Comp 352

Tutorial Session 7: Priority Queue

Outlines

- Introduction
- Heap definition
- Heap types
- Heap methods
- Heap implementation
- Element insertion into a Heap
- Element removal from a Heap
- Heap Building

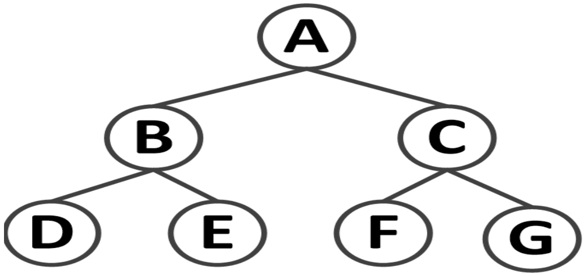
Introduction

- Problem statement:
  - The average running time of inserting an element to a sorted list is O(n), where the list has to be sorted after the insertion.
  - The average running time of removing an element from a sorted list is O(n).
- Objective:
  - Reducing the time complexity of inserting and removing process to/from a sorted list.
- Solution:
  - Priority Queue is proposed where the entries are stored in a binary tree.
  - The time complexity of the inserting and removing an element is O(log n); which is better than O(n).

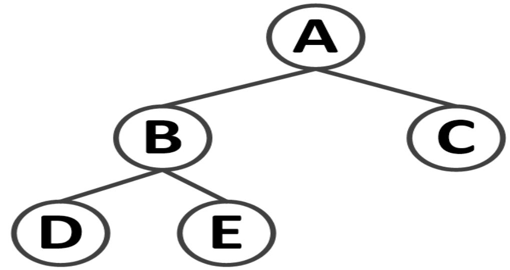
Priority Queues
Quick Overview (Heap Definition)

- ***Heap satisfies the Complete Binary Tree Property:***
  - A heap T with height h is a ***complete*** binary tree if levels 0,1,2,… ,h − 1 of T have the maximum number of nodes possible and in level h − 1, all the internal nodes are to the left of the external nodes and there is at most one node with one child, which must be a left child.

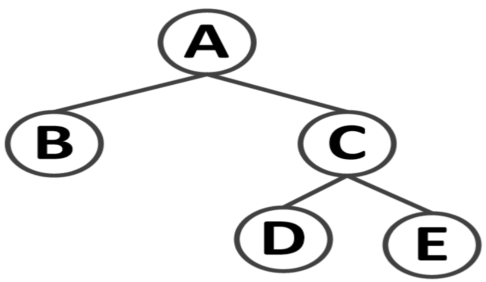
Priority Queues
Quick Overview (Heap Definition)



