

[Home](#)

[DAA](#)

[DS](#)

[DBMS](#)

[Aptitude](#)

[Selenium](#)

[Kotlin](#)

[C#](#)

[HTML](#)

[CSS](#)

[JavaScript](#)

Heap Sort

Binary Heap:

Binary Heap is an array object can be viewed as Complete Binary Tree. Each node of the Binary Tree corresponds to an element in an array.

1. Length [A], number of elements in array
2. Heap-Size[A], number of elements in a heap stored within array A.

The root of tree A [1] and gives index 'i' of a node that indices of its parents, left child, and the right child can be computed.

PARENT (i)

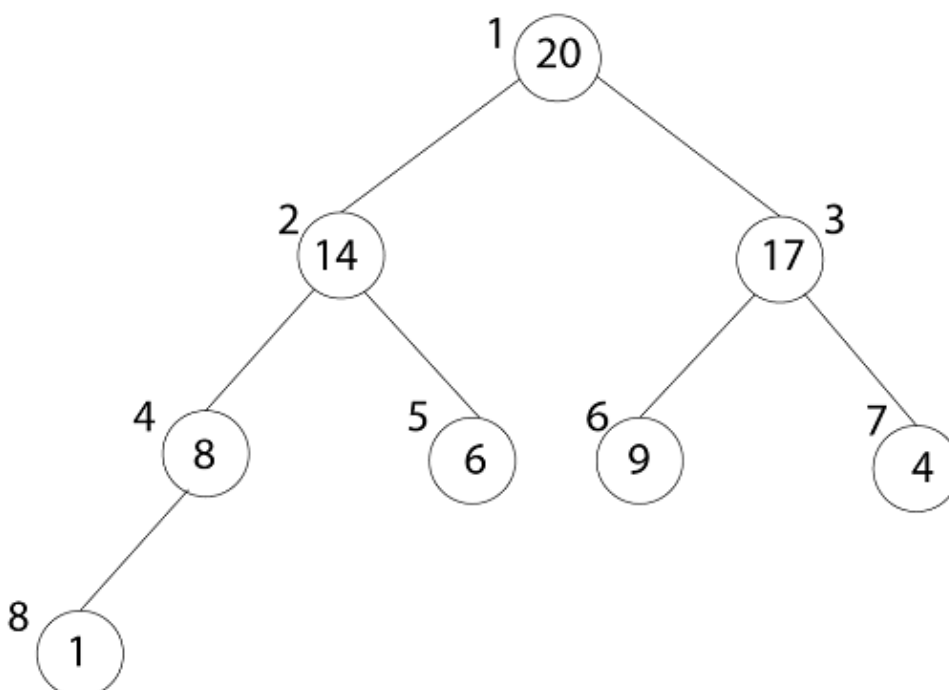
Return floor ($i/2$)

LEFT (i)

Return $2i$

RIGHT (i)

Return $2i+1$



Representation of an array of the above figure is given below:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|----|----|----|---|---|---|---|---|---|
| 20 | 14 | 17 | 8 | 6 | 9 | 4 | 1 | |

The index of 20 is 1

To find the index of the left child, we calculate $1*2=2$

This takes us (correctly) to the 14.

Now, we go right, so we calculate $2*2+1=5$

This takes us (again, correctly) to the 6.

Now, 4's index is 7, we want to go to the parent, so we calculate $7/2 = 3$ which takes us to the 17.

