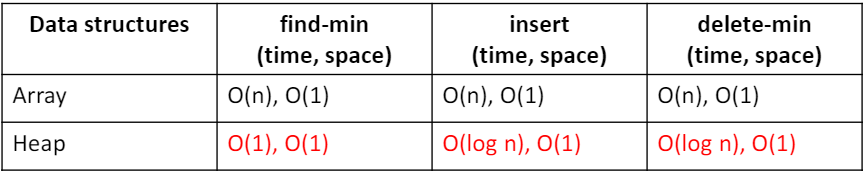
Heaps

A heap is a data structure for finding and deleting the minimum element, and inserting any element.

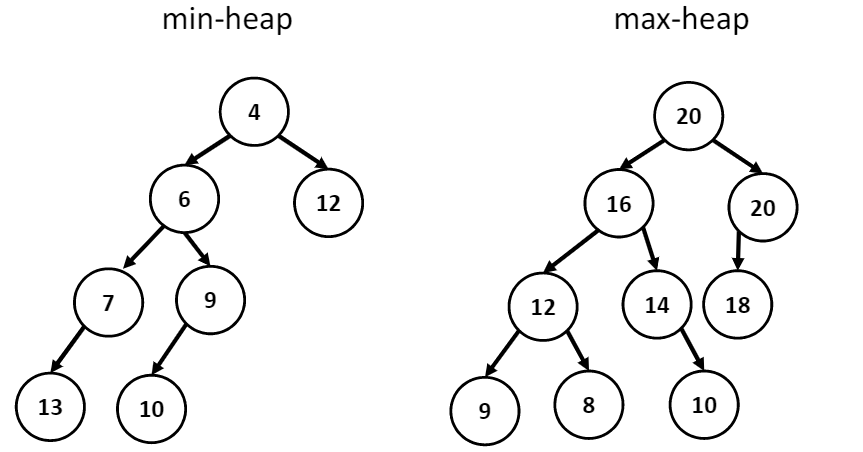


**Concept of Heaps**

* In some applications, it requires us to find a data element of the minimum key value. For example, when we do selection sort, in each iteration, we find the minimum element in unsorted elements.
* Can we design a more efficient data structure for finding the minimum element?

**Definition of Heap**

* An abstract minimum (or maximum) heap is a tree based data structure that satisfies the following heap properties:
  + Data elements have a tree relation. Each node N is associated with a key value (N). If node A has child node B, then key (A) <= key (B) (or key(A)>=key(B) for maximum heap)
  + The major operations include finding the minimum (or maximum) key value node, insert operation and delete minimum (or maximum) node operation.
  + A heap data structure is an implementation of the abstract heap. It is either a minimum heap (min-heap) or a maximum heap (max-heap).



**Min heap operations**

Find-min: find the node with minimum key value from a min-heap

Delete-min: delete the minimum key node from a min-heap

Insert: insert key data into heap

Decrease-key: decrease a key value of a node

Increase-key: increase a key value of a node

\*All heap operations need to maintain heap property

Similar operations are defined for max-heap

**Applications of heaps**

1. Heap-sort: insert elements into heap, find-max/find-min delete get one after another, return a sorted list of elements Time: O(log n), in place Space: O(1)
2. Used to maintain memory resources, heap region for dynamic memory allocation, i.e., malloc()
3. Used to implement priority queue. Use key as priority. Dequeue operation gets the highest priority element. Such priority queue is used in high performance graph algorithms
   1. Prim’s algorithm for minimum spanning tree
   2. Dijkstra’s algorithm for shortest path

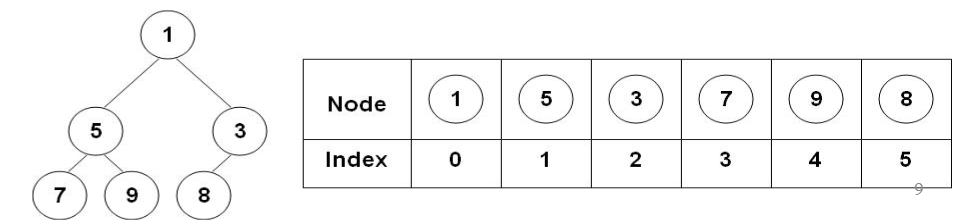
**Binary heaps**

Binary heap is a heap, with a complete binary tree structure

A complete binary tree has all levels except the last one fully filled. All nodes in the last level are left filled.