Full Complete Binary Tree



Complete Binary Tree

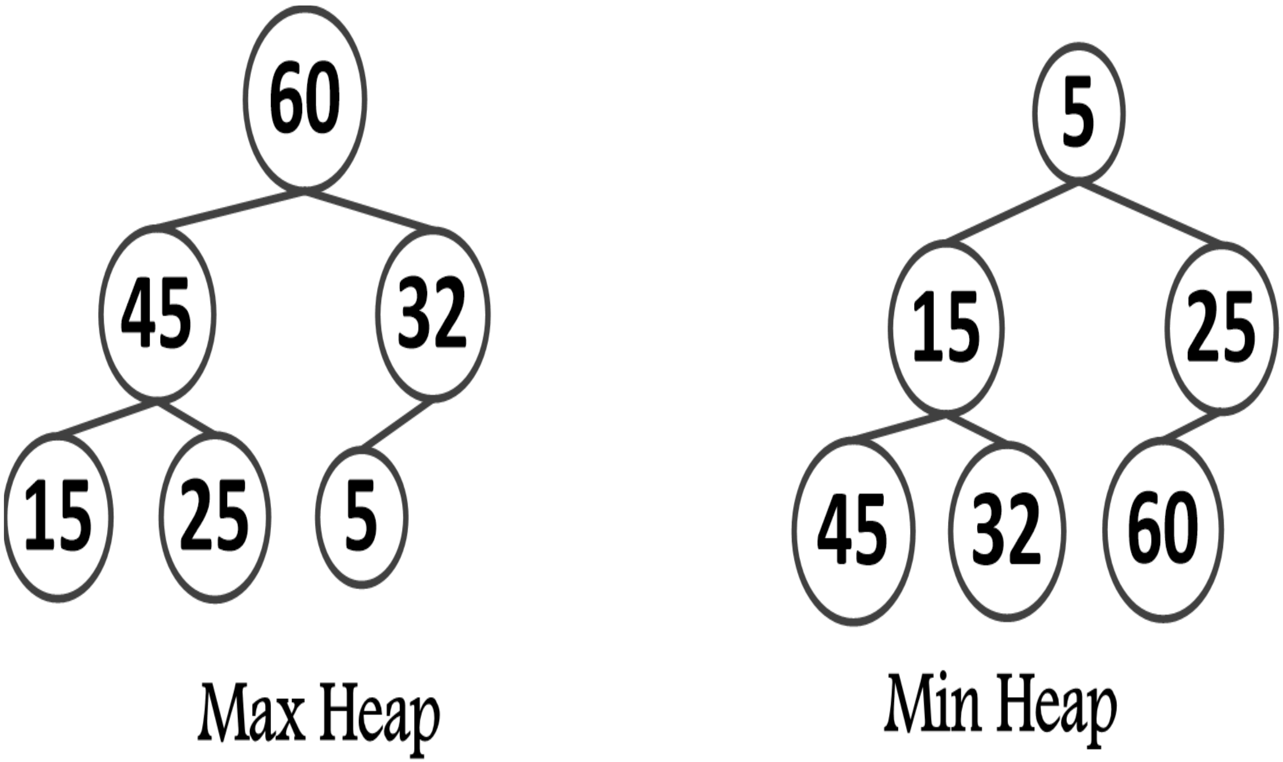


Not Full and Not Complete Binary Tree

Priority Queues
Quick Overview (Heap Types)

- ***Heap-Order Property (Priority Queue):***
  - **Min-Heap:** for every node *v* other than the root, the key stored at *v* is greater than or equal to the key stored at *v*'s parent.
  - **Max-Heap:** for every node *v* other than the root, the key stored at *v* is smaller than or equal to the key stored at *v*'s parent.

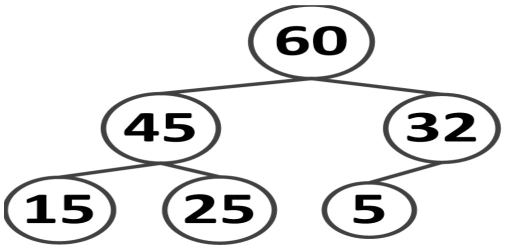
Priority Queues
Quick Overview (Heap Types)



Priority Queues
Quick Overview (Heap Methods)

- A ***Priority Queue P*** supports the following methods:
  - size(): Returns the number of entries in *P*.
  - isEmpty(): Checks whether P is empty or not.
  - min(): Returns the entry of the smallest key in Min-P.
  - max(): Returns the entry of the highest key in Max-P.
  - insert(k,x): Inserts an entry <key,value> into *P* and resorts P.
  - removeMin(): Returns and removes the entry of the smallest key from in a non empty min-*P*.
  - removeMax(): Returns and removes the entry of the highest key from in a non empty Max-*P*.

Priority Queues
Quick Overview (Heap Implementation)



1	2	3	4	5	6
60	45	32	15	25	5

PARENT (arr[*i*])
    return arr [floor(*i*/2)]
LEFT (arr[*i*])
    return arr[2*i*]
RIGHT (arr[*i*])
    return arr[2*i* + 1]

PARENT (15)
    return arr[floor(4/2)] = 45
LEFT (60)
    return arr[2*1] = 45
RIGHT (60)
    return arr[2*1 + 1] = 32

Priority Queues
Quick Overview (Element Insertion Algorithm)

Insertion into Max-heap

i = arr.size() + 1; // Assume the array is dynamic

While (*i != 1 & (*PARENT (*i*) < Entity (*i*)))

{

swap(PARENT (*i*),Entity (*i*))

(*i*) = floor (*i*/2)

}

Average Time complexity is O(log n)

Insertion into Min-heap

i = arr.size() + 1; // assume the array is dynamic

While (*i != 1 & (*PARENT (*i*) > Entity (*i*)))

