Heaps

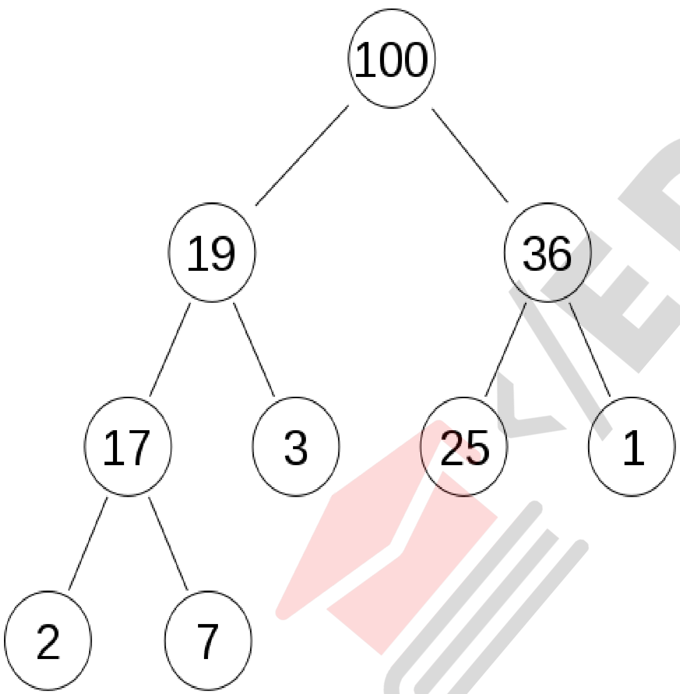
Definition of a heap

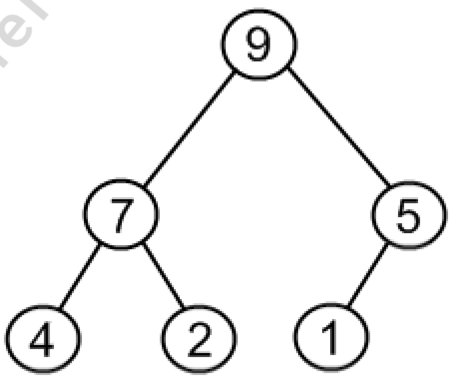
- A heap is a binary tree structure with the following properties:
  - The tree is **complete or nearly complete.**
  - The key value of each node is greater than or equal to the key value in each of its descendants.

  - Note: **whenever the term "heap" is used, it refers to max-heap**

Max heap

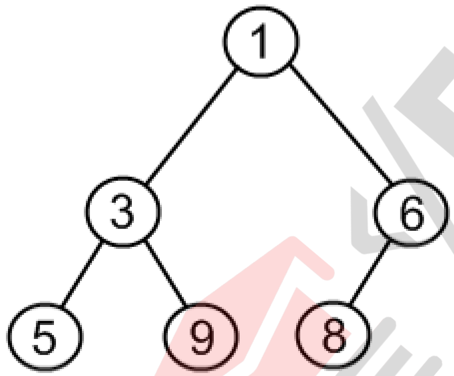
- A binary tree structure in which the key value in a node is greater than or equal to the key values in all of its subtrees.

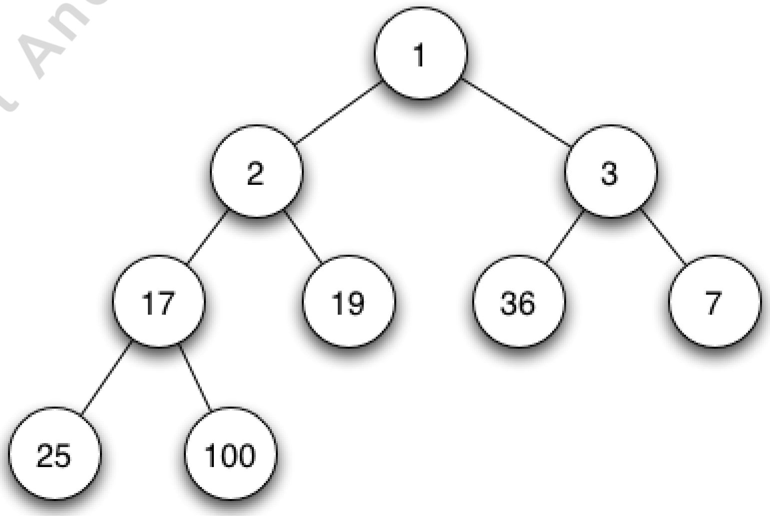




Min heap

- Min-heap:
  – A binary tree structure in which the key value in a node is less than or equal to the key values in all of its subtrees.





Basic heap algorithms

- Two basic maintainance operations are performed on a heap
  - Insert a node and
  - Delete a node
- Although it is a tree structure , it is **meaningless to traverse it, search it or print it out.**
- To implement the **insert and delete operations, two basic algorithms are required**
  - **Reheapup**
  - **reheapdown**

ReheapUp and ReheapDown operations

- Reheap Up operation
  - Reorders a “broken” heap by floating the last element up the tree until it is in its correct location in the heap.
  - In this the node must be placed in the last leaf level at the first empty position.
  - If the **new node’s key value>key value of the parent,** it is floated up the tree by exchanging the child and parent keys

- ReheapDown
  - Reorders a “broken” heap by pushing the root down the tree until it is in its correct position in the heap.
  - This algorithm is used mainly when the root is deleted from the tree.

reheapUp Algorithm

Algorithm reheapUp(int values[], int newNode)

1. if(newNode not the root)
    1. Parent= (newNode -1)/2
    2. If(values[newNode]>values[parent])
        1. Swap(values[newNode],values[parent])
        2. reheapUp(values[],parent)

reheapDown algorithm

Algorithm reheapDown(int values[], int root, int last)

1. [declare and initialize]

maxchild, rightchild, leftchild

leftchild=root * 2+1

rightchild=root * 2+2

2.if(leftchild <=last)

1. if(leftchild ==last)

maxchild=leftchild

else

1. if(values[leftchild] < values[rightchild])

. A. maxchild=rightchild

else

A. maxchild=leftchild

2. if(values[root]<values[maxchild])

1. swap(values[root],values[maxchild])

2. reheapDown(values [], maxchild,last)

Build Heap

Algorithm build_heap(heap , size)

1.Set walker to 1

2. Repeat until(walker <size)

    1. reheapUp(heap, walker)

    2. increment(walker)

Insert heap

Algorithm insertHeap(heap, last, data)

1. If (heap full)
   1. Return false
2. Increment last
3. Move data to last node
4. reheapUp(values[], last)
5. return last

Delete heap

Algorithm deleteHeap(heap, last,dataout)

1. if(heap empty)
    1. return false
2. Set dataout=root data
3. Move last data to root
4. Decrement last
5. reheapDown(values[], 0,last)
6. return true

Heap Sort

Algorithm heap_sort(A)

1. build_heap(heap,size)
2. Repeat while(n >=0)
    1. Swap(A[0],A[i])
    2. n=n-1
    3. reheapDown(A[],0,n)