**Array representation of binary heaps**

* A complete binary tree of n elements can be represented by an array of n elements
  + Order the complete binary tree node data elements in breadth-first and left-first order and put the elements into array in the same order. The root node has index 0.
  + The node of index I has children indices at 2i+1 and 2i+2, and a parent at index (i-1)/2
* The height of the complete binary tree is log n
* In case binary min-heap, the element at index 0 is the min element.



**Linked representation of binary heaps**

Use node structure for complete binary tree. Use count to represent the number of nodes in the binary heap.

Typedef struct btnode{

Int key;

Struct btnode \*left;

Struct btnode \*right;

} TNODE;

TNODE \*new\_node(int val);

Void insert(TNODE \*\*rootp, int \*count, TNODE \*x);

Void delete (TNODE \*\*rootp, int \*count)

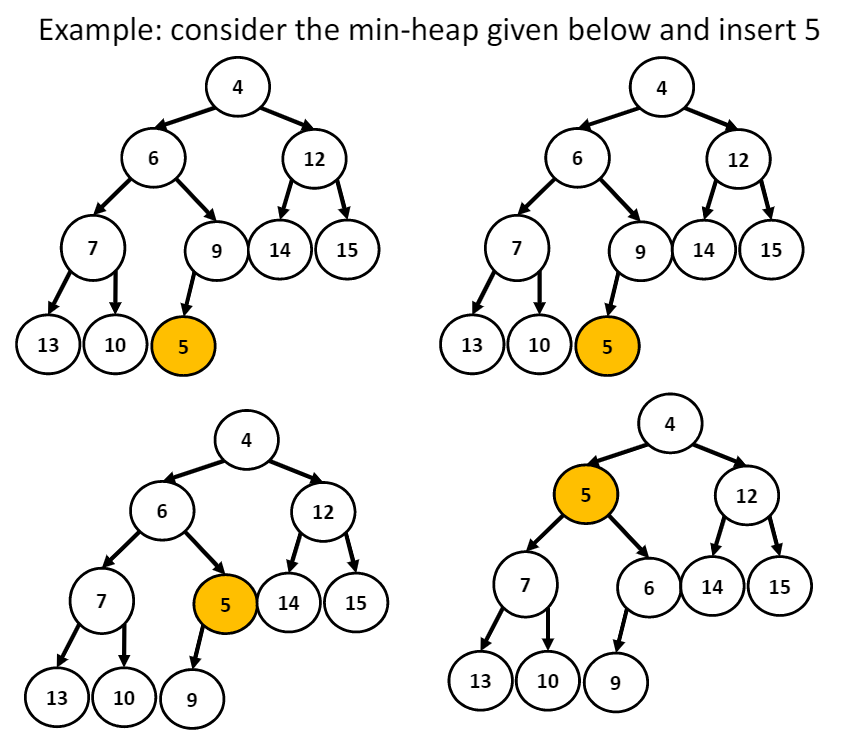
**Binary heap insertion**

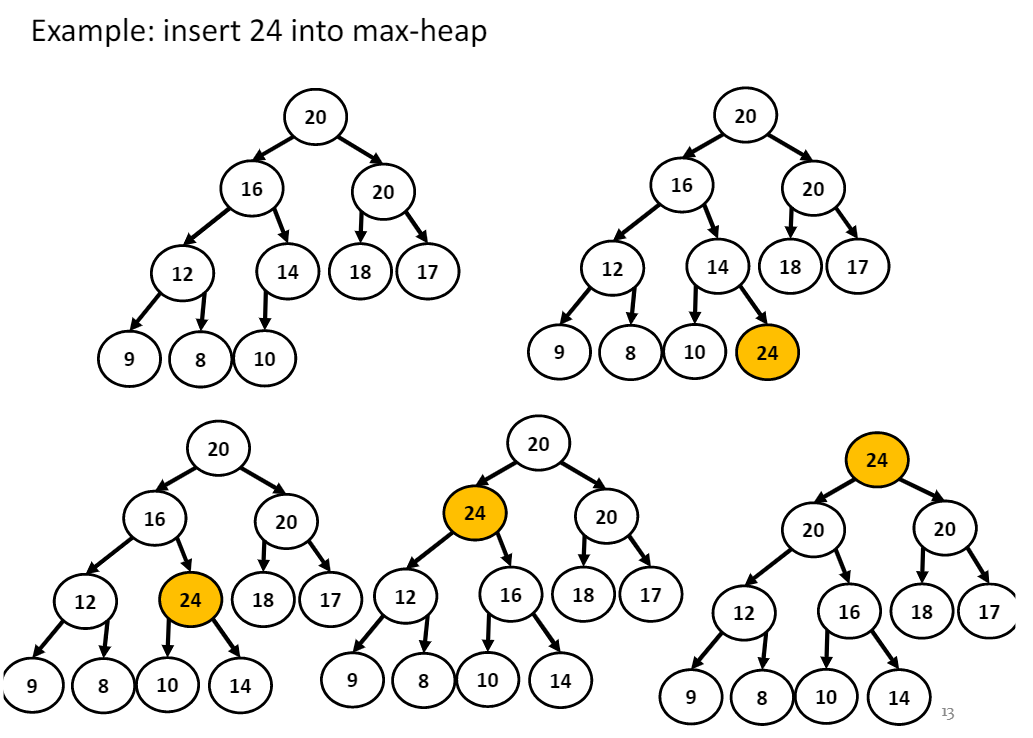
Algorithm: insert elements into binary min-heap H

