Unit- III

Priority Queue (Heaps)

Priority Queues

→ Basic Model of priority Queue

→ Simple Implementations of priority Queue

→

Basic Model of priority Queue

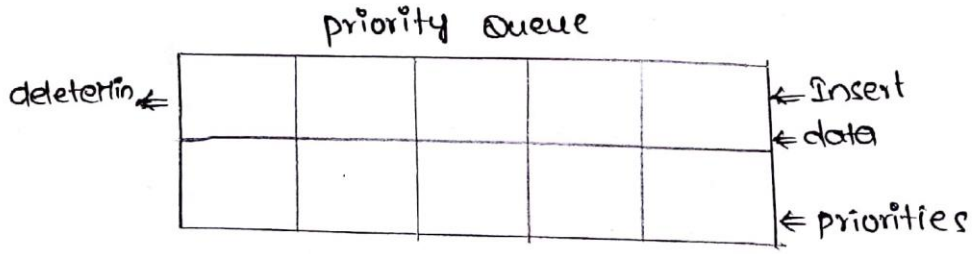
A Priority Queue performs Two operations

1) Insert
2) deletion

→ A priority Queue is a special Queue Based data structure in which the elements are inserted in order as they arrive along with the priority and the elements will be deleted Based on the priority

→ The Basic Model of priority Queue can Only perform two operations they are Insert, deleteMin.

Structure of Basic Model of priority Queue



Insert :- This operation inserts a new element into the priority queue along with priority

DeleteMin :- This operation finds and deletes Minimum element first from the priority Queue

→ The Basic Model priority queue unable to perform certain operations like deleteMax, getHighest priority, get lowest priority etc--- So that later the Basic Model priority Queue has been Implimented with new operations

Types of priority queues

The priority Queues have been divided into the two types 1) Max priority queue

2) Min priority queue

Max priority Queue

In the Max priority queue, the elements are Inserted in the order as they arrive along with priority and the maximum element will be deleted first from the priority queue

	0	1	2	3
Data	60	90	150	100
priorities	1	2	4	3

deletion Order :-
150 100 90 60

Min priority Queue

In the Min priority queue, the elements are Inserted in the order as they arrive along with priority and the minimum element will be deleted first from the priority queue.

	0	1	2	3
Data	10	20	30	15
priorities	5	4	3	4

FCFS

Deletion order :- 10 15 20 30

Data	50	80	50	90
priorities	5	4	5	3

Deletion order :- 50 50 80 90

Simple Implementations of priority queue

Usually the priority queues can be Implemented into four different ways

1) Arrays
2) Linked lists
3) BST
4) Binary Heap

1) Arrays Implementation of priority Queue :-

In the Arrays the elements of priority Queue can be Inserted in sequential order and deletion can be performed based on priority

Ex:- Max priority Queue

Arrival	Element	priority
1	1999	2
2	3998	5
3	2997	3
4	1996	1
5	3995	4

priority - Queue

Arrays representation

	1	2	3	4	5
Indexes ⇒	1999	3998	2997	1996	3995
	2	5	3	1	4

Insertion order :- 1999 3998 2997 1996 3995

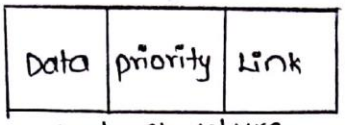
Deletion order :- 3998 3995 2997 1999 1996

Time complexity

Insertion ⇒ $O(1)$

Deletion ⇒ $O(n)$

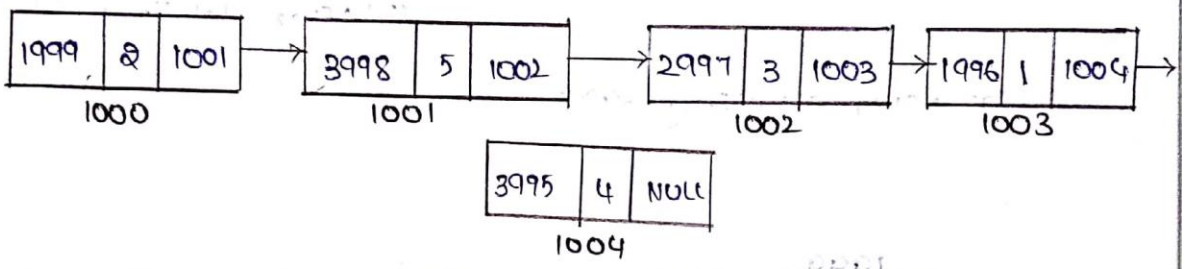
Linked list Implementation of priority Qu



Node structure

→ In this Implementation, especially we use doubly linked list in which every node contains three fields such as Data, priority, link fields