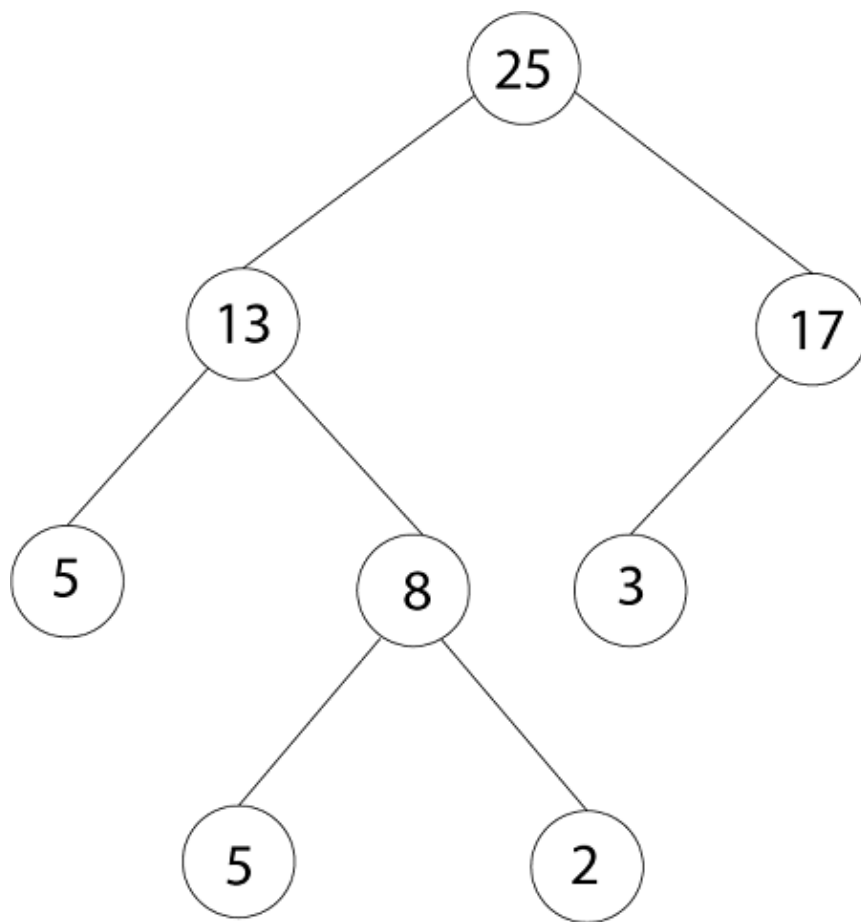
Heap Property:

A binary heap can be classified as Max Heap or Min Heap

1. Max Heap: In a Binary Heap, for every node i other than the root, the value of the node is greater than or equal to the value of its highest child

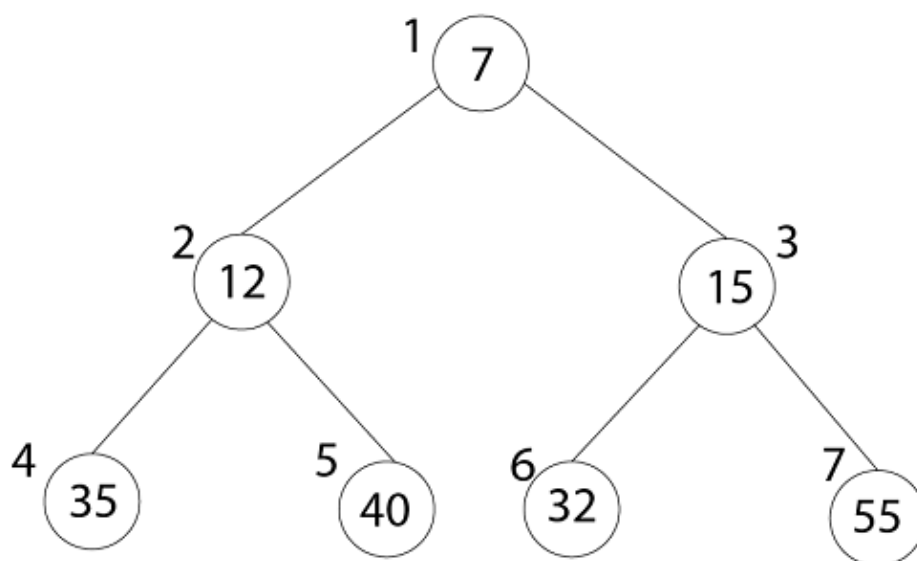
$$A[\text{PARENT}(i)] \geq A[i]$$

Thus, the highest element in a heap is stored at the root. Following is an example of MAX-HEAP



2. MIN-HEAP: In MIN-HEAP, the value of the node is lesser than or equal to the value of its lowest child.

$A[\text{PARENT}(i)] \leq A[i]$



Heapify Method:

1. Maintaining the Heap Property: Heapify is a procedure for manipulating heap Data Structure. It is given an array A and index I into the array. The subtree rooted at the children of A [i] are heap but node A [i] itself may probably violate the heap property i.e. $A[i] < A[2i]$ or $A[2i+1]$. The procedure 'Heapify' manipulates the tree rooted as A [i] so it becomes a heap.

MAX-HEAPIFY (A, i)

1. $l \leftarrow \text{left}[i]$
2. $r \leftarrow \text{right}[i]$
3. if $l \leq \text{heap-size}[A]$ and $A[l] > A[i]$
4. then $\text{largest} \leftarrow l$
5. Else $\text{largest} \leftarrow i$
6. If $r \leq \text{heap-size}[A]$ and $A[r] > A[\text{largest}]$
7. Then $\text{largest} \leftarrow r$
8. If $\text{largest} \neq i$
9. Then exchange $A[i]$ $A[\text{largest}]$
10. MAX-HEAPIFY (A, largest)

Analysis:

The maximum levels an element could move up are $\Theta(\log n)$ levels. At each level, we do simple comparison which $O(1)$. The total time for heapify is thus $O(\log n)$.