1. Add the new element node to the bottom left
2. If the new node value is less than its parent, swap their values. Set the parent node as new node, goto step 2.
3. Stop

Step 2 is called heapify

Time: O(log n) Space: O(n)



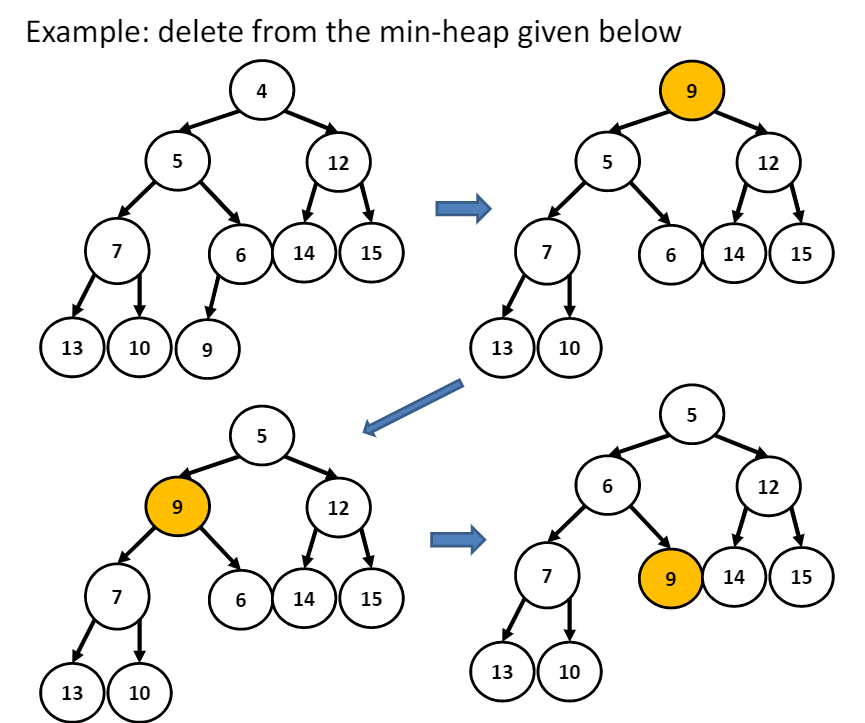


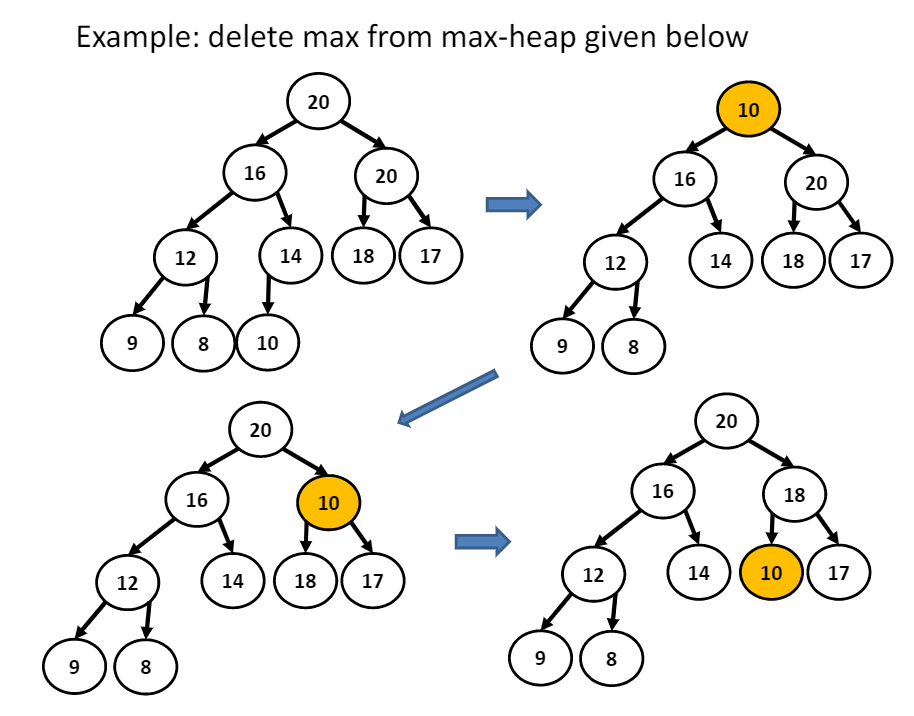
**Binary heap deletion**

Algorithm: delete an element from binary min-heap H delete-min

1. Replace the root node’s value with the last node’s value so that H is still a complete binary tree but not necessarily a heap
2. Delete the last node
3. Sink down the new root node’s value so that H satisfies the heap property. In this step, interchange the root node’s value with its child node’s value whichever is smallest among its children.