Linked List representation

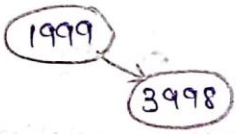


Insertion order :- 1999, 3998, 2997, 1996, 3995

3) Binary Search Tree Implementation of priority Queues

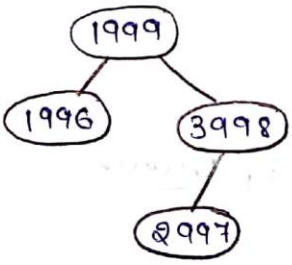
Insertion :-

(a) Insert 1999

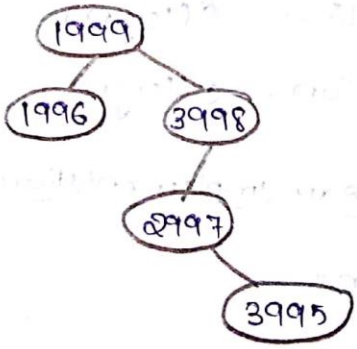


(b) Insert 3998

(c) Insert 2997



(d) Insert 1996



(e) Insert 3995

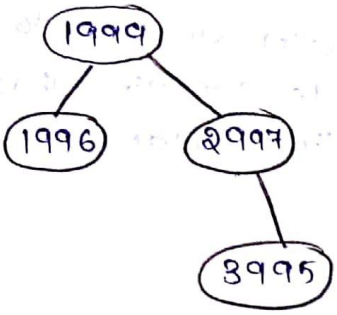
Deletion

1) Deleting the leaf node

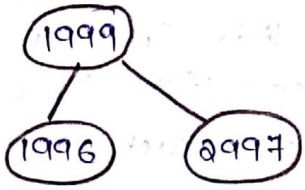
2) Deleting the node having only one child

3) Deleting the node having two childs

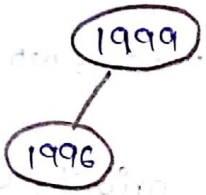
Deletion



(a) After deleting 3998



(b) After deleting 3995



(c) After deleting 2997

(d) After deleting 1999

(e) After deleting 1996

Time complexity

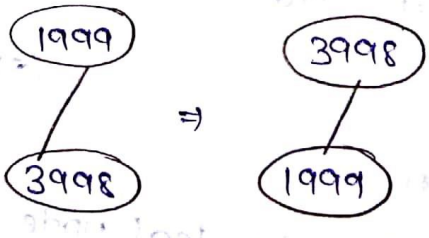
Insertion - O(log n)

deletion - O(log n)

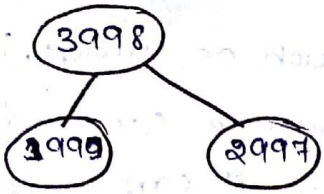
Binary heap implementation of priority Queue

Insertion:

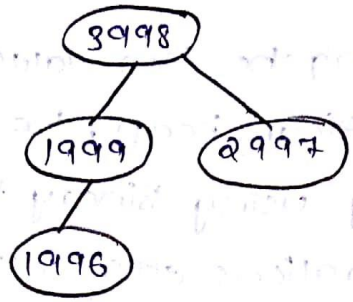
a) Insert 1999



b) Insert 3998

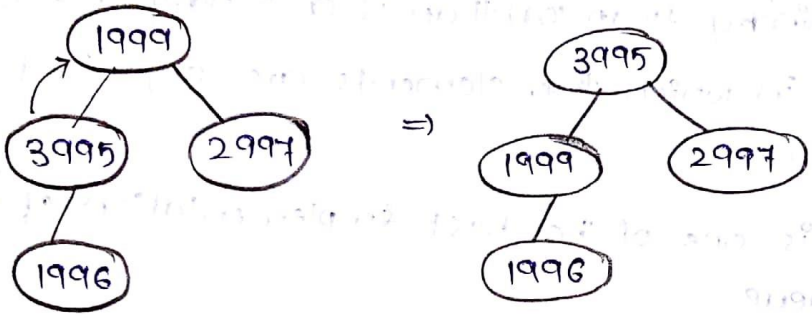


c) Insert 2997

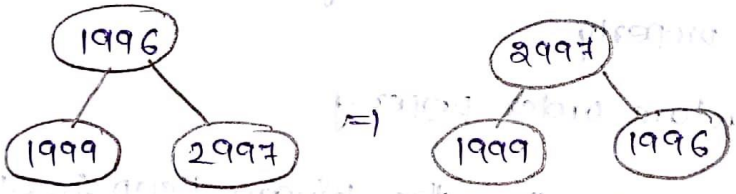


(d) Insert 1996

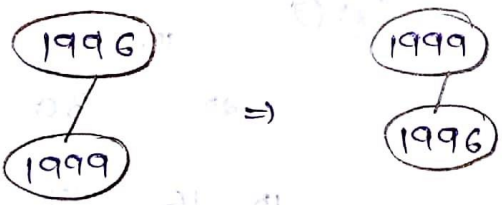
Deletion



a) After deleting 3998



b) After deleting 3995



(c) After deleting 2997

(d) After deleting 1999

Time complexity

1) Insertion - O(logn)

2) delelion - O(logn)

e) After deleting 1996

Note:- Among the four datastructures such as arrays, linked list, BST, Binary heap. the priority queue can be implimented efficiently using Binary Heap because it can perform more operations efficiently than the remaining data structures

Binary Heap

A Binary Heap (or) Heap is a Binary Tree based datastructure in which the elements are organised in Hirarechal order

→ Binary Heap is one of the best implementation of the priority queue.