Building a Heap:

```
BUILDHEAP (array A, int n)
1 for i  $\leftarrow$  n/2 down to 1
2 do
3 HEAPIFY (A, i, n)
```

HEAP-SORT ALGORITHM:

HEAP-SORT (A)

1. BUILD-MAX-HEAP (A)
2. For $I \leftarrow \text{length}[A]$ down to 2
3. Do exchange $A[1] \leftrightarrow A[i]$

4. Heap-size [A] \leftarrow heap-size [A]-1
5. MAX-HEAPIFY (A,1)

Analysis: Build max-heap takes $O(n)$ running time. The Heap Sort algorithm makes a call to 'Build Max-Heap' which we take $O(n)$ time & each of the $(n-1)$ calls to Max-heap to fix up a new heap. We know 'Max-Heapify' takes time $O(\log n)$

The total running time of Heap-Sort is $O(n \log n)$.

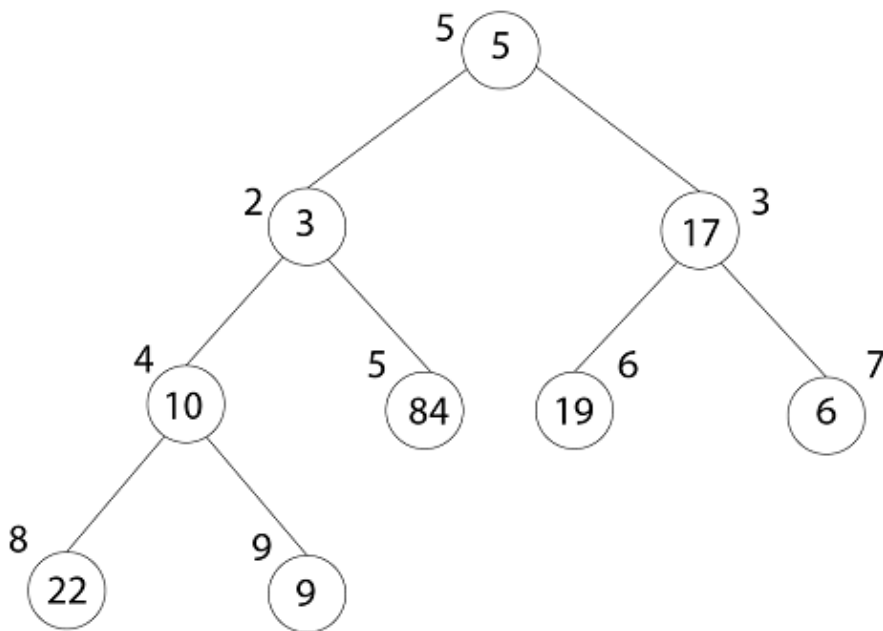
Example: Illustrate the Operation of BUILD-MAX-HEAP on the array.

A = (5, 3, 17, 10, 84, 19, 6, 22, 9)

Solution: Originally:

Heap-Size (A) = 9, so first we call MAX-HEAPIFY (A, 4)

