CIS 2101

Priority Queue

Priority Queue

- A set ADT with operations
  1. Insert
  2. Deletemin
- Priority Queue includes operations Initialize and Makenull
- **Application:** Process Scheduling in a time-shared computing system

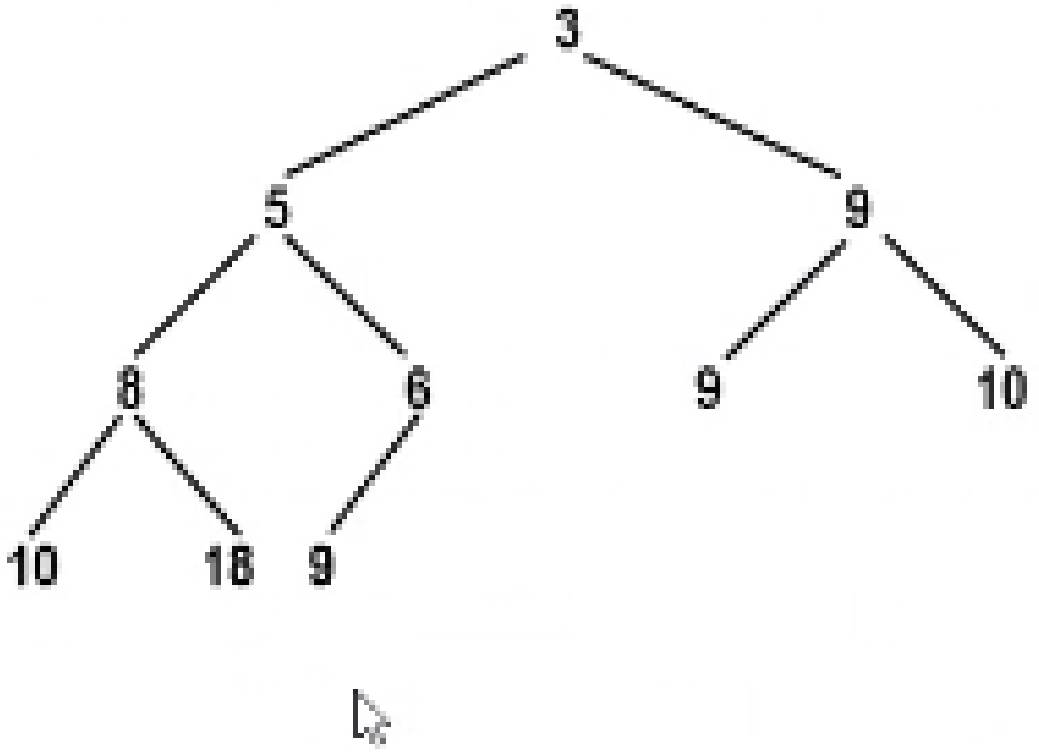
Priority Queue

- Operations
  1. Insert(x,A) - adds element x in set A , if the element x is not yet a member of the set.
  2. Deletemin(A) - removes and returns the smallest element in set A if the set is not empty; otherwise the operation fails

Priority Queue Implementations

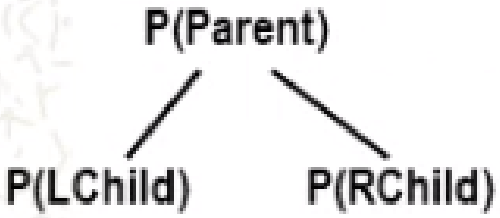
1. Bit-vector
2. Linked list
3. Array
4. Cursor based
5. Partially Ordered Tree (P.O.T.)

P.O.T. Illustration



Characteristics of P.O.T.

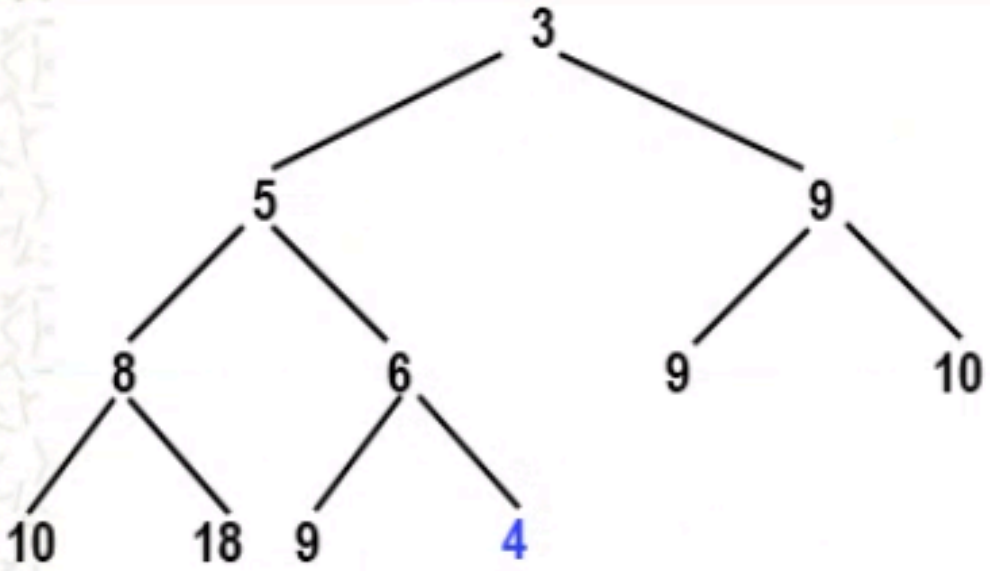
1. Binary Tree – a tree in which each node has either a left child, or a righ child or both left and right children
2. Balanced Tree – a tree in which the height is a minimum for the # of nodes
3. At the lowest level, where the leaves may be missing, we require that all missing leaves are to the right of all the leaves present in the lowes level.
4. POT Property – the priority of the parent is less than or equal to the priority of the children



P(Parent) <= P (Children)

Illustrations of P.O.T. Operations

Insert 4

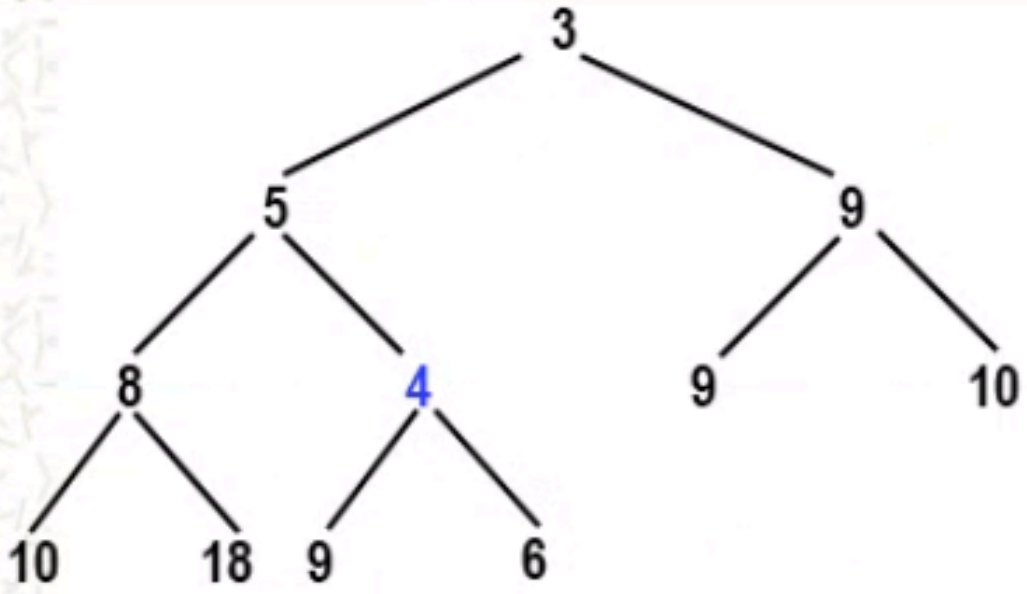


Is P.O.T. property satisfied? **NO!**

What is to be done? **SWAP 6 & 4 !**

Insert 4

(Swap 6 & 4)

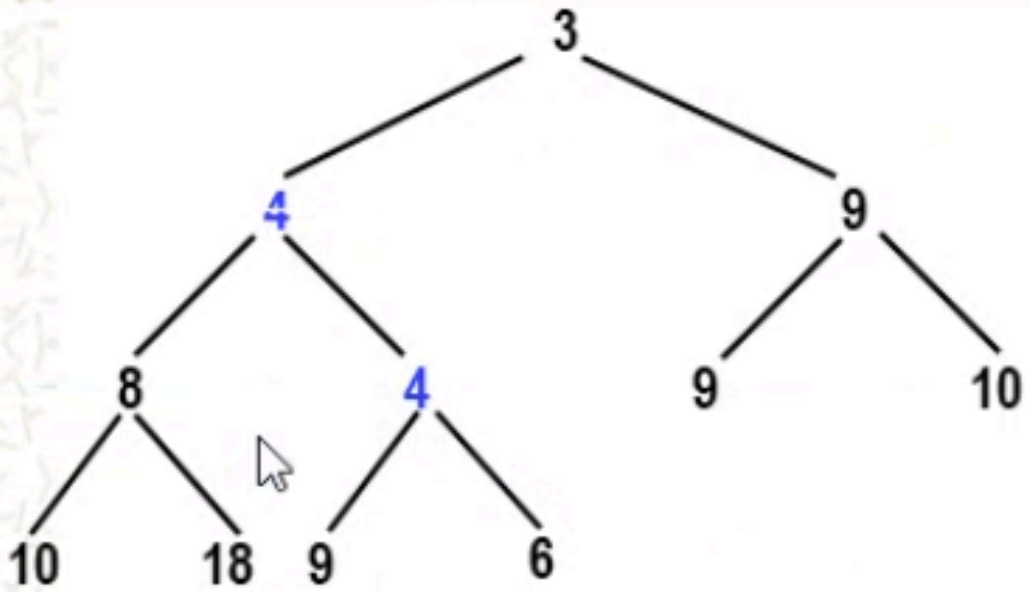


Is P.O.T. property satisfied? **NO!**

What is to be done? **SWAP 5 & 4 !**

Insert 4

(Swap 5 & 4)