Time: O(log n) Space: O(n)

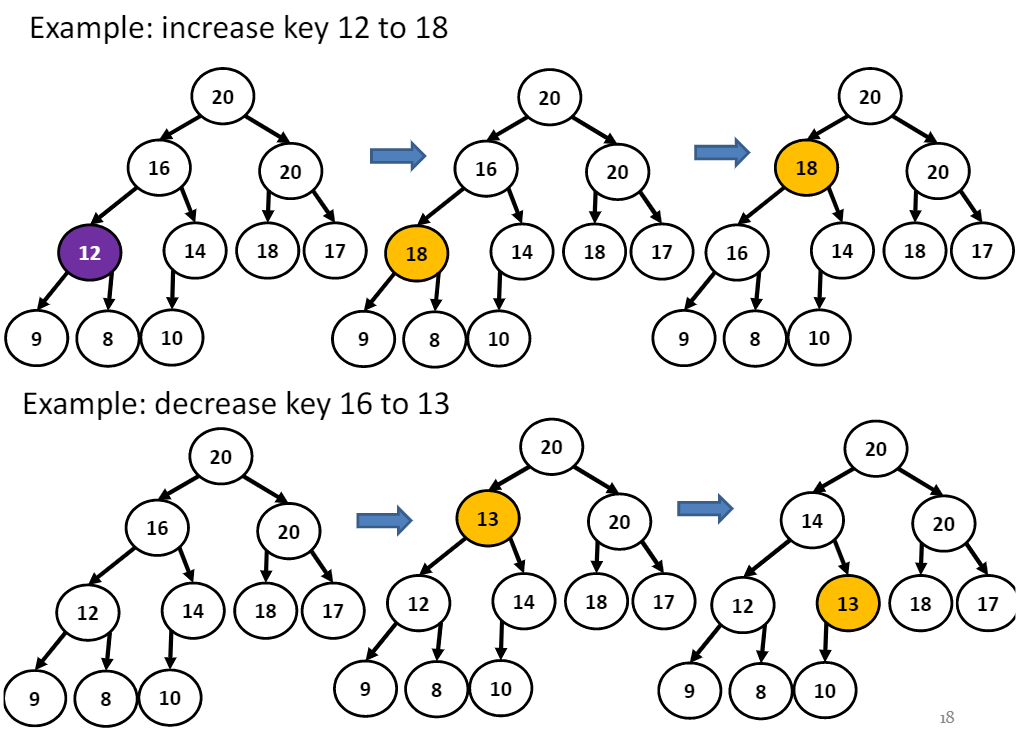




**Change key at given node**

Algorithm: change key value at given node index

1. Set the new value at the node
2. Heapify the node to satisfy the heap property



**Heap sort**

Given data x[i], i=0,…,n-1. Sort x[i] in increasing order in given array y[n].

1. Insert x[i] into binary max-heap h[n], i=0,…,n-1
2. Extract the max from heap and put at the next position of the sorted array y[] until heap becomes empty

Time: O(n log n) Space: O(1) using for loop

Given data in array x[n]. Sort x[i] in place of array x[n] (sort in place)

1. Convert x[n] to binary max-heap in place of x[n]
2. Extract the max from heap and put at the next backward position of x[n]

Time: O(n log n) Space: O(1)