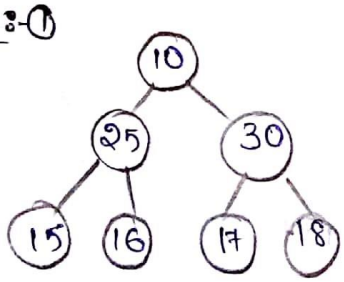
→ usually, the Binary Heap follows or posses two properties

1) Shape (or) structure order property
2) Heap order property.

Shape (or) structure order property

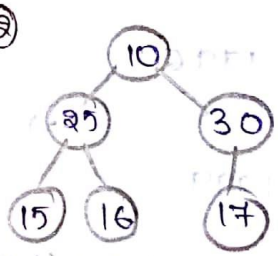
→ This property states that the Binary Heap should be either Complete (or) almost Complete Binary Tree.

Ex:-①



Complete Binary Tree

Ex:-②



Almost complete Binarytree

Heap order property

→ This property states that Every [Parent] node in the Heap is [less than (or) Equal to] (or) [greater than (or) Equal to] its children

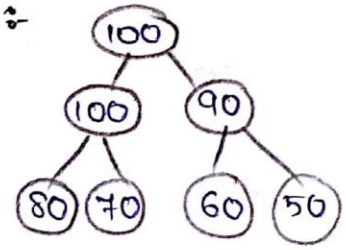
→ Depends upon the heap property, the Binary Heaps are classified into two types they are

1) Max Heap
2) Min Heap

Max Heap

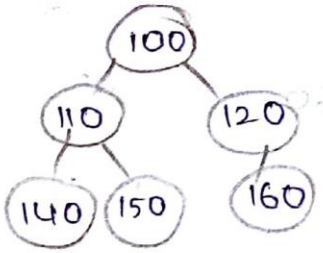
In the Max Heap Every parent node is greater than or Equal to its children

Eg:-



Min Heap

In the Minheap Every parent node is less than or Equal to its children.



Basic operations of Binary Heap

→ The Binary Heap has Basically two operations

1) Insert
2) Delete

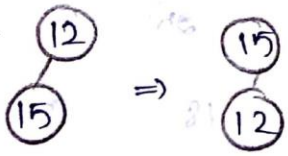
Insert :- this operation inserts a new element into the heap

