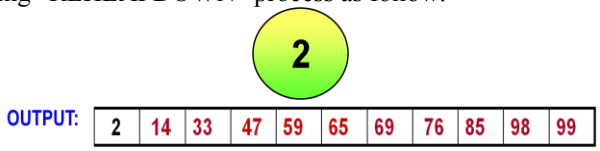
After Reheapdown the leaf node was placed in a correct place. And the tree has become perfect Heap Tree.



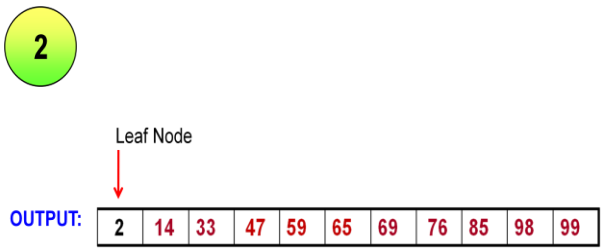
Step10: Delete the root node '14' and place it in the output, and place the last leaf node '2' in the root position.



After swapping the root node and the last node, the next step is to rebuild the heap tree. Because after swapping the resultant tree is not a Heap tree, so rebuild the heap tree using "REHEAPDOWN" process as follow:

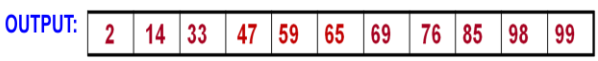


Step11: The remaining node is '2' place it in the output.



Phase 3

Continue Step-2 until the heap tree is empty. After placing the node '2' in the output, the Heap tree has become empty. The sorted data is placed in the output as follow:



c. Priority Queue implementation using Heap Tree:

In a multi-user environment the operating scheduler decides which of the several processes to run. Generally the processes are allowed to run for a fixed amount of time. One algorithm uses a queue. Jobs are initially placed at the end of the queue. The Scheduler will always take the first job in the queue. It will run the job till it finishers or till its time limit is up. This strategy is generally not appropriate because for some jobs the waiting time may be more than its actual processing time. These jobs must have certain priority over other jobs. This type of application requires a special type of queue called 'Priority Queue'. Another name for Heap is Priority Queue.The priority queue has two basic operations:

a) Insert- which is similar to Enqueue – to insert an element in the queue.

b) Delete-which is similar to Dequeue – deletes the data with highest priority [4].

Elements associated with their priority values are to be stored in the form of heap tree, which can be formed based on their priority values.The top priority element that has to be processed first is at the root. So, it can be deleted and heap can be rebuilt to get the next element to be processed.

Example:

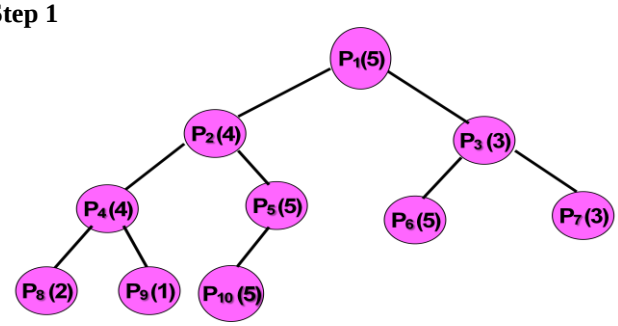
Consider the following processes with their priorities:

process	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Priority	5	4	3	4	5	5	3	2	1	5

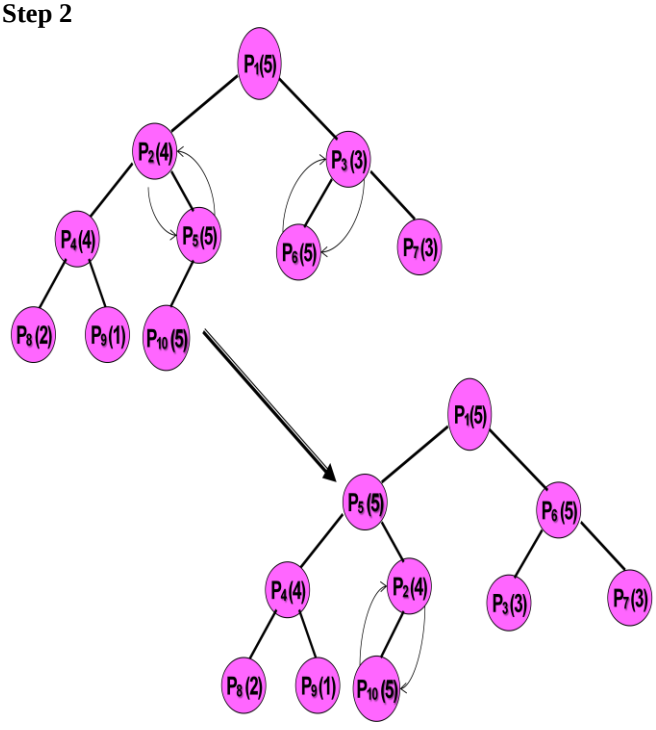
1st Top Priority element and process : 5(p1,p5,p6,p10)
2nd Top Priority element and process : 4 (p2,p4)
3rd Top Priority element and process : 3 (p3,p7)
4th Top Priority element and process : 2 (p8)
5th Top Priority element and process : 1 (p9)

Construction of Heap Tree

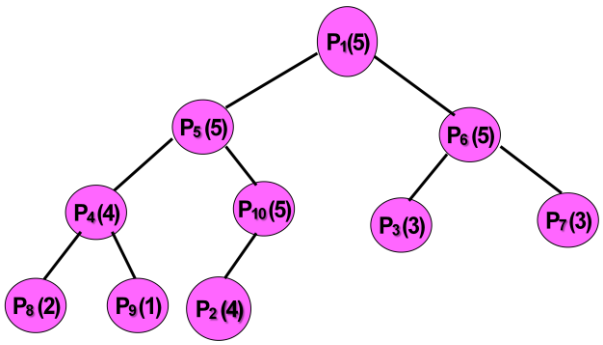
Step 1



Step 2



Step 3



VII. CONCLUSION

The concept of a heap is simple, but the actual implementation can appear tricky, How do you remove the root node and still ensure that it is eventually replaced by the correct node? How do you add a new node to a heap and ensure that it is moved into the proper spot. Heap sort is relatively simple algorithm built upon the heap data structure. Heap sort has guaranteed O(n*log(n))) performance, though the constant factor is typically a bit higher than for other algorithms such as quick sort. Heap sort is not a stable sort, so the original ordering of equal elements may not be maintained.

VIII. ACKNOWLEDGEMENT

I would like to thank my parents and my brother for their support for completing this journal. Last, and most obvious but not least, I thank the IJARCS for their valuable guidance to correct my article.

IX. REFERENCES

[1]. Richard F.Gilberg, Behrouz A.Forouzan, Data Structures – A pseudocode Approach with C++2nd Edition, 2005, Thomson-Books/Cole, pp. 407-434.

[2]. Sartaj Sahni – Data Structures, Algorithms and Applications in C++, 2nd Edition, 2005, Universities Press(India) Private Limited,pp. 464-491.

[3]. Samanta Debasis – Classic Data Structure, 2nd Edition, 2009, Prentice Hall, pp. 266-272.

[4]. A.A.Puntambekar, N.A.Despande, S.S.Sane-Data Structures,2006, Technical Publications, pp. 436-445.

[5]. Kushwaha Dharmender Singh, Misra Arun Kumar- entice Hall Learning Private Limited,pp. 560- 575.

[6]. AlexAllain, www.cprogramming.com/tutorial/ Computer sicencetheory/heapsort.html.