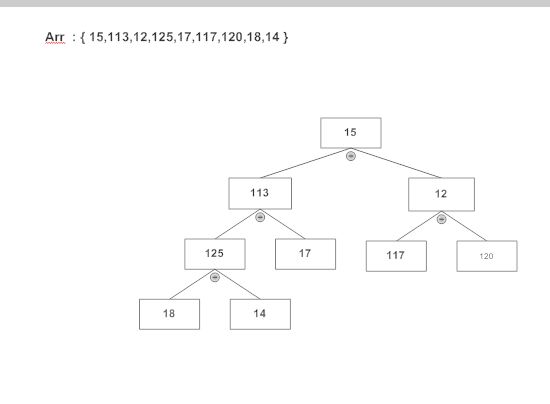
**Q2-1**

**illustrate the operation of HEAPSORT on the array A = {15, 113, 12, 125, 17, 117, 120, 18, 14}. Show, visually, the binary tree just after the BUILD-HEAP is called and at the end of each of the HEAPIFY operation**



PARENT(Index)

Return index/2

Left(Index)

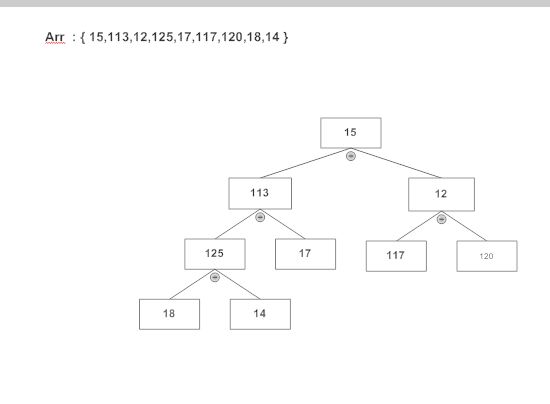
Return 2\*index

Right(Index)

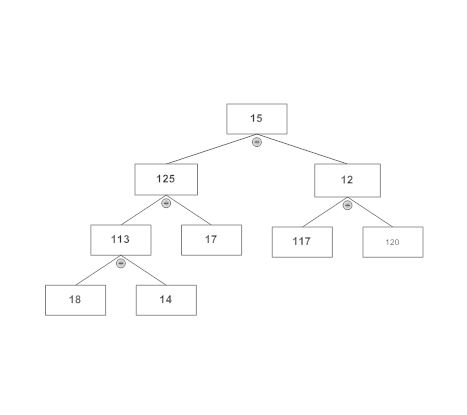
Return 2\*Index + 1

**Maintaining the Heap Property** :

**Heapify :** It is an important subroutine for manipulating heaps. When heapify is called it is assumed that the binary tree rooted at left(Index) and right (Index) are heap



(a)



(b)

**HEAPIFY (Arr,Index)**

L -> Left<------- 1

R -> Right <-------Index

If L<= heap-Size[Arr] and Arr[R] > Arr[Index]

Then Length 🡨 R

If largest Index

Then Exchange Arr[Index] 🡨🡪Arr[Length]

HEAPIFY(Arr,Largest)

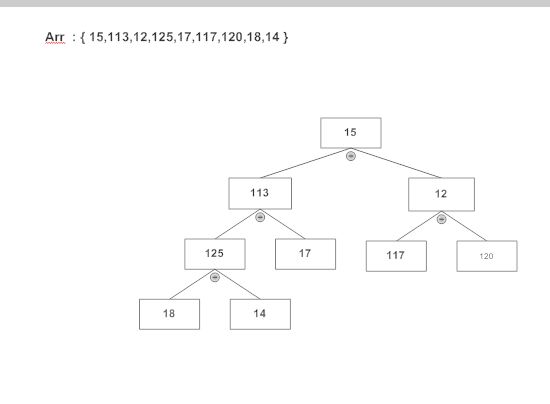
**BUILDHEAP(Arr)**

Heap-Size[Arr] 🡨 Length[Arr]