And $I = \frac{4.5}{2} = 2$ to 1



After MAX-HEAPIFY (A, 4) and $i=4$

$L \leftarrow 8, r \leftarrow 9$

$L \leq \text{heap-size}[A]$ and $A[L] > A[i]$

$8 \leq 9$ and $22 > 10$

Then Largest $\leftarrow 8$

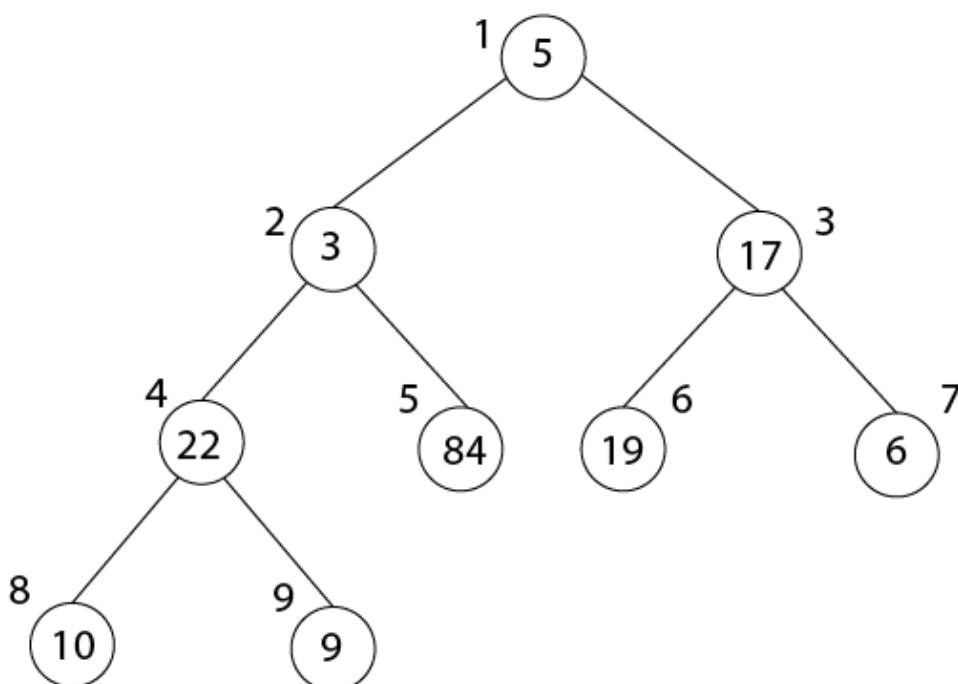
If $r \leq \text{heap-size}[A]$ and $A[r] > A[\text{largest}]$

$9 \leq 9$ and $9 > 22$

If largest (8) $\neq 4$

Then exchange $A[4] \leftrightarrow A[8]$

MAX-HEAPIFY (A, 8)



After MAX-HEAPIFY (A, 3) and $i=3$

$l \leftarrow 6, r \leftarrow 7$

$l \leq \text{heap-size}[A]$ and $A[l] > A[i]$

$6 \leq 9$ and $19 > 17$

$\text{Largest} \leftarrow 6$

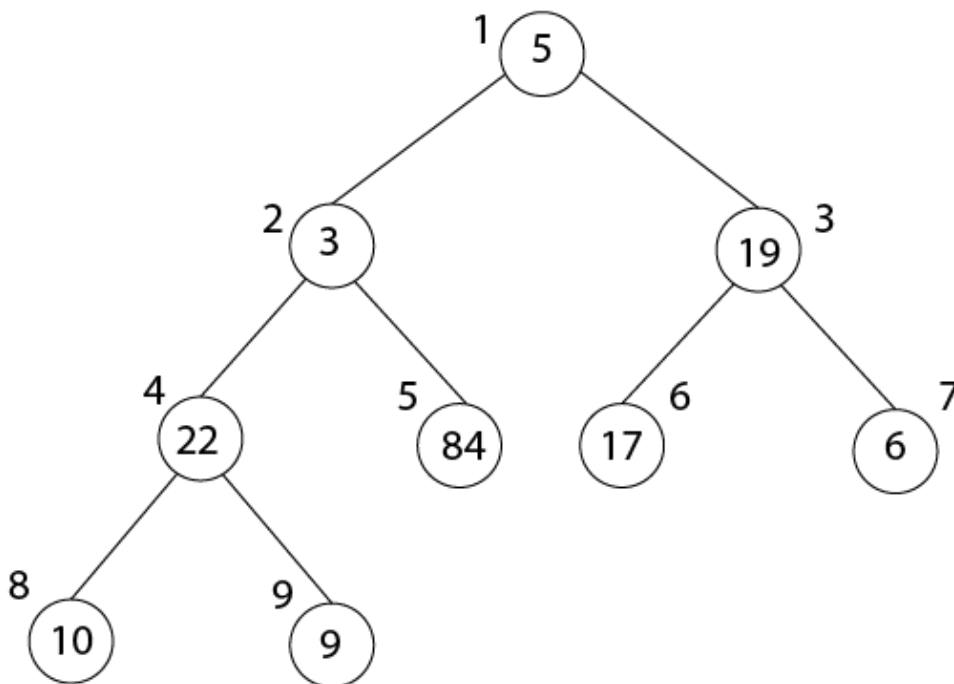
If $r \leq \text{heap-size}[A]$ and $A[r] > A[\text{largest}]$

$7 \leq 9$ and $6 > 19$

If $\text{largest} (6) \neq 3$

Then Exchange $A[3] \leftrightarrow A[6]$

MAX-HEAPIFY (A, 6)