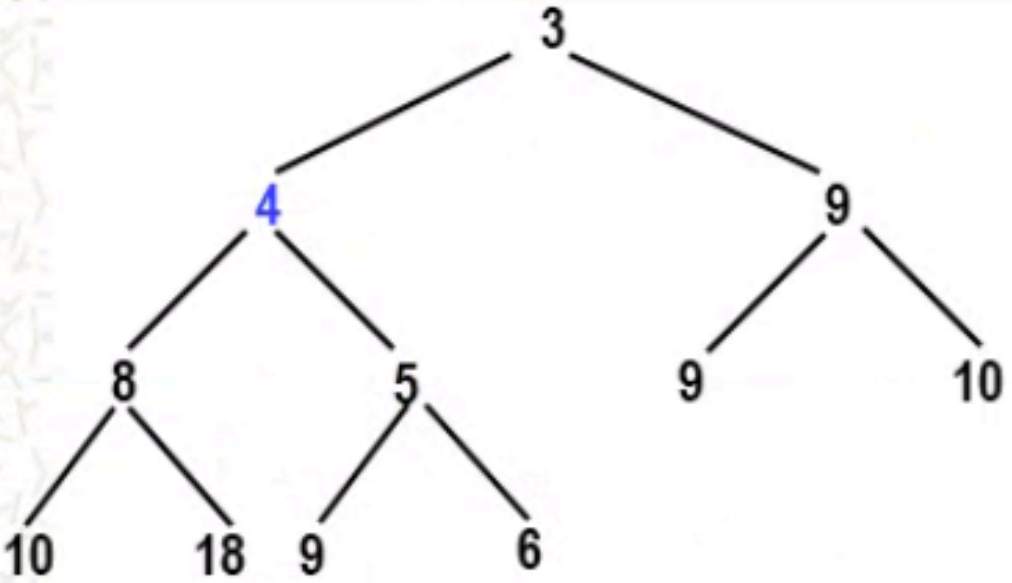
Swap 5 & 4

5

temp

Insert 4

(Swap 5 & 4)

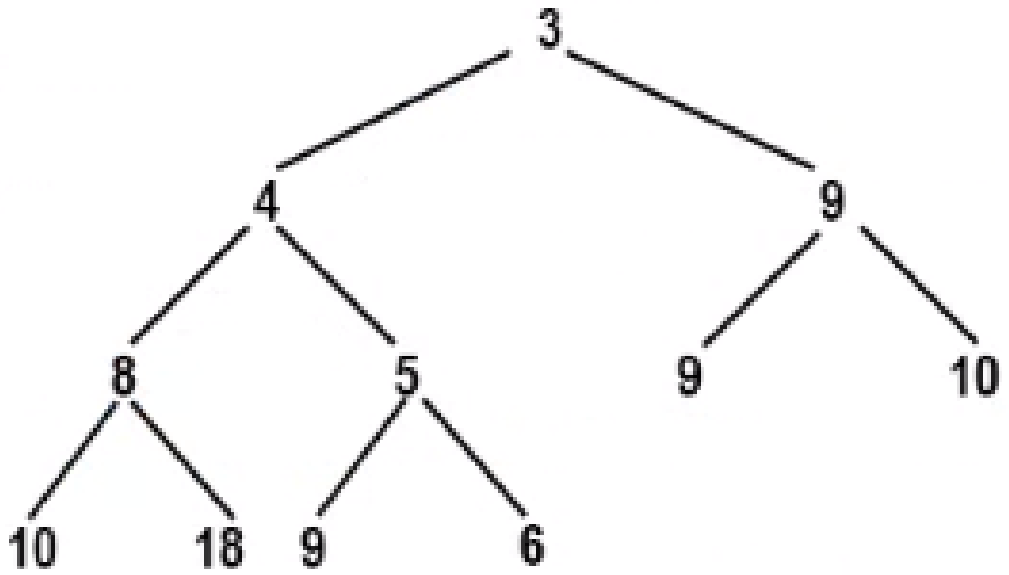


Is P.O.T. property satisfied? **YES!**

Steps in INSERT Operation

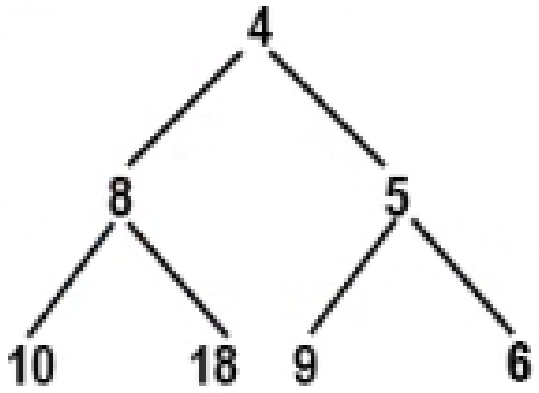
1. Place new element x at
   the lowest level to the right of the leaves present
   OR
   next level if current level is full
2. while (x is not root and POT property is not satisfied)
   SWAP(parent, x);

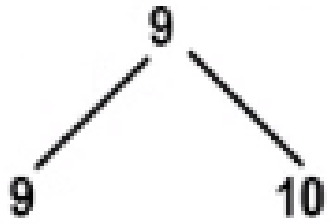
Deletemin



Deletemin

To be returned



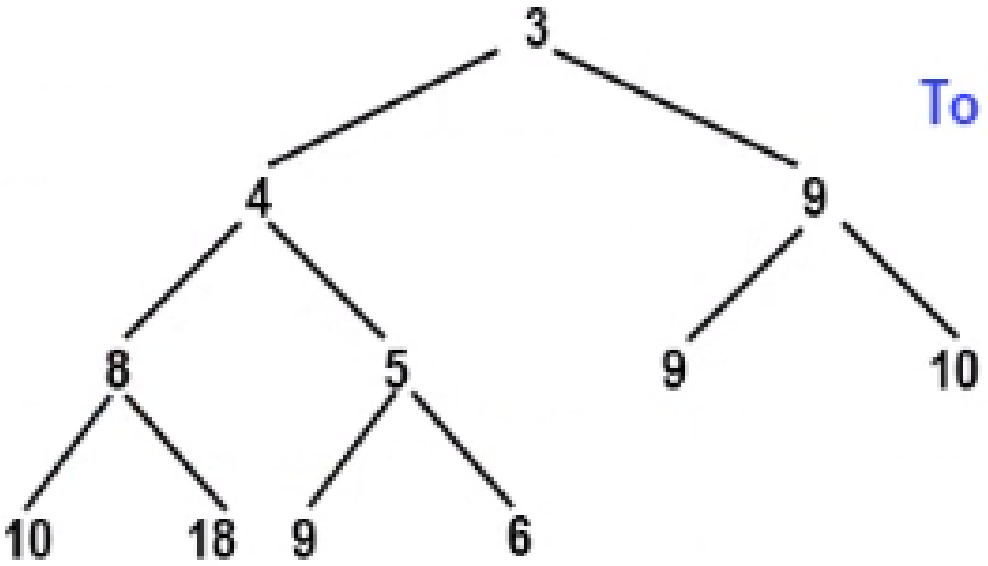


3
min

What is the result? A Forest!