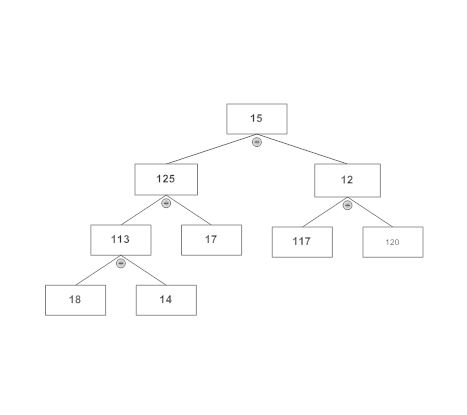
For index 🡨 Length[Arr]/2 down to 1

Do HEAPIFY(Arr,Index)

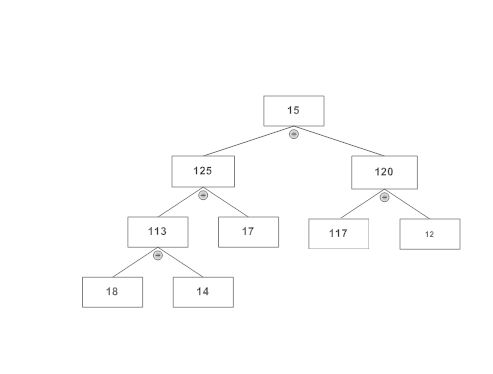
We can build a heap from an unordered array in linear time

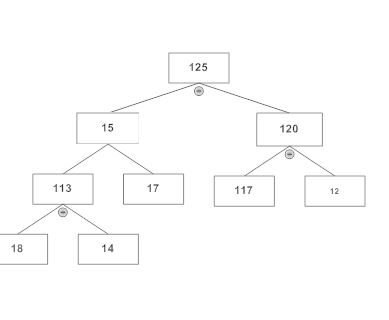


(a)

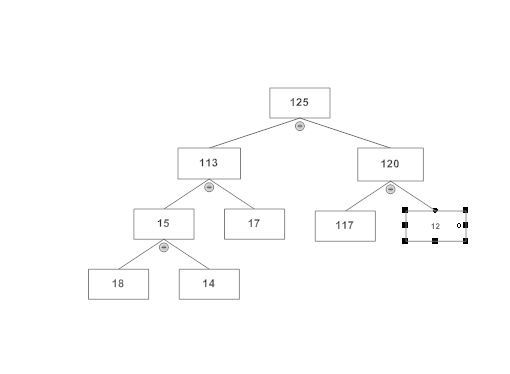


(b)

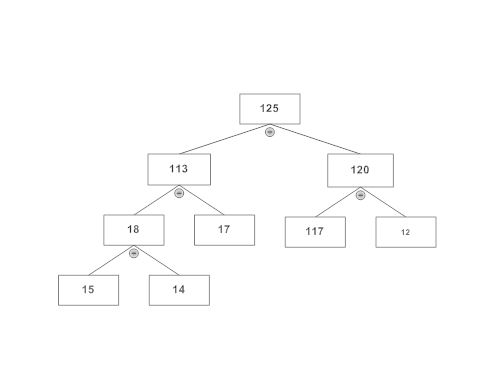




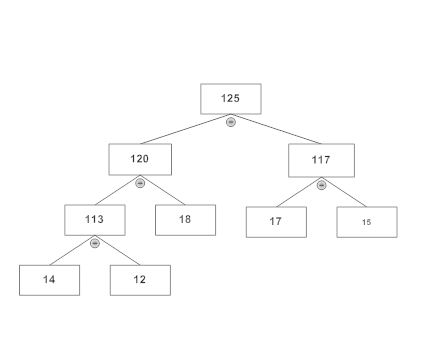
(d)



(e)



(f)



(g)

**HEAPSORT(Arr)**

For Index 🡨 Length[Arr] down to 2

Do exchange Arr[1] 🡨🡪 Arr[Index]

Heap-Size[Arr] 🡨 Heap-Size [Arr] – 1

HEAPIFY(Arr,1)