Ex :- construct a Max heap using following elements

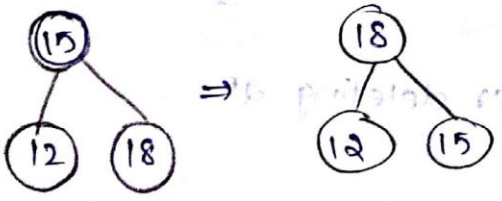
12, 15, 18, 25, 30, 60

Step① :- 12

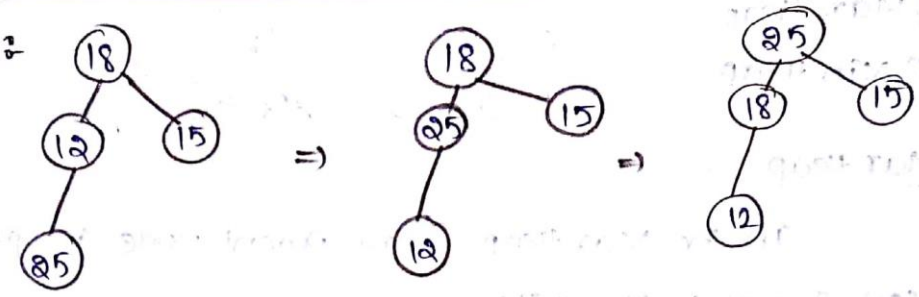
Step② :-



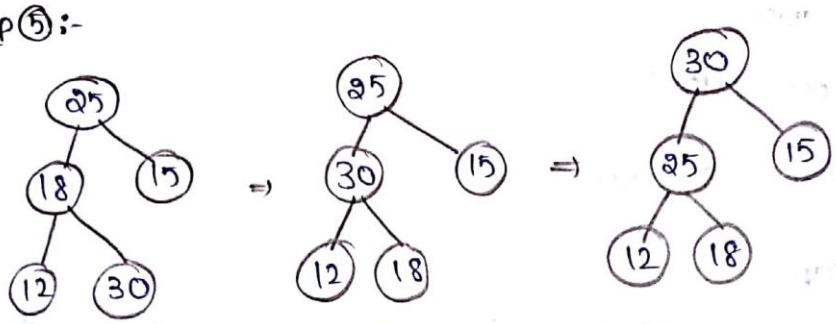
Step③ :-



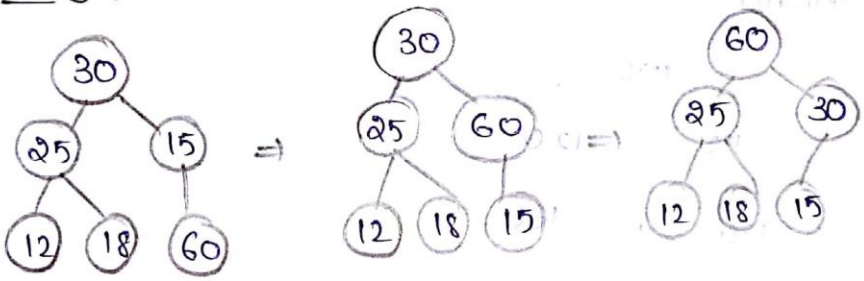
Step④:-



Step⑤:-



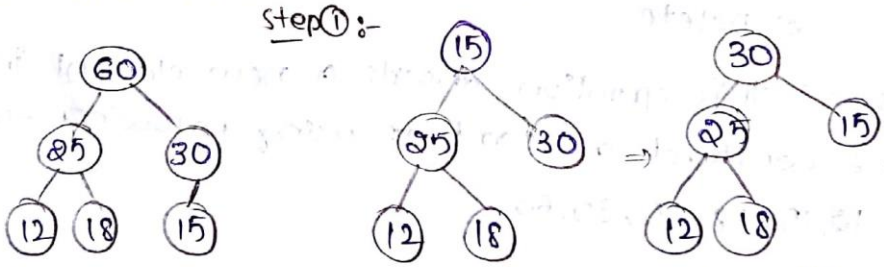
Step⑥:-



Delete :-

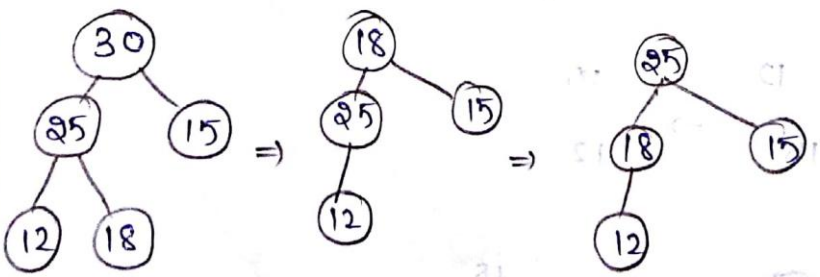
This operation deletes the ~~root~~ Specific element by replacing with last element in the binary heap.

Step①:-



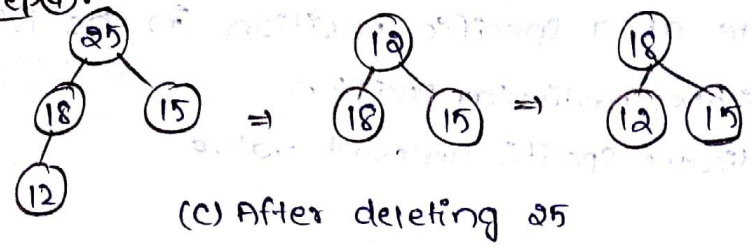
(a) After deleting 60

Step②:-



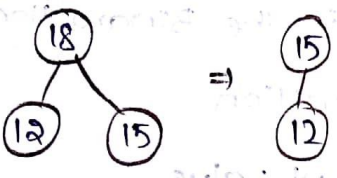
(b) After deleting 25

Step ④ :-



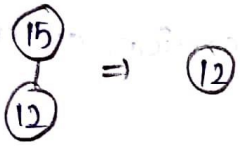
(c) After deleting 25

Step ⑤ :-

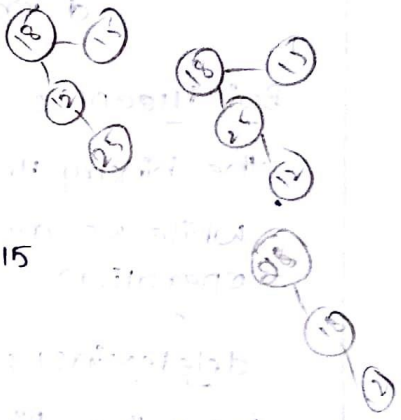


(d) After deleting 18

Step ⑤ :-



(e) After deleting 15



Step ⑥ :-

empty

(f) After deleting 12

other operations of Binary Heap:

Binary Heap has some other operations they are 1) getMin()

2) getMax()

3) Increase key (p, d)

4) Decrease key (p, d)

5) Build Heap()

6) deleteMin()

7) deleteMax()

8) display()

getMin() :- This operation returns minimum element in the binary heap.

getMax() :- This operation returns maximum element in the Binary heap

Increase key (p, d) :- This operation increases the value

by some value at a specific position in the Binary Heap.

where, p indicates particular position
d indicates specific amount value

(p, d)

Decreasekey (p,d) :- This operation decreases the value by some value at a specific position in the Binary Heap.

where, p indicates particular position
d indicates specific amount value

BuildHeap() :- This operation can be used to reconstruct the binary Heap inorder to obtain Maxheap (or) Minheap while we are performing either insertion or deletion operation

deleteMin() :- This operation deletes minimum element from the Binary Heap.

deleteMax() :- This operation deletes maximum element from the Binary Heap.