Solution? Replace the root node

Deletemin

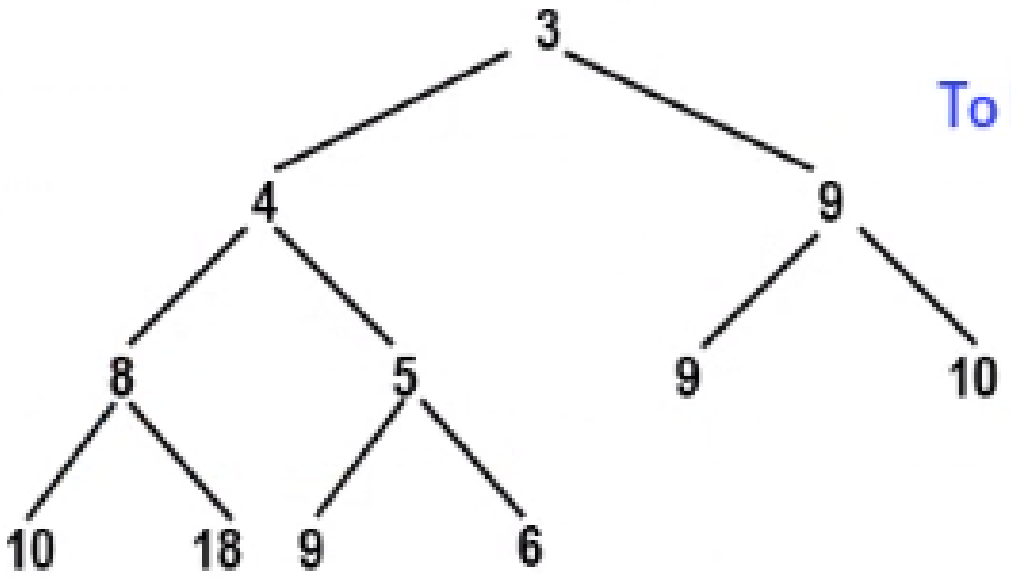


To be returned

3

min

Deletemin

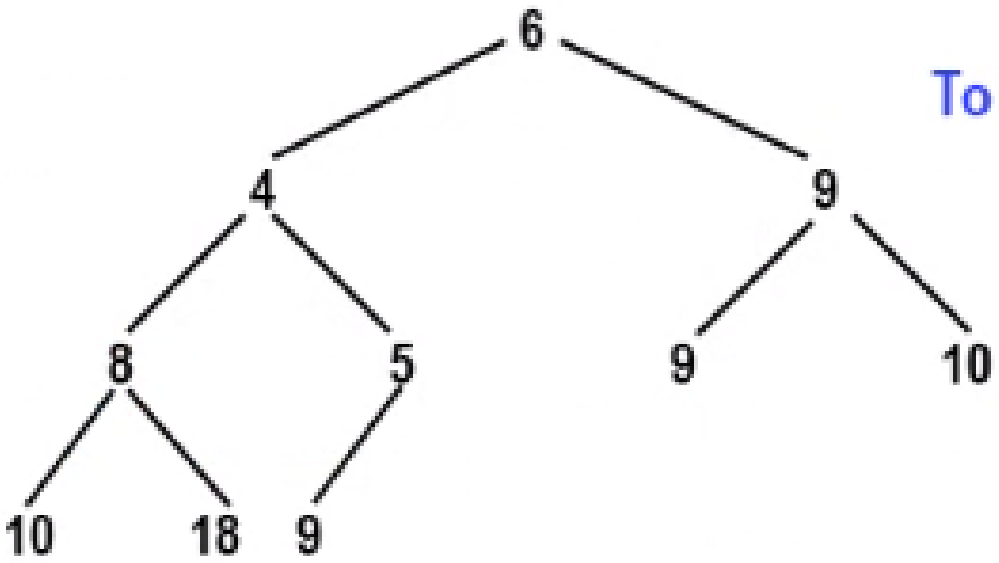


To be returned

3

min

Deletemin

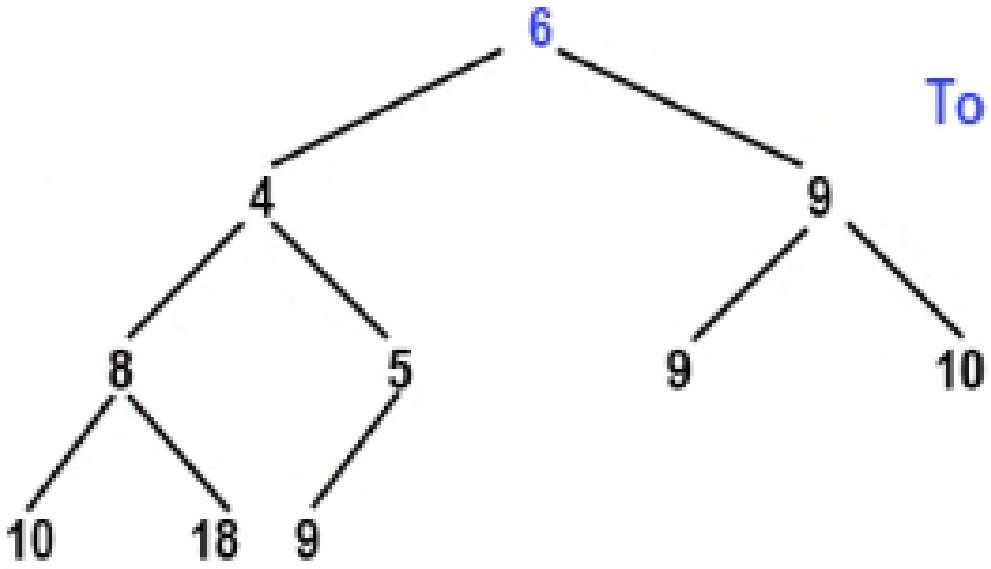


To be returned

3

min

Deletemin



To be returned

3

min

Is P.O.T. property satisfied? **NO!**

What is to be done? **Swap 6 with its smallest child!**

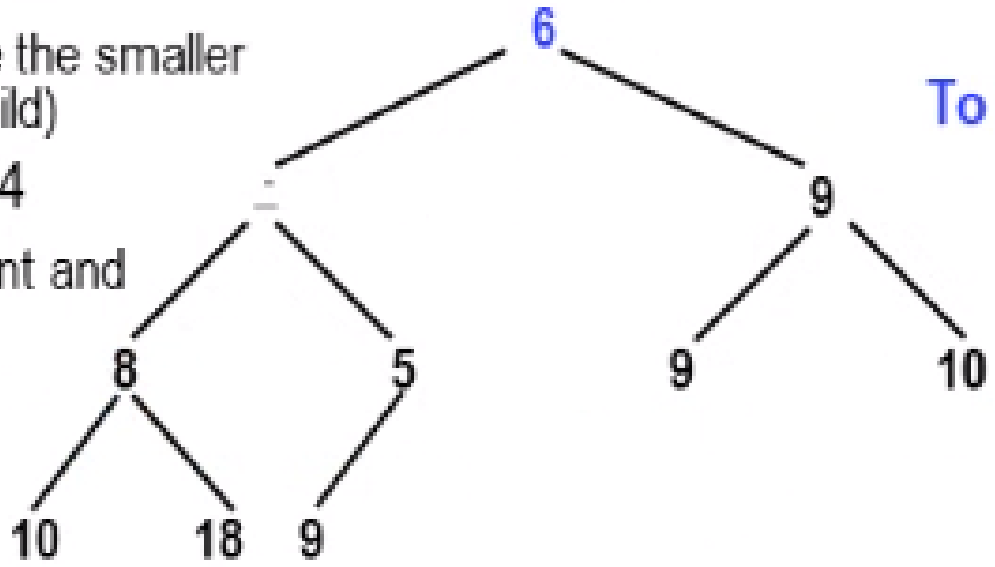
Deletemin

(Swap 4 & 6)

1. Determine the smaller child (SChild)

   SChild = 4

2. Swap parent and SChild



To be returned

3
min

4
temp

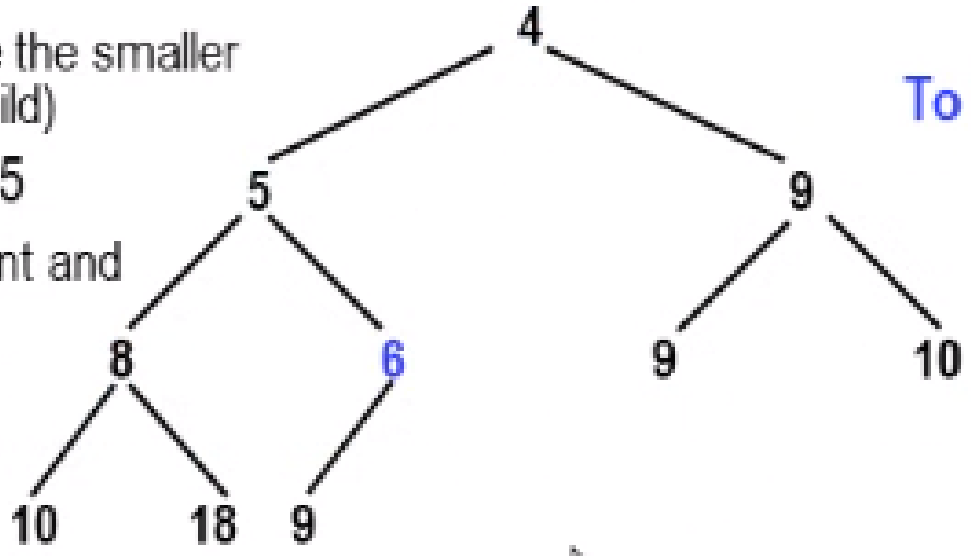
Deletemin

(Swap 5 & 6)

1. Determine the smaller child (SChild)

   SChild = 5

2. Swap parent and SChild



To be returned

3
min

5
temp

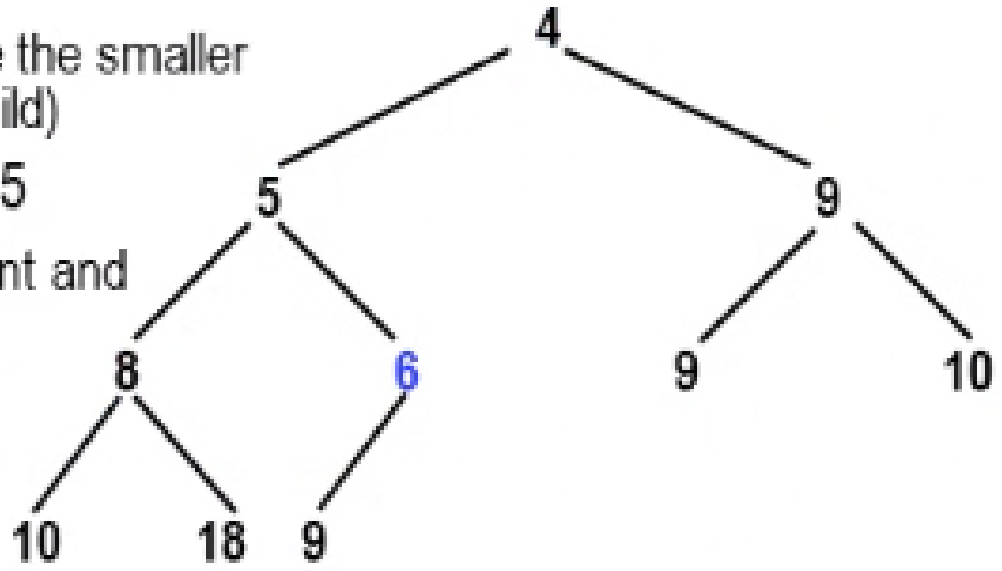
Deletemin

(Swap 5 & 6)

1. Determine the smaller child (SChild)

   SChild = 5

2. Swap parent and SChild



Is P.O.T. property satisfied? **YES!**

Return Minimum Priority : **3**

Deletemin operation is **DONE!**

Steps in Deletemin Operation

```
1. min = root node
2. Replace root with the element X found
   at the lowest level far right
3. While ( (X is not leaf) and NOT POT) {
       SChild = smaller child of parent X;
       SWAP(X, SChild);
   }
4. return min;
```

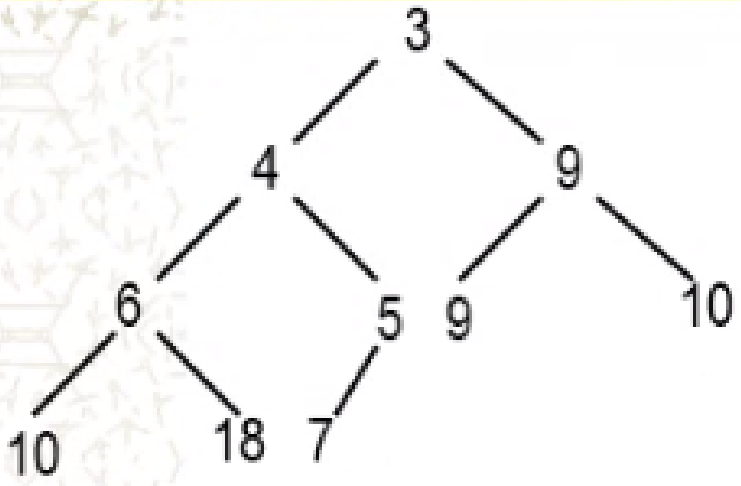
Implementation of POT

Heap

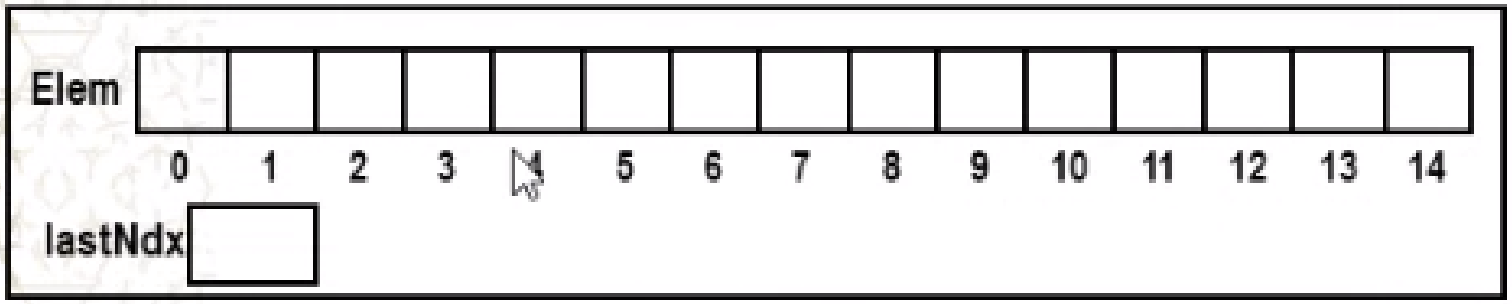
- Array Implementation of P.O.T.