After MAX-HEAPIFY (A, 2) and $i=2$

$l \leftarrow 4, r \leftarrow 5$

$l \leq \text{heap-size}[A]$ and $A[l] > A[i]$

$4 \leq 9$ and $22 > 3$

$\text{Largest} \leftarrow 4$

If $r \leq \text{heap-size}[A]$ and $A[r] > A[\text{largest}]$

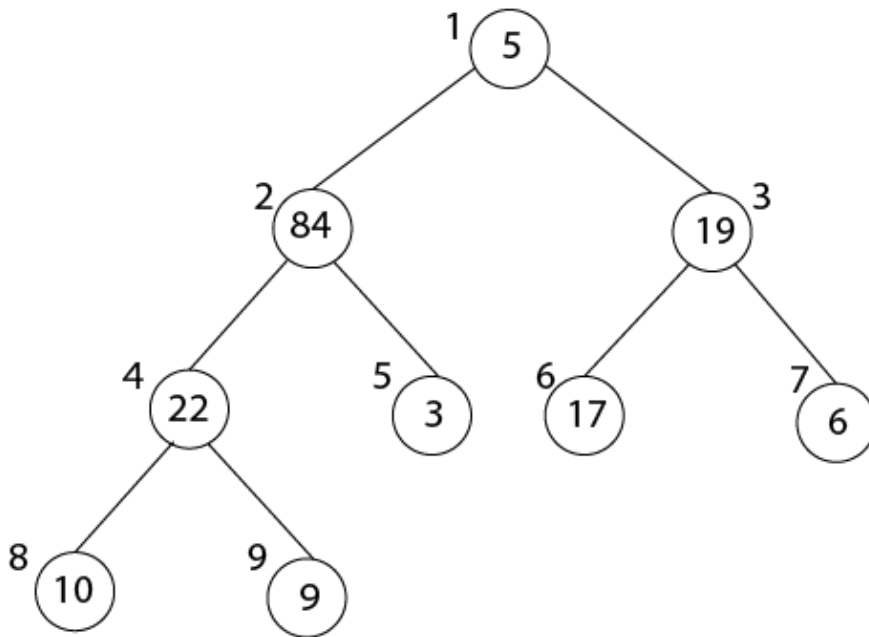
$5 \leq 9$ and $84 > 22$

$\text{Largest} \leftarrow 5$

If $\text{largest} (4) \neq 2$

Then Exchange $A[2] \leftrightarrow A[5]$

MAX-HEAPIFY (A, 5)



After MAX-HEAPIFY (A, 1) and $i=1$

$l \leftarrow 2, r \leftarrow 3$

$l \leq \text{heap-size}[A]$ and $A[l] > A[i]$

$2 \leq 9$ and $84 > 5$

$\text{largest} \leftarrow 2$

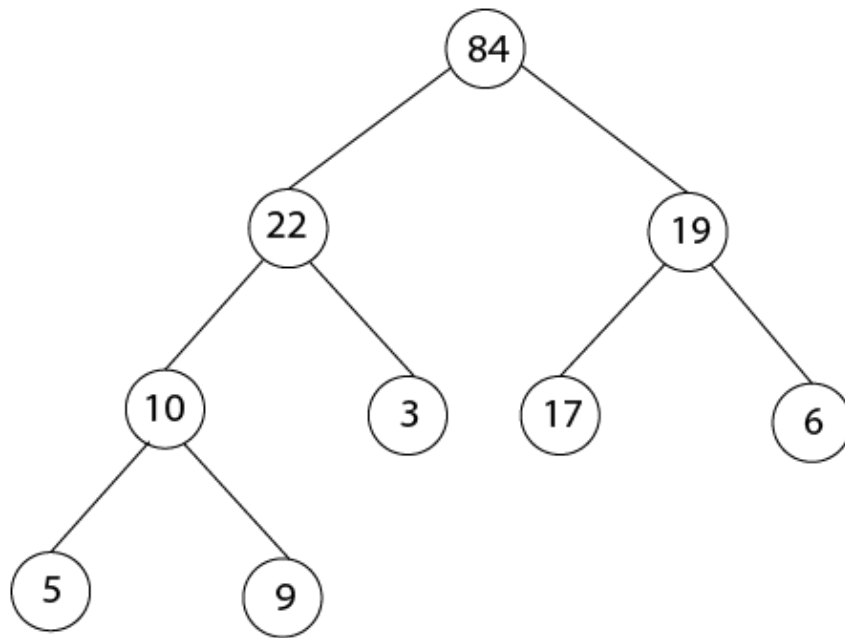
If $r \leq \text{heap-size}[A]$ and $A[r] > A[\text{largest}]$

$3 \leq 9$ and $19 < 84$

If $\text{largest} (2) \neq 1$

Then Exchange $A[1] \leftrightarrow A[2]$

MAX-HEAPIFY (A, 2)



Priority Queue:

As with heaps, priority queues appear in two forms: max-priority queue and min-priority queue.