Display:- this operation displays all the elements in the Binary heap

Program to illustrate Binary Heap

```
#include <iostream.h>
# include <conio.h>
int heap[100], n=0;
int insert(int,int);
int delet(int);
void display();
void main()
{
  clrscr();
  int choice, num;
  do
  {
    cout<<"\n 1. insert() "<<endl;
```

```
cout<< " 2. delete ()"<<endl;
cout << "3. display()" <<endl;
cout <<" 4. Exit ()"<<endl;
cout << "\n Enter your choice :";
 cin>> choice;
 Switch (choice)
 {
  case1 : cout <<"\n Enter the element to be inserted to
                    the heap:";
    cin>>num;
    insert (num, n);
     n=n+1;
      break;
      case 2: cout << "\n Enter the element to be deleted
                    from the heap :";
       cin>>num;
       delet (num);
        break;
        case 3:
         display();
         break;
         case4 : cout << "Exit" <<endl;
          break;
          default:
          cout <<" enter a valid choice :";
          }
          }
           while (choice != 4);
            getch();
          }
   // Insertion
  int insert (int num, int loc)
  {
    int parent;
     while (loc >0)
     {
       parent = (loc-1) /2;
        IF (num <= heap (parent])
```

```
{
  heap[loc]=num;
  return 0;
 }
  heap[loc]=heap[parent];
  loc=parent;
 }
  heap[loc]=num;
 }
// deletion
 int delet(int num)
 {
  int left, right, i, temp, parent;
  for(i=0; i<num; i++)
  {
   if(num==heap[i])
   break;
  }
  if(num!=heap[i])
  {
   cout<<"the element"<<num<<"is not found in
          the heap"<<endl;
   return 0;
  }
   heap[i]=heap[n-1];
   n=n-1;
  /* parent=(i-1)/2;
   if(heap[i]>heap[parent])
   {
    insert(heap[i], i);
    return 0;
   } */
```

```
left = 2*i+1;
right = 2*i+2;
while(right<n)
{
if (heap[i]>=heap[left] && heap[i]>=heap[right])
 return 0;
 if(heap[right]<=heap[left])
 {
   temp=heap[i];
    heap[i]=heap[left];
    heap[left]=temp;
    i=left;
  }
 else
 {
   temp=heap[i];
    heap[i]=heap[right];
    heap[right]=temp;
     i=right;
   }
     left=2*i+1;
     right=2*i+2;
    /* if(left==n-1)&&heap[i]
     {
       temp=heap[i];
        heap[i]=heap[left];
        heap[left]=temp;
     }*/
    }
   }
    // display
     void display()
```

```
{
  int i;
  if (n==0)
  {
  cout<<"\n heap is empty"<<endl;
  }
  for (i=0; i<n; i++)
  {
    cout<<heap[i]<<" ";
  }
}
```

Output :-

1. Insert()
2. delete()
3. display()
4. exit()

Enter your choice : 1

Enter the element to be inserted to the heap : 30

1. Insert()
2. delete()
3. display()
4. exit()

Enter your choice : 1

Enter the element to be inserted to the heap : 50

1. Insert()
2. delete()
3. display()
4. exit()

enter your choice : 1

enter the element to be inserted to the heap : 40

1. Insert()
2. delete()
3. display()

4. Exit

Enter your choice: 1

Enter the element to be inserted to the heap: 20

1. Insert()
2. delete()
3. display()
4. Exit()

Enter your choice: 3

50 40 30 20

1. Insert()
2. delete()
3. display()
4. Exit()

Enter your choice: 2

Enter the element to be deleted from the heap: 40

1. Insert()
2. delete()
3. display()
4. exit()

Enter your choice: 3

50 30 20

1. Insert()
2. delete()
3. display()
4. exit()

Enter your choice: 4

Exit()

Binomial queue or heap:-

A Binomial queue is an extension of Binary heap. the Binomial queue is able to perform union or merge operation between two binomial queues very fast together with the other operations provided by the binary heap.

Structure of Binomial Queue

The Binomial queue can be structured or formed by a set of Binomial trees. A binomial tree is a general tree with special shape and also has certain properties

→ usually, the Binomial queue can be denoted by 'H' and binomial tree can be denoted by B_h where h represents height of the binomial tree

properties of Binomial tree

1) The Binomial tree (B_h) of height h has exactly 2^h no of nodes

ex:- The Binomial tree B_0 has $2^0 = 1$ node
B_1 has $2^1 = 2$ nodes
B_2 has $2^2 = 4$ nodes

2) There is atmost one Binomial tree for each height including height = 0

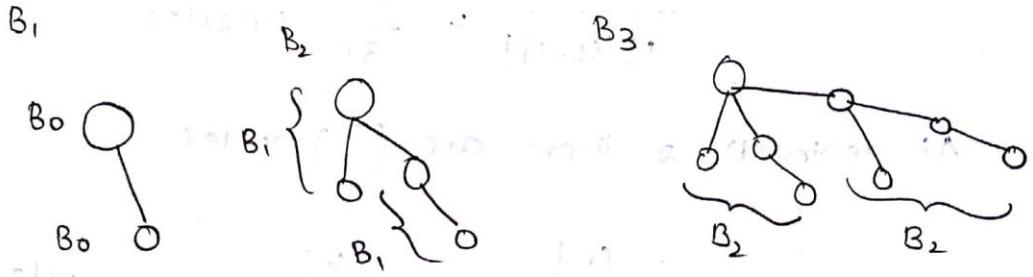
ex:-

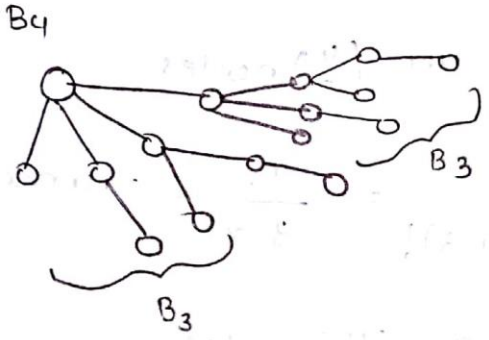
B_0 (Node)