Heap
(Data Structure)

POT



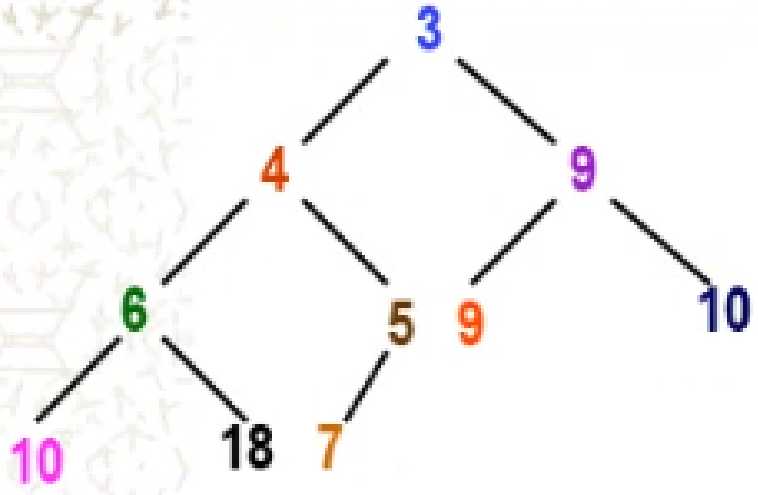
HEAP



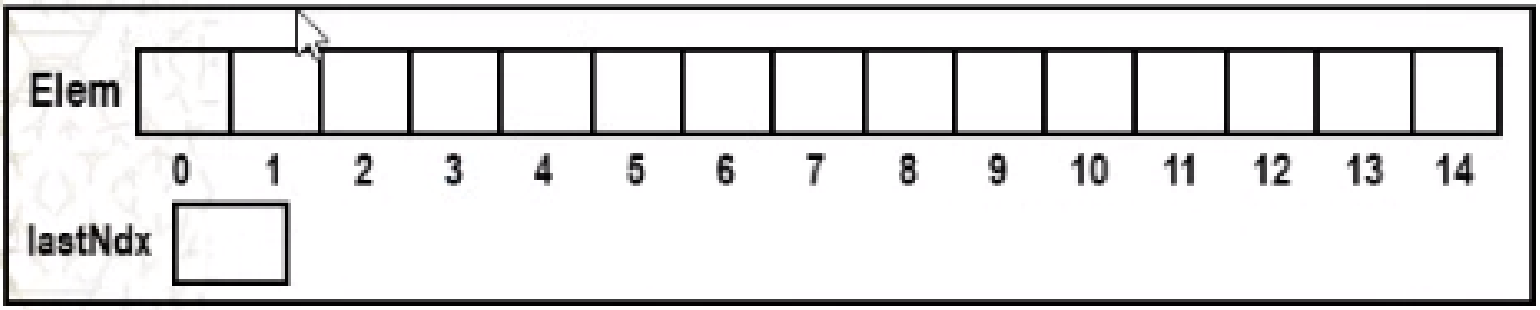
Heap

(How elements are stored)

POT



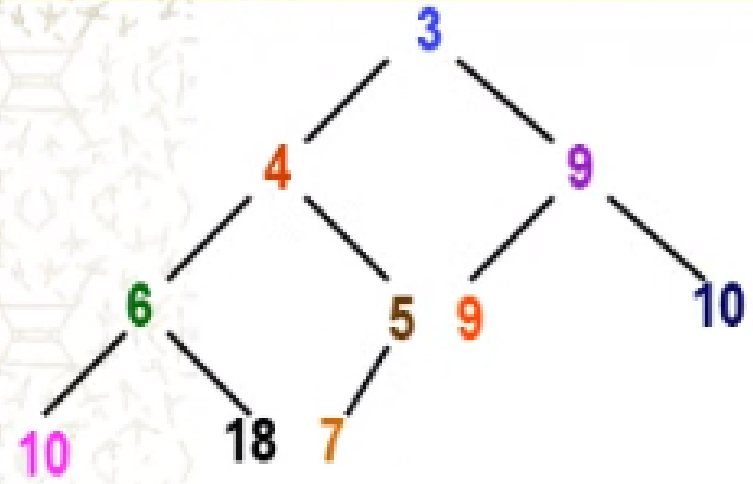
HEAP



Heap

(How elements are stored)

POT



HEAP

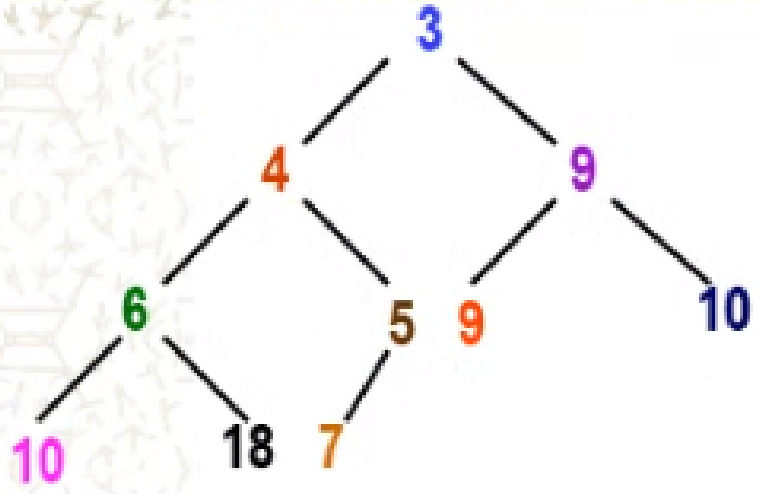
Elem	3	4	9	6	5	9	10	10	18	7					
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

lastNdx: 9

Heap

(How elements are stored)

POT



Parent Node	Parent Index	LChild Index	RChild Index
3	0	1	2

HEAP

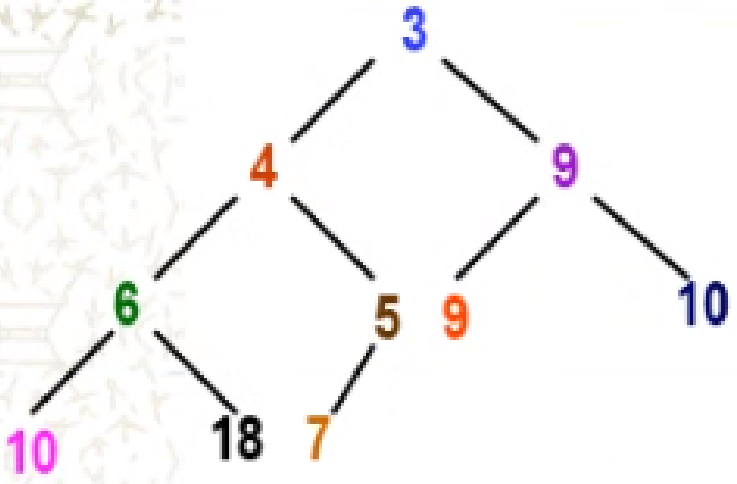
Elem	3	4	9	6	5	9	10	10	18	7					
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

lastNdx 9

Heap

(How elements are stored)