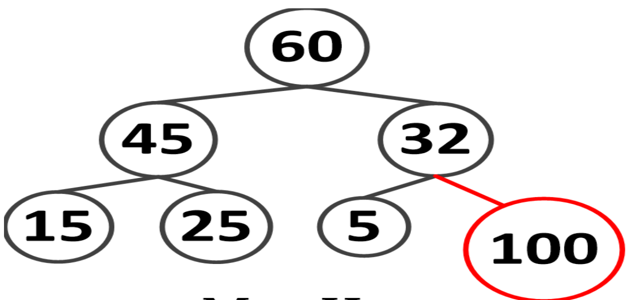
{

swap(PARENT (*i*),Entity (*i*))

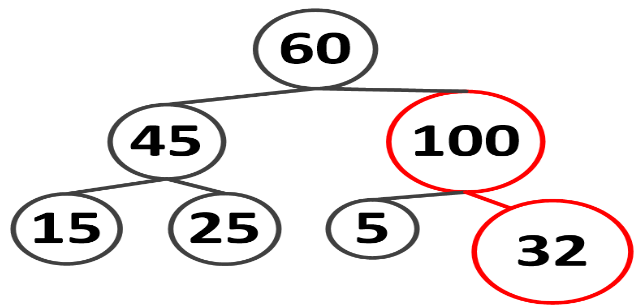
(*i*) = floor (*i*/2)

}

Priority Queues
Quick Overview (Insertion an Element)

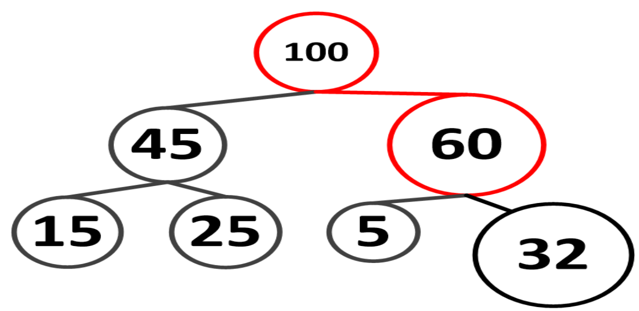


Max Heap



1	2	3	4	5	6	7
60	45	32	15	25	5	100

1	2	3	4	5	6	7
60	45	100	15	25	5	32



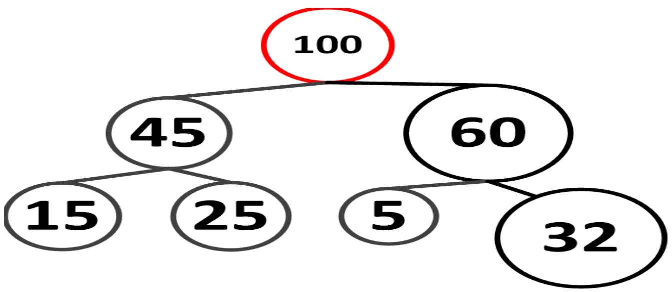
1	2	3	4	5	6	7
100	45	60	15	25	5	32

Priority Queues
Quick Overview (Remove Max Element)

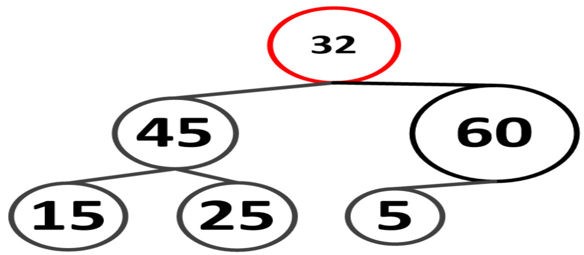
```
RemoveMax algorithm
arr[1] = arr[size_of_arr]
i = 1
while (arr[2 i] != Null or arr[2i + 1]=! Null)
{
If(arr[i] < max (arr[2 i] , arr[2i + 1]))
{
Swap (arr[i], max (arr[2 i] , arr[2i + 1]))
i = index_of (max (arr[2 i] , arr[2i + 1]))
}
else
Break;
}
```

The average time complexity is $O(\log n)$

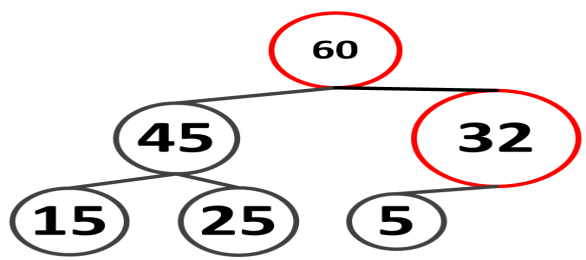
Priority Queues
Quick Overview (Remove Max Element)