A priority queue is a data structure for maintaining a set S of elements, each with a combined value called a key. A max-priority queue guides the following operations:

INSERT(S, x): inserts the element x into the set S , which is proportionate to the operation $S = S \cup \{x\}$.

MAXIMUM (S) returns the element of S with the highest key.

EXTRACT-MAX (S) removes and returns the element of S with the highest key.

INCREASE-KEY(S, x, k) increases the value of element x's key to the new value k, which is considered to be at least as large as x's current key value.

Let us discuss how to implement the operations of a max-priority queue. The procedure **HEAP-MAXIMUM** consider the **MAXIMUM** operation in $\Theta(1)$ time.

HEAP-MAXIMUM (A)

1. return A [1]

The procedure **HEAP-EXTRACT-MAX** implements the **EXTRACT-MAX** operation. It is similar to the for loop of **Heap-Sort** procedure.

HEAP-EXTRACT-MAX (A)

```
1 if A. heap-size < 1
2 error "heap underflow"
3 max ← A [1]
4 A [1] ← A [heap-size [A]]
5 heap-size [A] ← heap-size [A]-1
6 MAX-HEAPIFY (A, 1)
7 return max
```

The procedure **HEAP-INCREASE-KEY** implements the **INCREASE-KEY** operation. An index i into the array identify the priority-queue element whose key we wish to increase.

HEAP-INCREASE-KEY(A, i, key)

```
1 if key < A[i]
2 errors "new key is smaller than current key"
3 A[i] = key
4 while i>1 and A [Parent (i)] < A[i]
5 exchange A [i] with A [Parent (i)]
6 i =Parent [i]
```

The running time of HEAP-INCREASE-KEY on an n -element heap is $O(\log n)$ since the path traced from the node updated in line 3 to the root has length $O(\log n)$.

The procedure MAX-HEAP-INSERT implements the INSERT operation. It takes as an input the key of the new item to be inserted into max-heap A . The procedure first expands the max-heap by calculating to the tree a new leaf whose key is $-\infty$. Then it calls HEAP-INCREASE-KEY to set the key of this new node to its right value and maintain the max-heap property

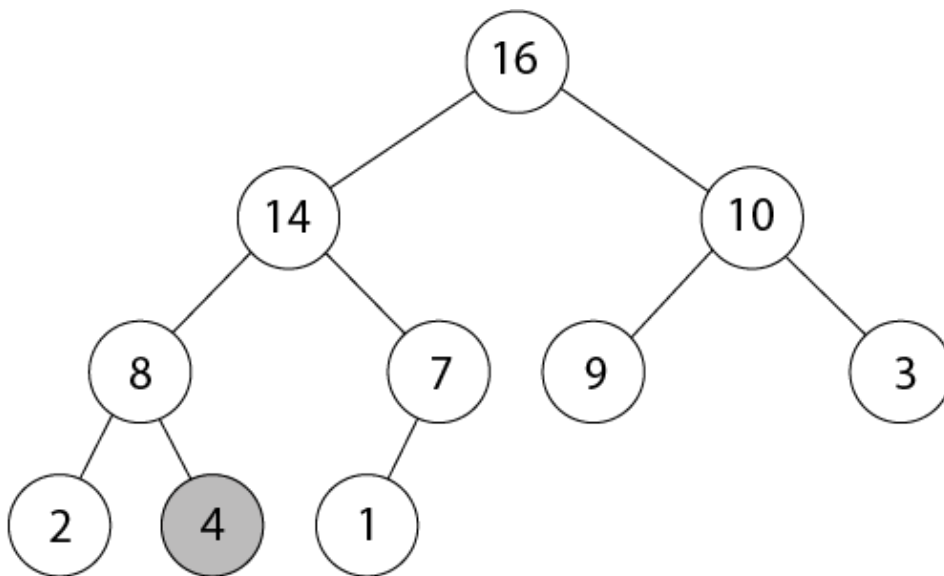
MAX-HEAP-INSERT (A, key)

```
1 A. heap-size = A. heap-size + 1
2 A [A. heap-size] = -  $\infty$ 
3 HEAP-INCREASE-KEY (A, A. heap-size, key)
```