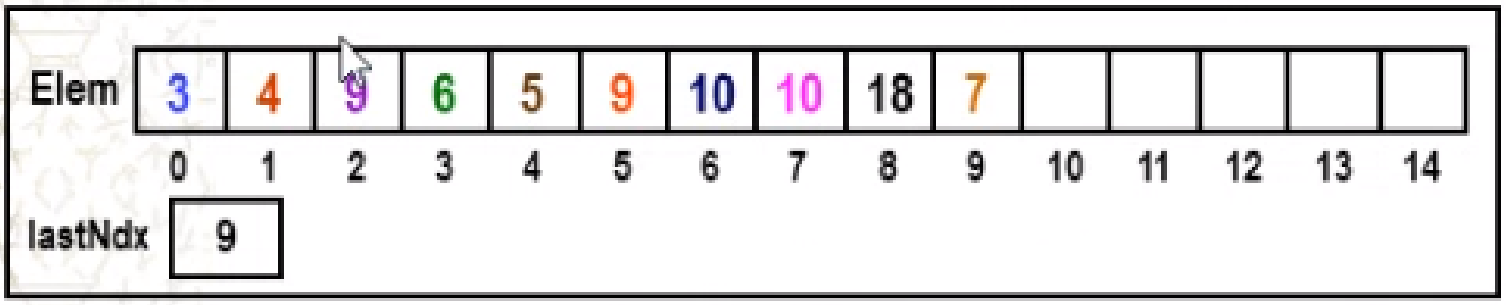
POT

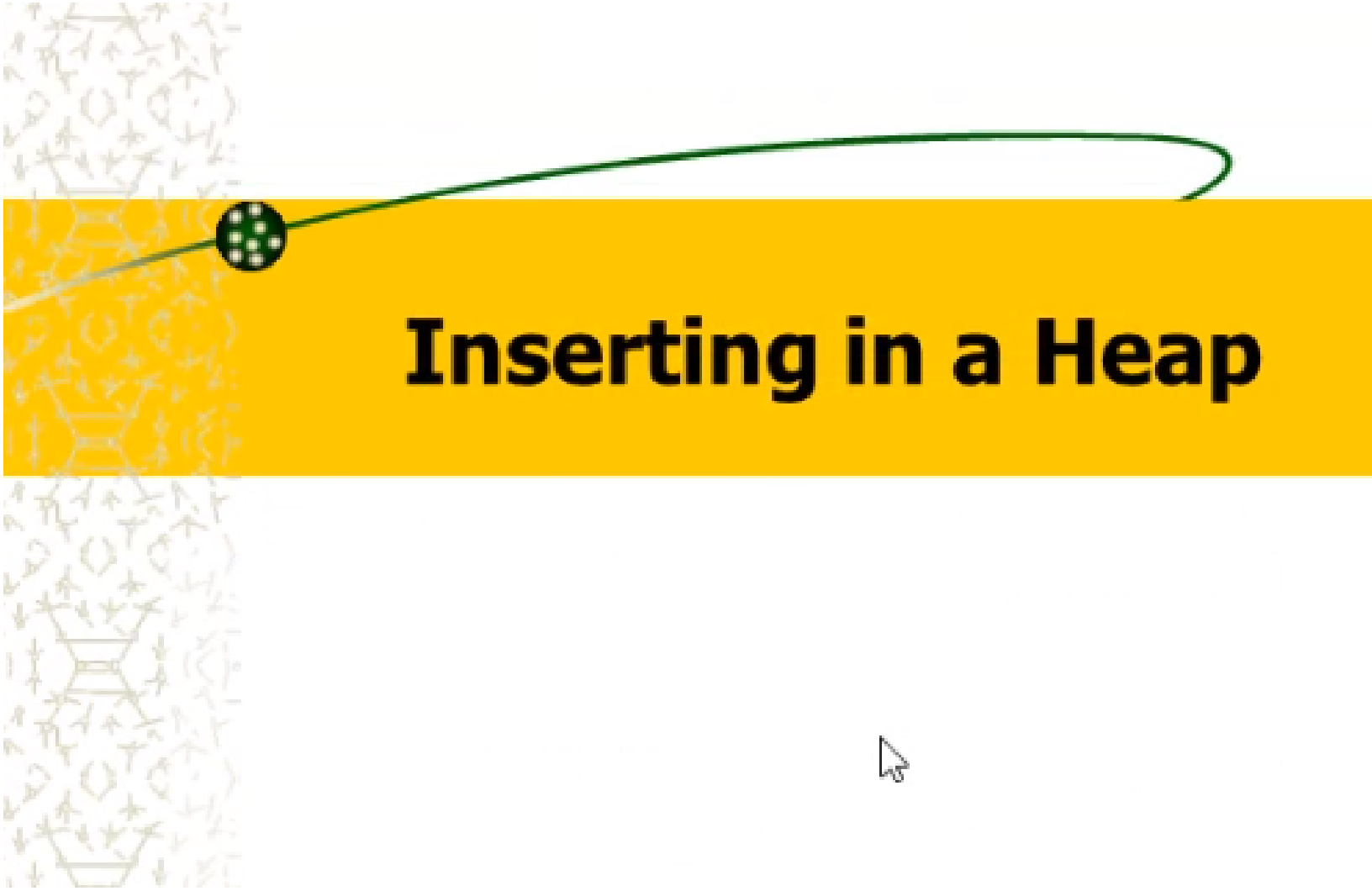


Parent Node	Parent Index	LChild Index	RChild Index
3	0	1	2
4	1	3	4
9	2	5	6
6	3	7	8

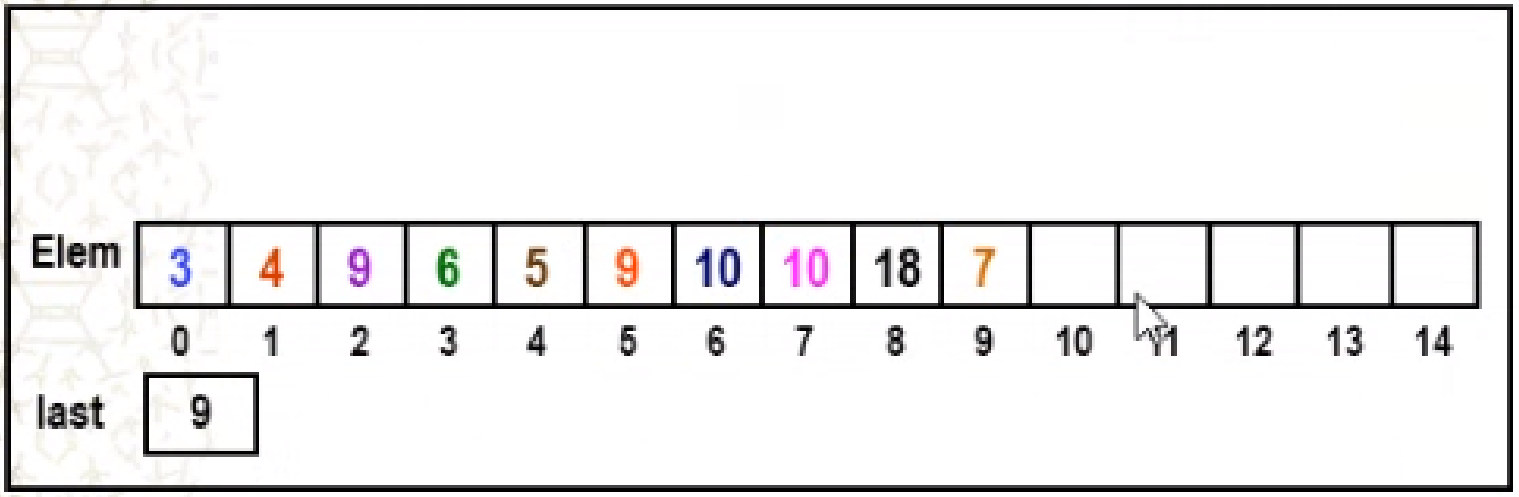
Can you see a pattern ?