Max Heap



1	2	3	4	5	6	7
100	45	60	15	25	5	32



1	2	3	4	5	6	7
32	45	60	15	25	5	

1	2	3	4	5	6	7
60	45	32	15	25	5	

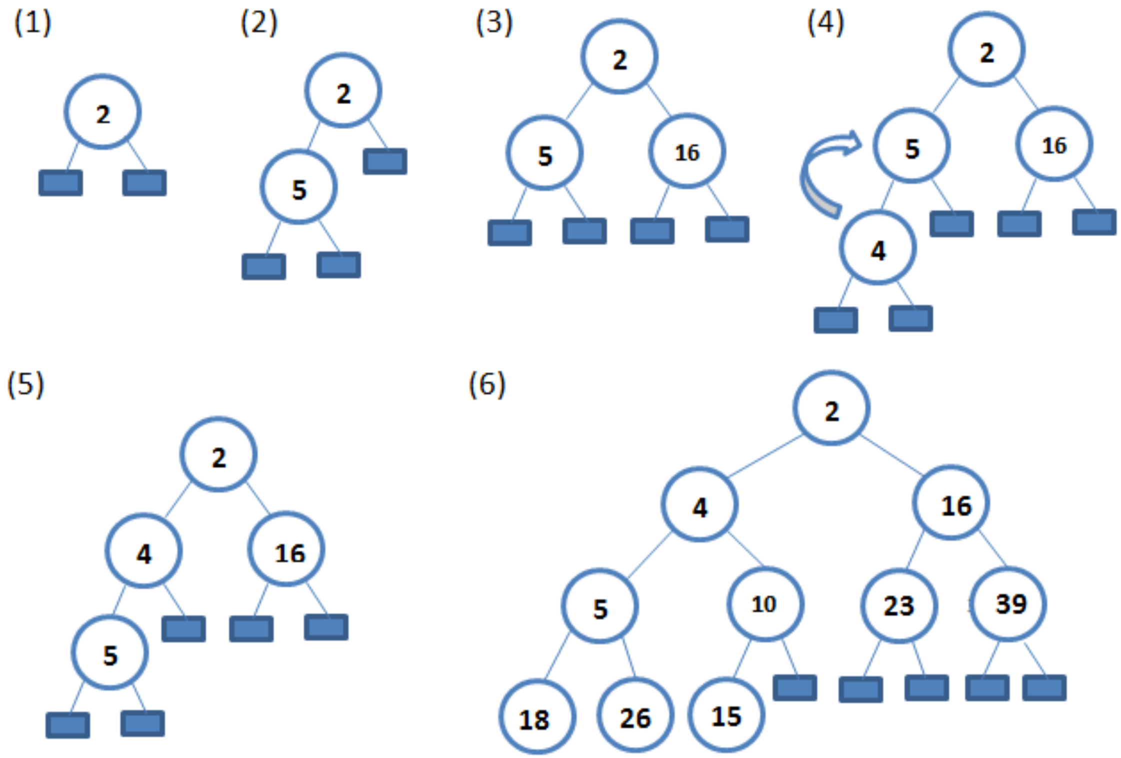
Priority Queues
Quick Overview (Heap Building)

Example : llustrate the execution of the heap-sort algorithm on the following sequence:

(2,5,16,4,10,23,39,18,26,15).

Show the contents of both the heap and the sequence at each step of the algorithm.

Priority Queues
Quick Overview (Heap Building)



Sequence contents at each of the above steps:
(1) (5; 16; 4; 10; 23; 39; 18; 26; 15)
(2) (16; 4; 10; 23; 39; 18; 26; 15)
(3) (4; 10; 23; 39; 18; 26; 15)
(4) (10; 23; 39; 18; 26; 15)
(5) (23; 39; 18; 26; 15)
(6) (39; 18; 26; 15)

Exercise 2

At which nodes of a max-heap can an entry with the largest key be stored?

Solution: the root node

At which nodes of a max-heap can an entry with the smallest key be stored?

Solution: the external nodes

At which nodes of a min-heap can an entry with the largest key be stored?

Solution: the external nodes

At which nodes of a min-heap can an entry with the smallest key be stored?

Solution: the root node