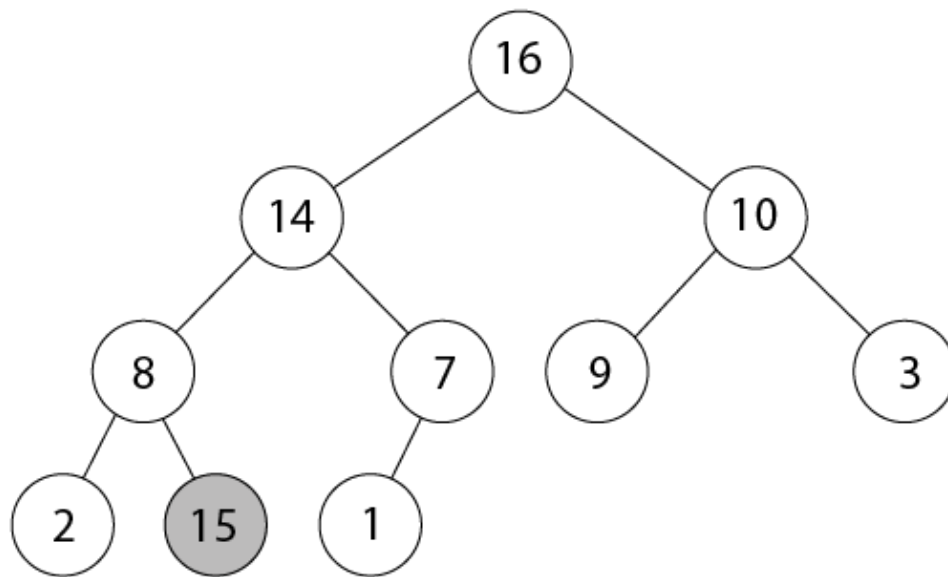
The running time of MAX-HEAP-INSERT on an n -element heap is $O(\log n)$.

Example: Illustrate the operation of HEAP-EXTRACT-MAX on the heap

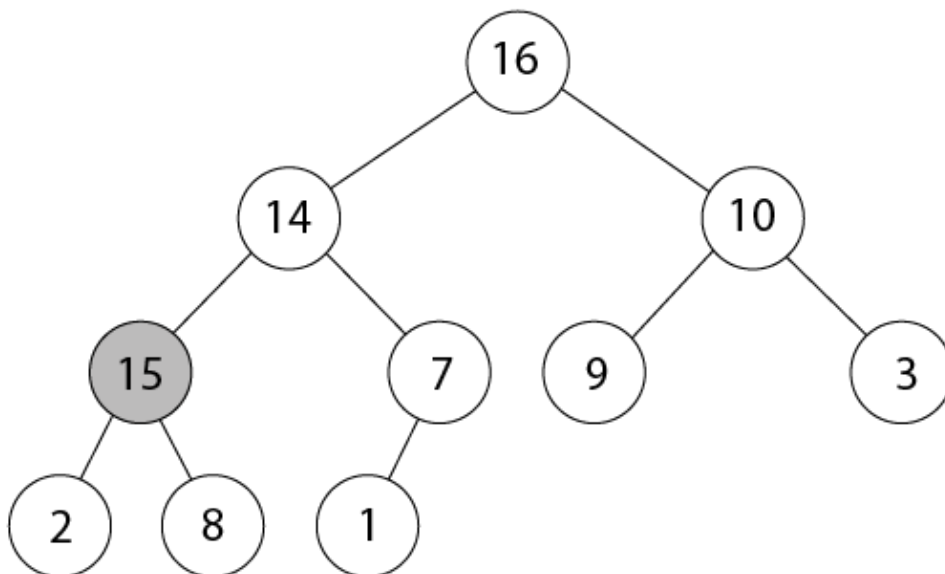
A= (15,13,9,5,12,8,7,4,0,6,2,1)