HEAP

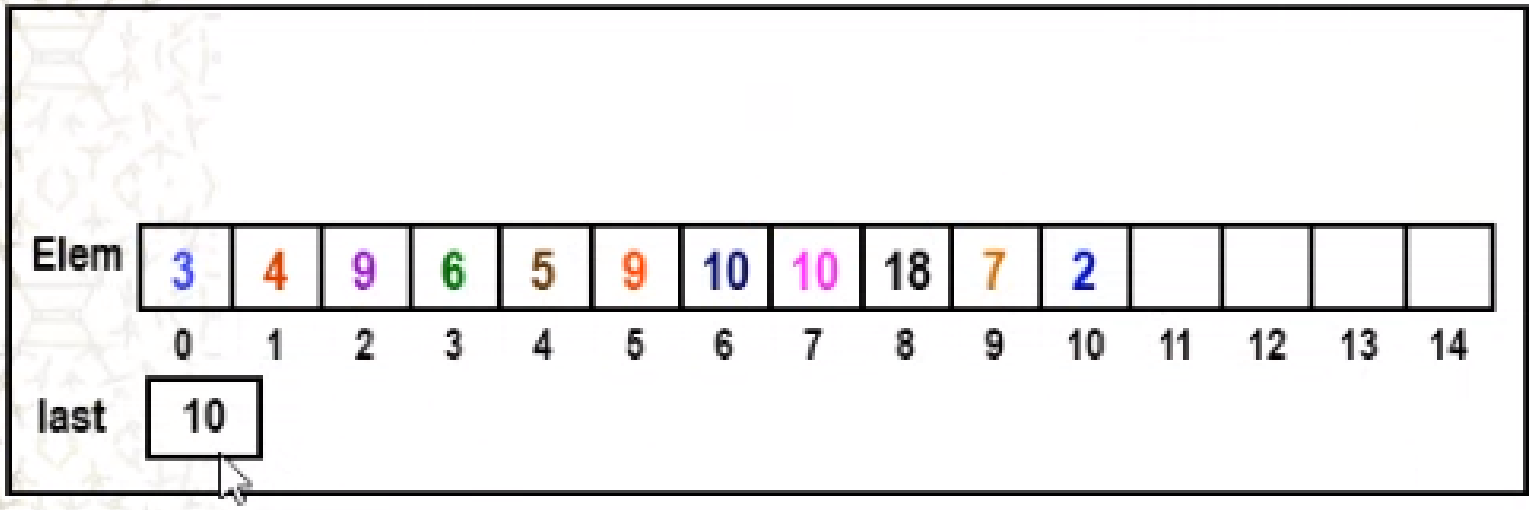



Inserting in a Heap

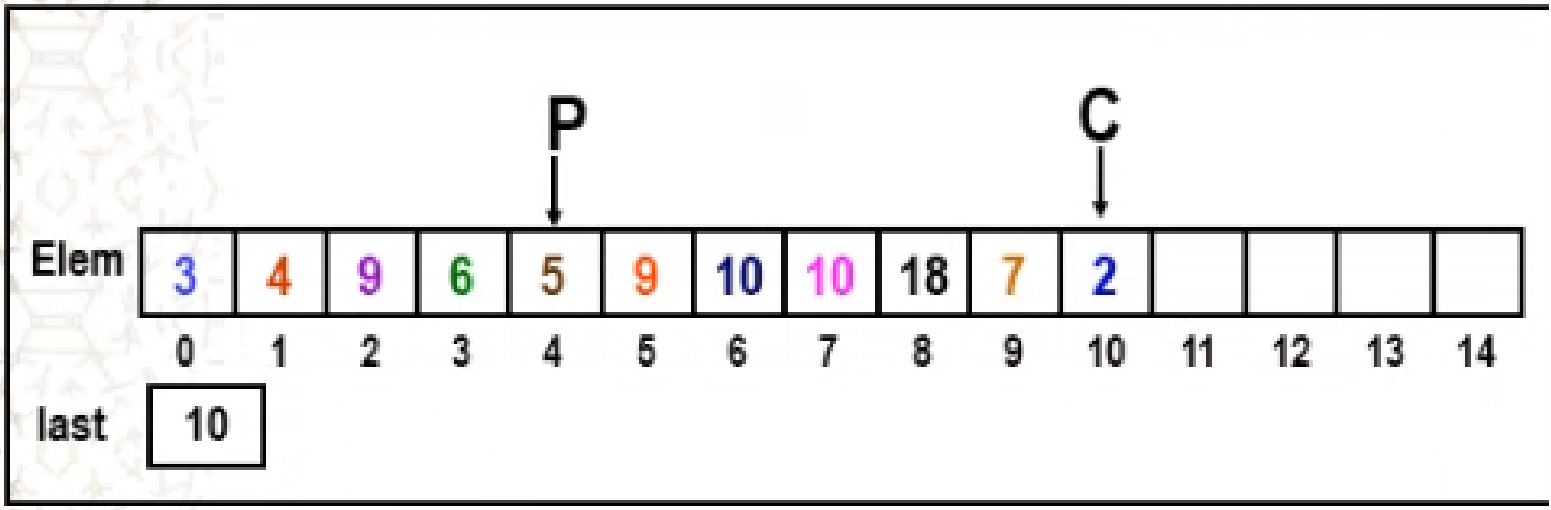
Insert 2 in the Heap



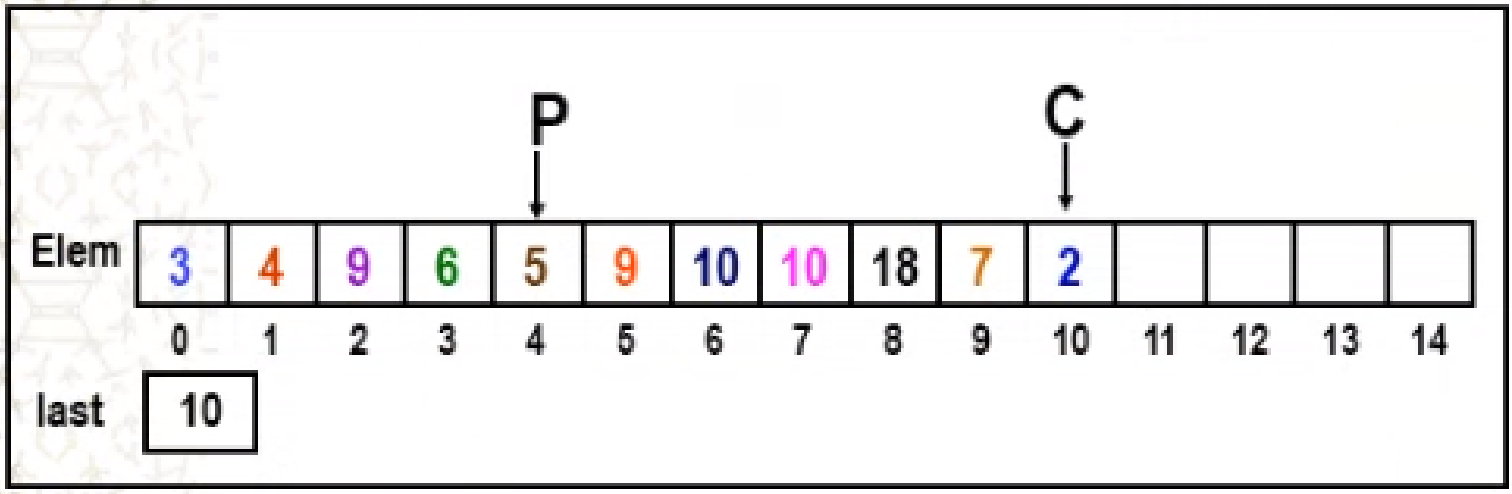
Insert 2 in the Heap



Insert 2 in the Heap



Insert 2 in the Heap

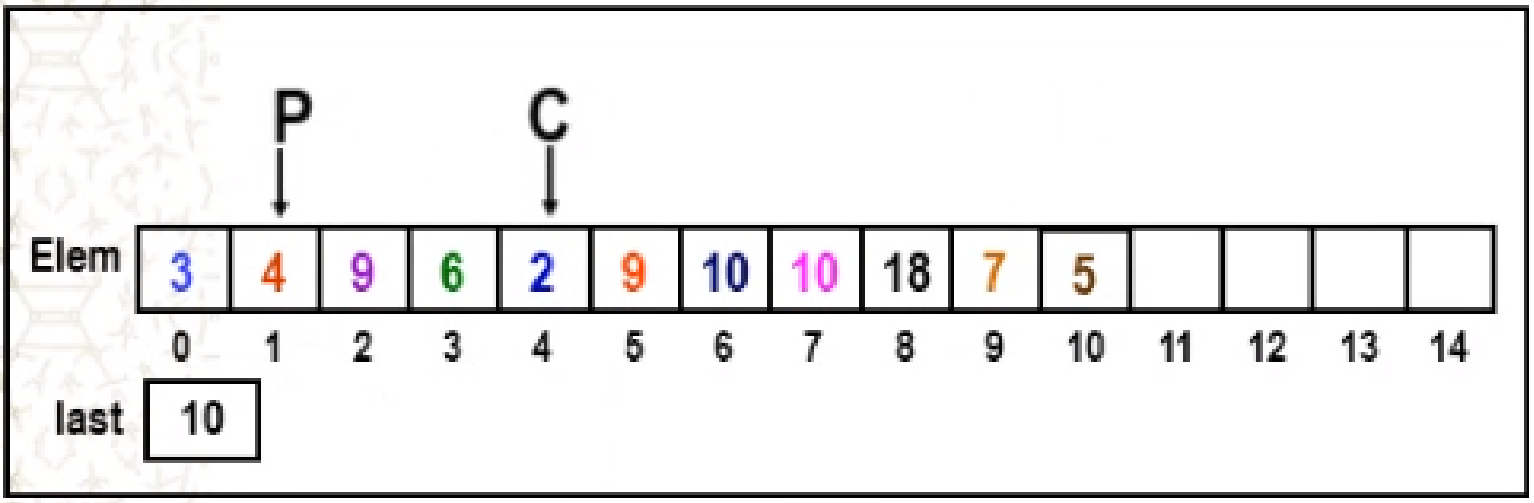


Is P.O.T. property satisfied ? NO!

What is to be done ? SWAP 2 & 5 !

Insert 2 in the Heap

(1st Iteration)

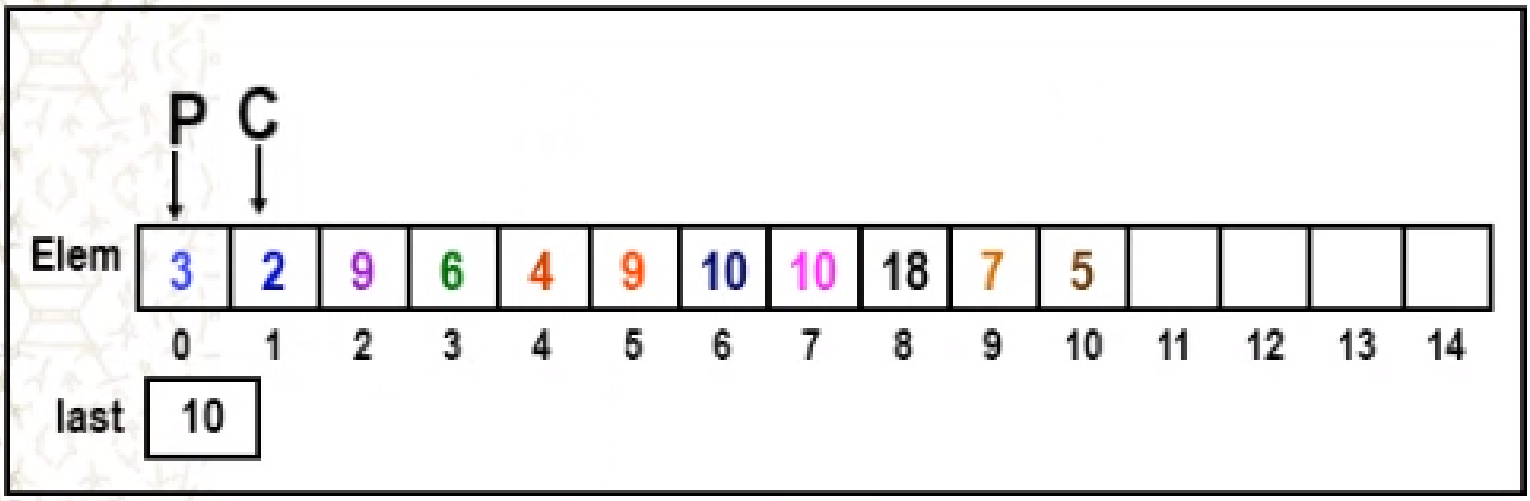


Is P.O.T. property satisfied ? **NO!**

What is to be done ? **SWAP 2 & 4 !**

Insert 2 in the Heap

(2nd Iteration)