3) The Binomial tree (B_h) can be formed by attaching one Binomial tree (B_{h-1}) to the root of another

Binomial tree (Bh-1).

Ex :-





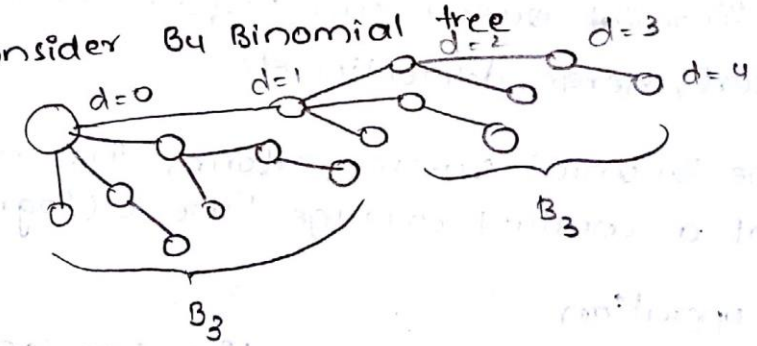
4) There are $\binom{h}{d}$ nodes at every depth(d) level of Binomial tree (Bh).

where d is the depth
h is height

$\binom{h}{d}$ is binomial coeffecient

$$\binom{h}{d} = \frac{h!}{d!(h-d)!}$$

Eg :- consider B4 Binomial tree



depth(d) = 0, 1, 2, 3, 4

At depth(d)=0, there are $\binom{4}{0}$ nodes

$$= \frac{4!}{0!(4-0)!} = \frac{4!}{1 \times 4!} = 1 \text{ node}$$

At depth(d)=1, there are $\binom{4}{1}$ nodes

$$= \frac{4!}{1!(4-1)!} = \frac{4!}{3!} = 4 \text{ nodes}$$

At depth(d) = 2, there are $\binom{4}{2}$ nodes

$$= \frac{4!}{2!(4-2)!} = \frac{4!}{2! \times 2!} = 6 \text{ node}$$

At depth(d) = 3, there are $\binom{4}{3}$ nodes

$$= \frac{4!}{3!(4-3)!} = \frac{4!}{3! \times 1!} = 4 \text{ nodes}$$

At depth(d) = 4, there are $\binom{4}{4}$ nodes

$$= \frac{4!}{4!(4-4)!} = \frac{4!}{4! \times 1!} = 1 \text{ node}$$

5) Every Binomial tree, usually follows minheap Order property.

Operations of Binomial Queue

→ The Binomial queue can perform various operations insert, merge, deleteMin etc---

→ The Binomial Queue performs the three operations at a constant average time $O(\log n)$

Insert operation

The Insert operation can construct a Binomial Queue by inserting a single node Binomial tree into the Binomial queue. If there are any two Binomial trees of same height in the Binomial queue then merge operation will be performed between them.

By attaching the larger root Binomial tree as a child to the smaller root Binomial tree

Ex :- construct a Binomial queue with the following elements.

12, 21, 24, 65, 16, 18

Step ① :- Insert 12

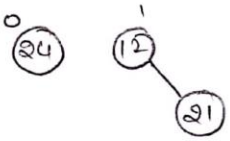
H_1 : (12)

Step ② :- Insert 21

H_1 : (21) (12) ⇒ (21) + (12) ⇒

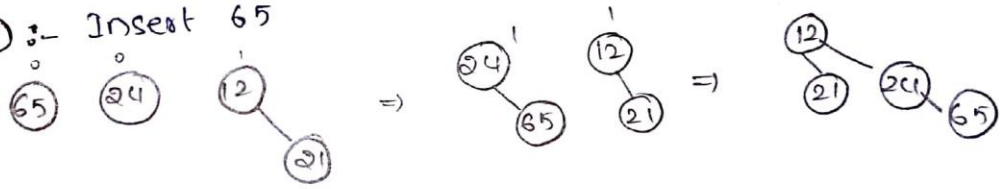
Step ③ :- Insert 24

H_1 :



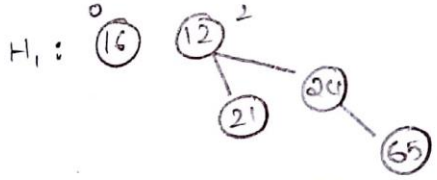
Step ④ :- Insert 65

H_1 :