Fig: Operation of HEAP-INCREASE-KEY**Fig: (a)**

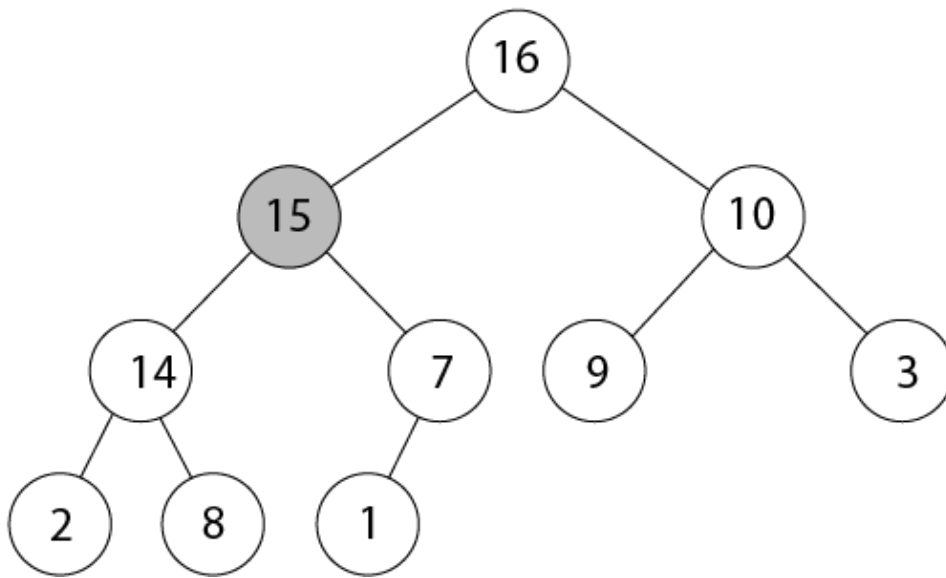
In this figure, that max-heap with a node whose index is 'i' heavily shaded

**Fig: (b)**

In this Figure, this node has its key increased to 15.

**Fig: (c)**

After one iteration of the while loop of lines 4-6, the node and its parent have exchanged keys, and the index i moves up to the parent.

**Fig: (d)**

The max-heap after one more iteration of the while loops, the $A[\text{PARENT}(i) \geq A(i)]$ the max-heap property now holds and the procedure terminates.

Heap-Delete:

Heap-DELETE (A, i) is the procedure, which deletes the item in node ' i ' from heap A , HEAP-DELETE runs in $O(\log n)$ time for n -element max heap.

HEAP-DELETE (A, i)

1. $A[i] \leftarrow A[\text{heap-size}[A]]$
2. $\text{Heap-size}[A] \leftarrow \text{heap-size}[A] - 1$
3. MAX-HEAPIFY (A, i)

[← Prev](#)[Next →](#)

 [For Videos Join Our Youtube Channel: Join Now](#)


Feedback


- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share




Learn Latest Tutorials


 [Splunk tutorial](#)
Splunk

 [SPSS tutorial](#)
SPSS


 [Swagger tutorial](#)
Swagger


 [T-SQL tutorial](#)
Transact-SQL

 [Tumblr tutorial](#)
Tumblr


 [React tutorial](#)
ReactJS

 [Regex tutorial](#)
Regex


 [Reinforcement learning tutorial](#)
Reinforcement Learning




R Programming tutorial
R Programming




RxJS tutorial
RxJS




React Native tutorial
React Native




Python Design Patterns
Python Design Patterns



Python Pillow tutorial
Python Pillow




Python Turtle tutorial
Python Turtle




Keras tutorial
Keras


Preparation




Aptitude
Aptitude




Logical Reasoning
Reasoning



Verbal Ability
Verbal Ability




Interview Questions
Interview Questions




Company Interview Questions
Company Questions


Trending Technologies




Artificial Intelligence
Artificial Intelligence




AWS Tutorial
AWS




Selenium tutorial
Selenium




Cloud Computing
Cloud Computing




Hadoop tutorial
Hadoop




ReactJS Tutorial
ReactJS




Data Science Tutorial
Data Science




Angular 7 Tutorial
Angular 7




Blockchain Tutorial
Blockchain



Git Tutorial
Git




Machine Learning Tutorial
Machine Learning




DevOps Tutorial
DevOps


B.Tech / MCA




DBMS tutorial
DBMS




Data Structures tutorial
Data Structures



DAA tutorial
DAA




Operating System
Operating System




Computer Network tutorial
Computer Network




Compiler Design tutorial
Compiler Design




Computer Organization and Architecture
Computer Organization




Discrete Mathematics Tutorial
Discrete Mathematics




Ethical Hacking
Ethical Hacking



Computer Graphics Tutorial
Computer Graphics



Software Engineering
Software Engineering




html tutorial
Web Technology




Cyber Security tutorial
Cyber Security




Automata Tutorial
Automata




C Language tutorial
C Programming




C++ tutorial
C++




Java tutorial
Java




.Net Framework tutorial
.Net



Python tutorial
Python



List of Programs
Programs



Control Systems tutorial
Control System



Data Mining Tutorial
Data Mining



Data Warehouse Tutorial
Data Warehouse