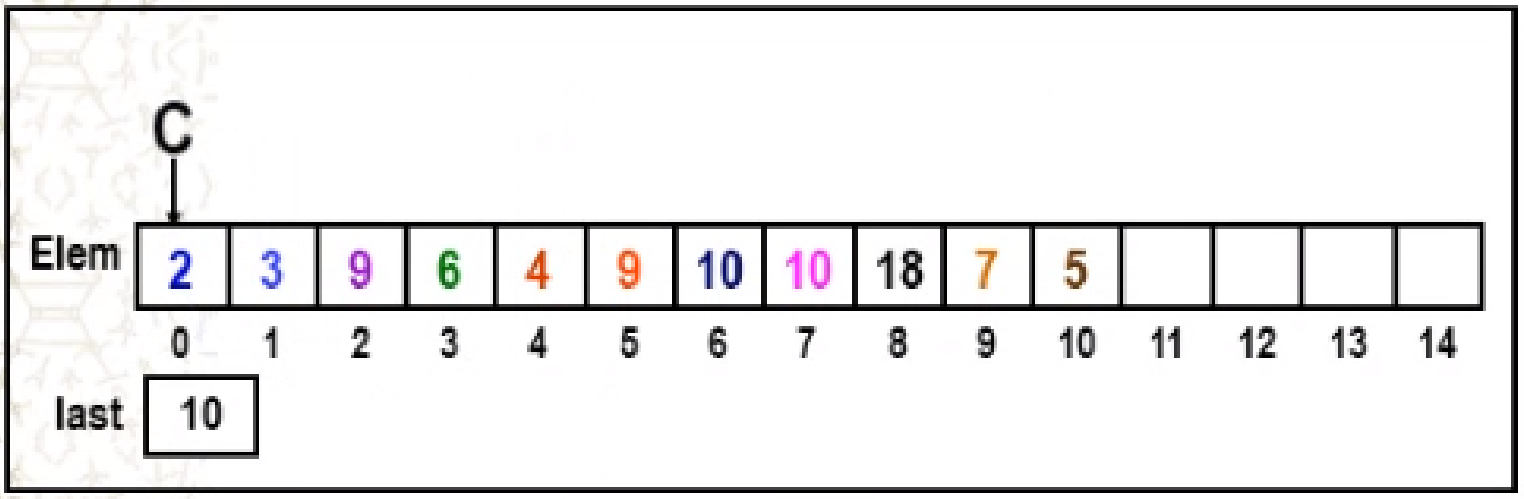


Is P.O.T. property satisfied ? **NO!**

What is to be done ? **SWAP 2 & 3 !**

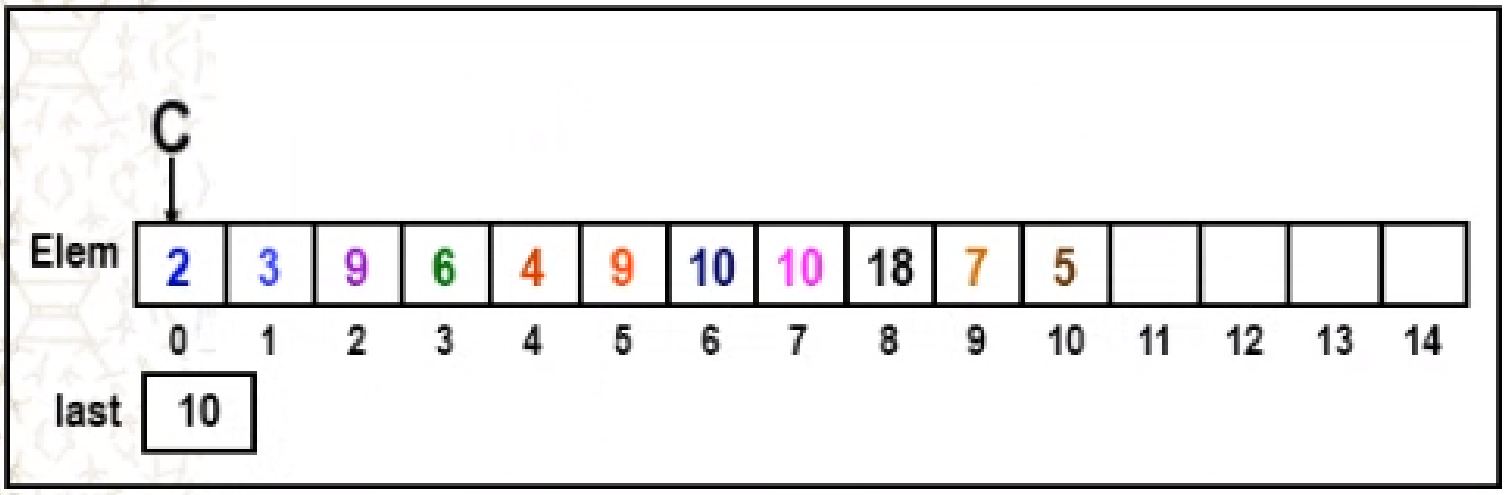
Insert 2 in the Heap

(3rd Iteration)



Insert 2 in the Heap

(3rd Iteration)



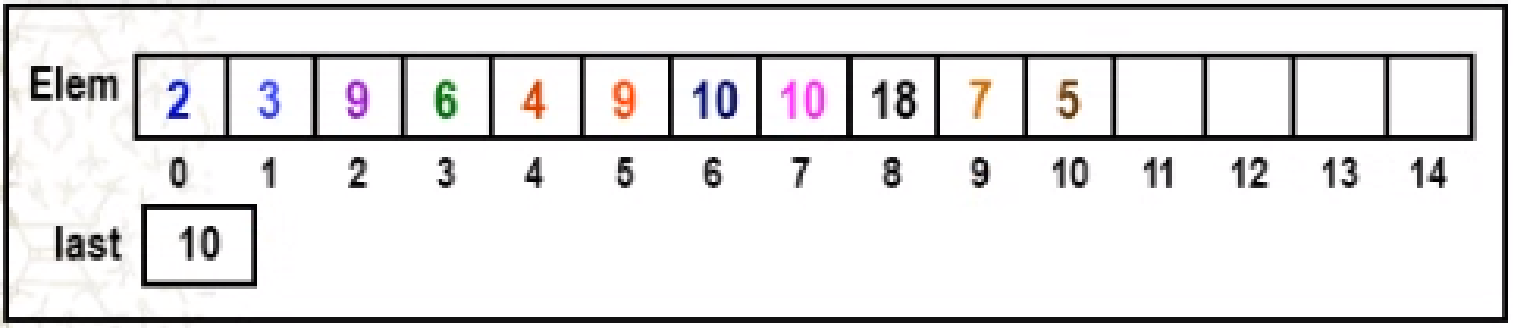
Is P.O.T. property satisfied ? **YES !**

Insert Operation is **DONE !**

Insert 2 in the Heap

(3rd Iteration)

Heap H



1. Given the illustration of the heap above, write an appropriate definition of datatype Heap and declaration of Heap H.
2. Based on the definition and declaration, write the codes of the following operations:
   a) Insert
   b) Deletemin

Heapsort

- A sorting technique which uses the concept of a P.O.T. implemented using an array

Heapsort

- A sorting technique which uses the concept of a P.O.T. implemented using an array
- Steps in HEAPSORT (**Descending order**)

1. INSERT all elements to be sorted in an initially empty Partially Ordered Tree
2. While (POT is not empty)
   - DELETEMIN and store the deleted minimum element in the position of the element which has replaced the root

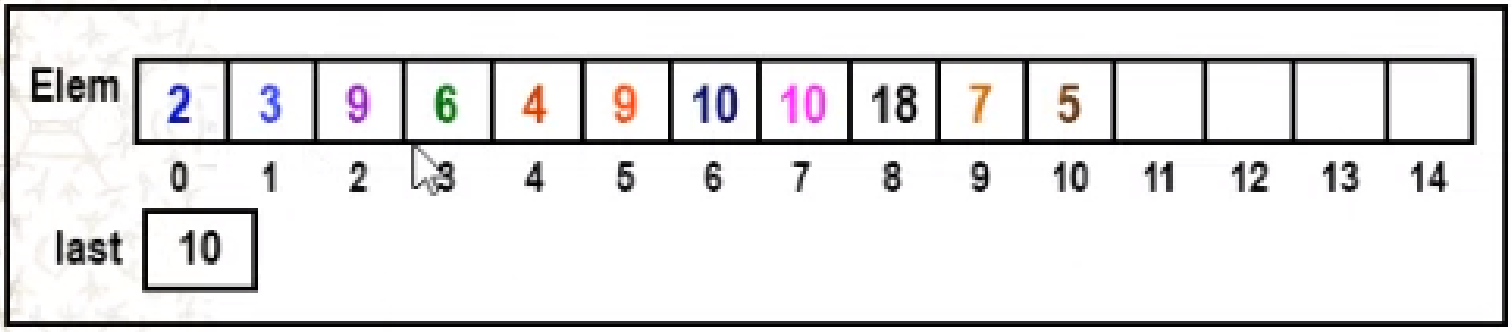
Heapsort

Step 1: Insert all elements in an initially empty POT (heap)

OR

Heapify the list (i.e. make list into a heap)

Resulting Heap H

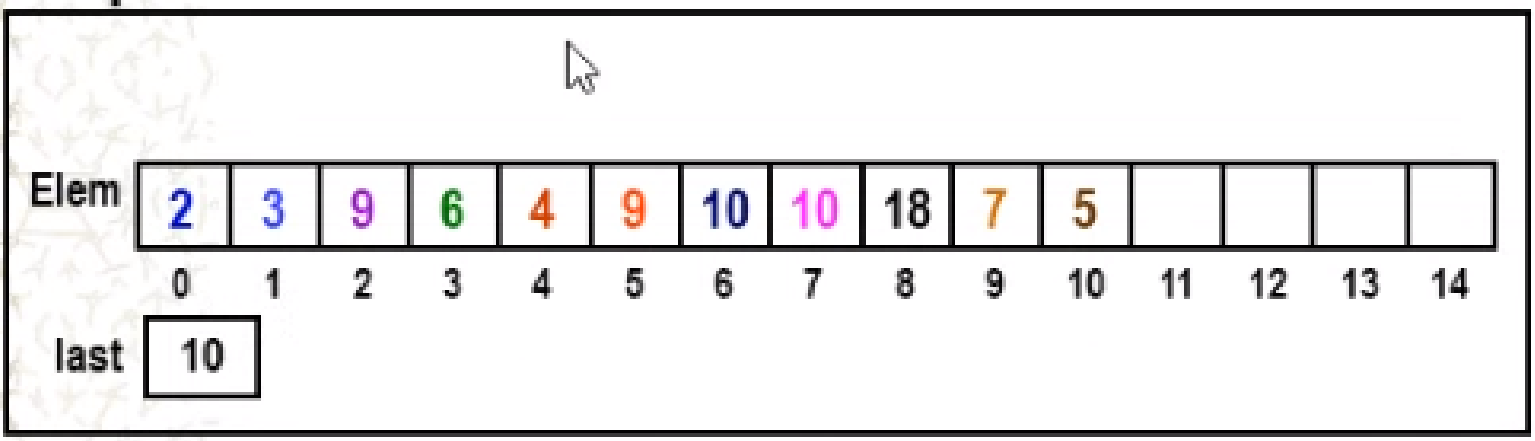


Heapsort: Step 2a

Deletemin: 2

2	10
temp	oldLast

Heap H



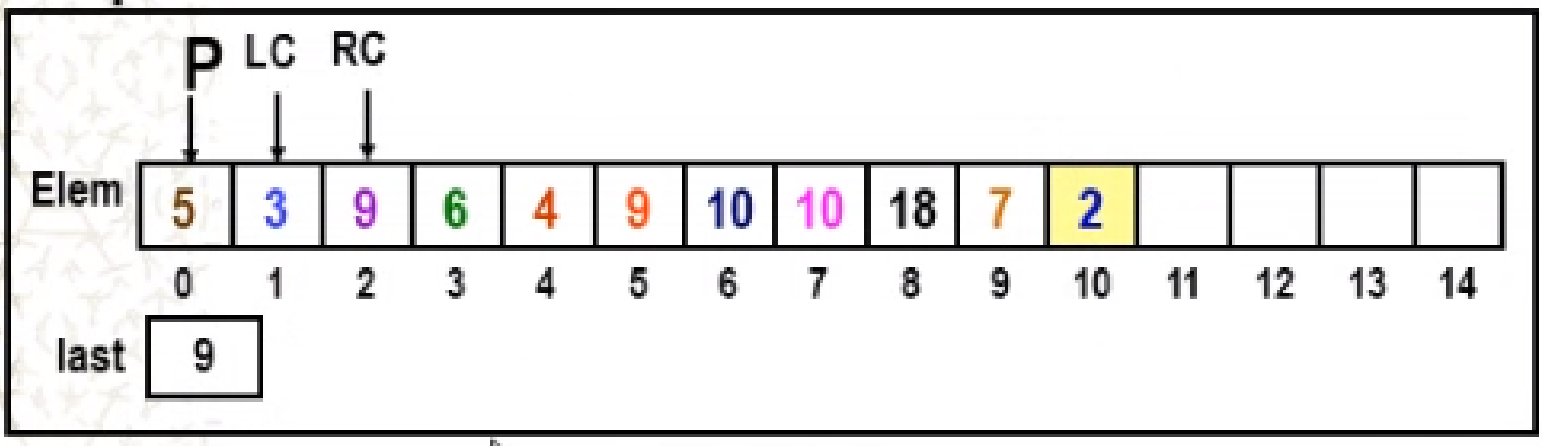
Heapsort: Step 2a

Deletemin: 2

2	10
temp	oldLast

Push the new root down the Tree until P.O.T. Property is Satisfied!!