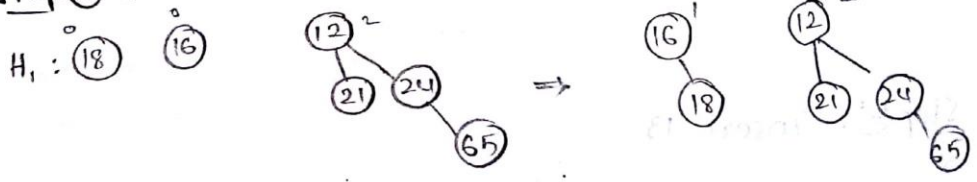
Step ⑤ :- Insert 16

H_1 :



Step ⑥ :- Insert 18

H_1 :



Ex :- construct a Binomial queue H_2 with following elements

23, 51, 24, 65, 14, 26, 13

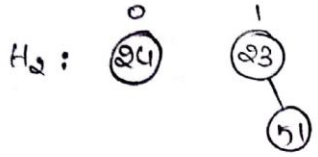
Step ① :-

H_2 : (23)

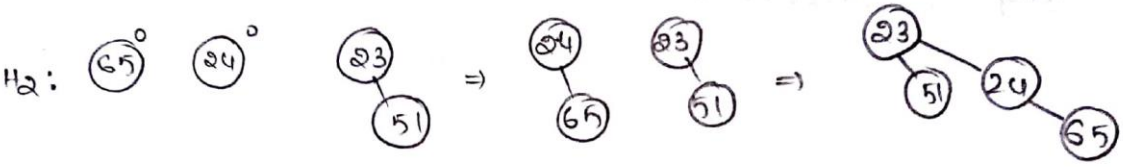
Step② :- Insert 51

H_2: 

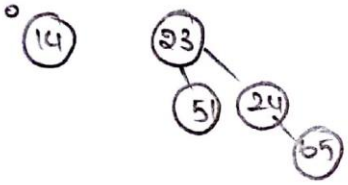
Step③ :- Insert 24

H_2: 

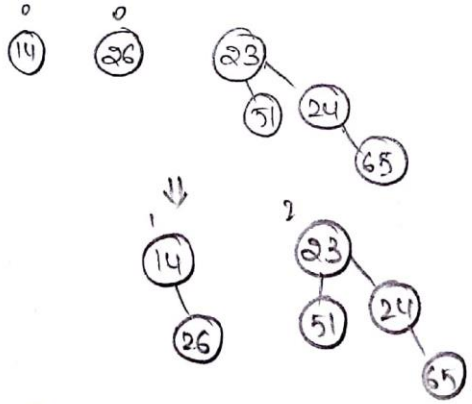
Step④ :- Insert 65

H_2: 

Step⑤ :- Insert 14



Step⑥ :- Insert 26



Step⑦ :- Insert 13

Merge operation

→ The Merge Operation not Only merges the Binomial Trees of Same height in a Single Binomial Queue but also merges Binomial Trees of same height in two different Binomial Queues.

→ Merging two different Binomial queues is very Easy operation.

procedure steps

1) Add the corresponding Binomial trees of same height in two Binomial Queues and place the resulting binomial tree into new Binomial queue.

2) compare the binomial trees height from "0 to max height" in both binomial queues.

case(i) :- If neither binomial Queue has the binomial tree of same height then skip merge operation and moves to compare next height.

case(ii) :- If one binomial queue has a binomial tree of height 'h' and other does not have them leave it and place it into new binomial queue

case(iii) :- If there are two binomial trees of same height in both binomial queues then the larger root binomial tree is attached as child to the smaller root binomial tree and then place the resulting binomial tree into the new binomial queue.

3) Repeat the merge process untill the max height binomial trees are encountered in both binomial queues.

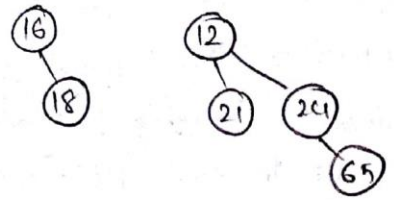
Ex :- consider two binomial queues H_1 & H_2 with six & seven trees are encountered in both binomial queues.

Let H_3 be the new Binomial queue to store the result of merge operation between H_1 & H_2