Heap H



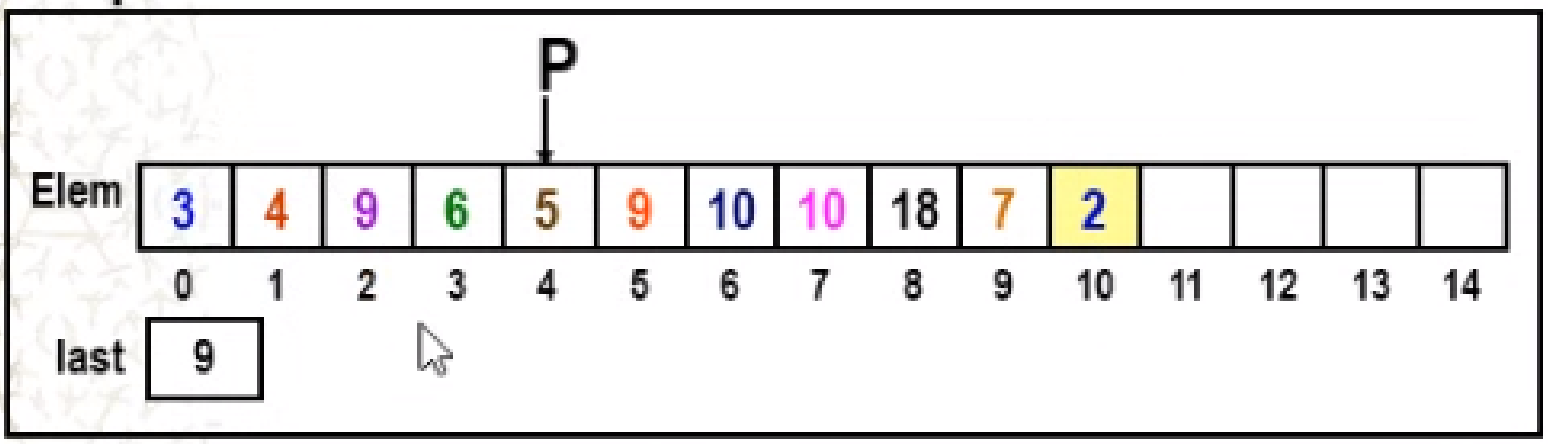
Heapsort: Step 2a

Deletemin: 2

5	10
temp	oldLast

Push the new root down the Tree until P.O.T. Property is Satisfied!!

Heap H



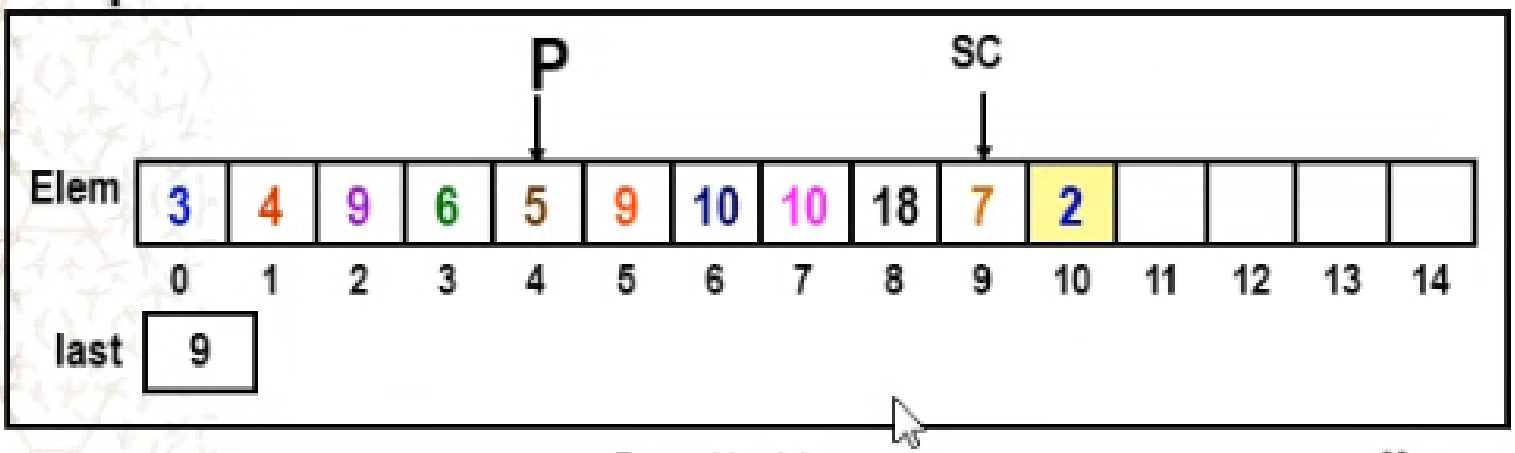
Heapsort: Step 2a

Deletemin: 2

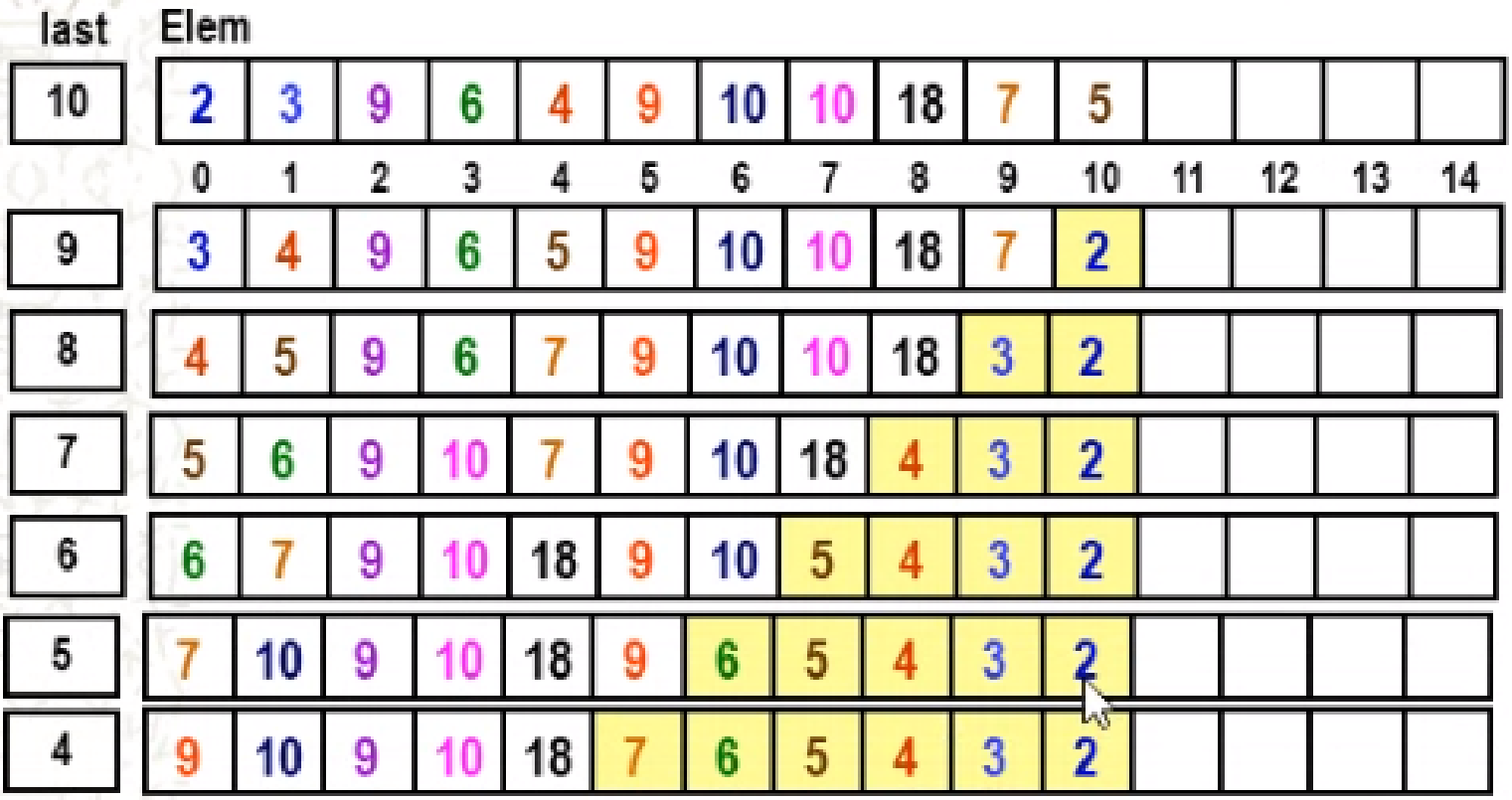
5	10
temp	oldLast

Push the new root down the Tree until P.O.T. Property Satisfied!!

Heap H



Heapsort Result



Heapsort Result

last	Elem														
10	2	3	9	6	4	9	10	10	18	7	5				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	9	10	9	10	18	7	6	5	4	3	2				
3	9	10	18	10	9	7	6	5	4	3	2				
2	10	10	18	9	9	7	6	5	4	3	2				
1	10	18	10	9	9	7	6	5	4	3	2				
0	18	10	10	9	9	7	6	5	4	3	2				
-1	18	10	10	9	9	7	6	5	4	3	2				

From Heap to Sorted List