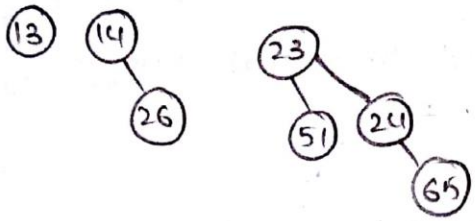
H_1 :- 12, 21, 24, 65, 16, 18

H_2 :- 23, 51, 24, 65, 14, 26, 13

H_1 :-



H_2 :-

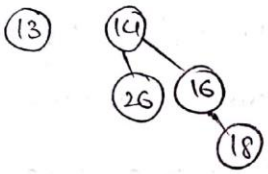


Step(1) :-

H_3 :-

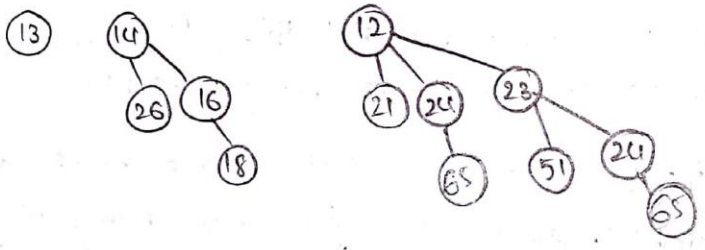
Step(2) :-

H_3 :-



Step(3) :-

H_3 :-

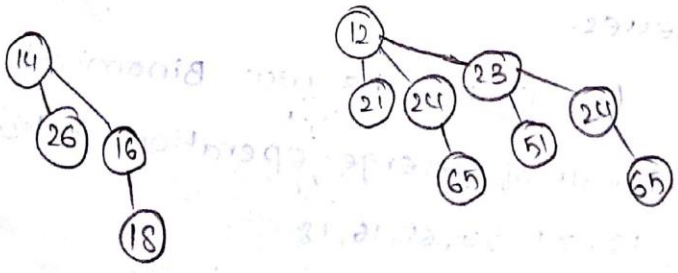


deleteMin operation

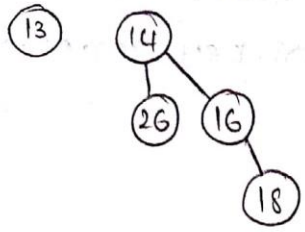
→ The deleteMin operation deletes the minimum element from the Binomial queue by breaking the original Binomial queue into two temporary Binomial queues and then merge operation will be performed between the temporary Binomial queues and the resultant Binomial trees are placed into original Binomial queue.

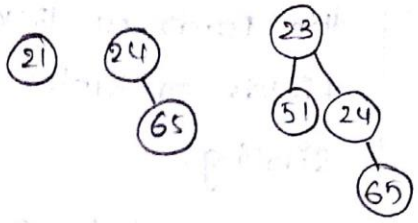
Eg:- let H_3 be original Binomial Queue and H_3', H_3'' are the two temporary Binomial Queues of H_3

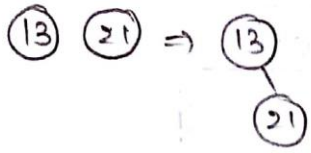
H_3 :

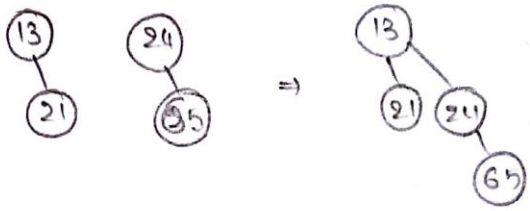


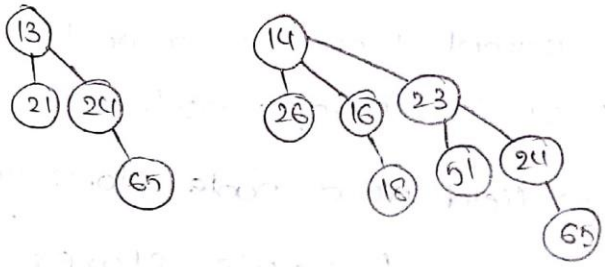
Step①:- delete 12

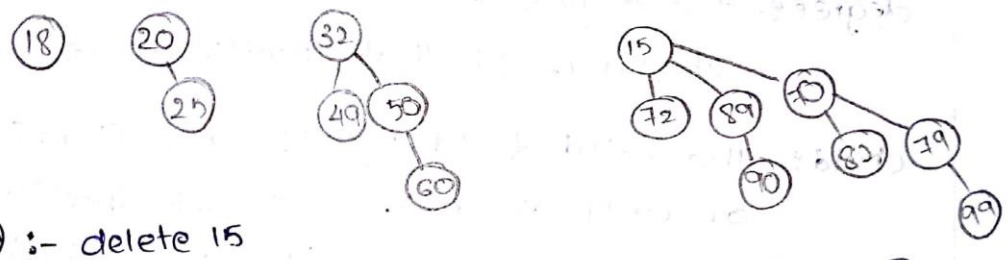
H_3' :- 

H_3'' :- 

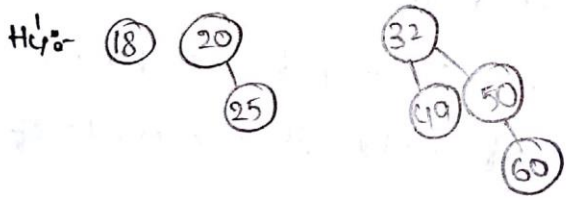
Step②:- H_3 :- 

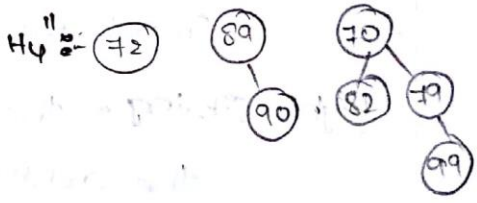
Step③:- H_3 :- 

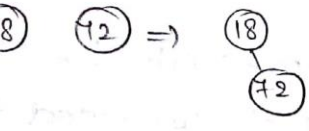
Step④:- 

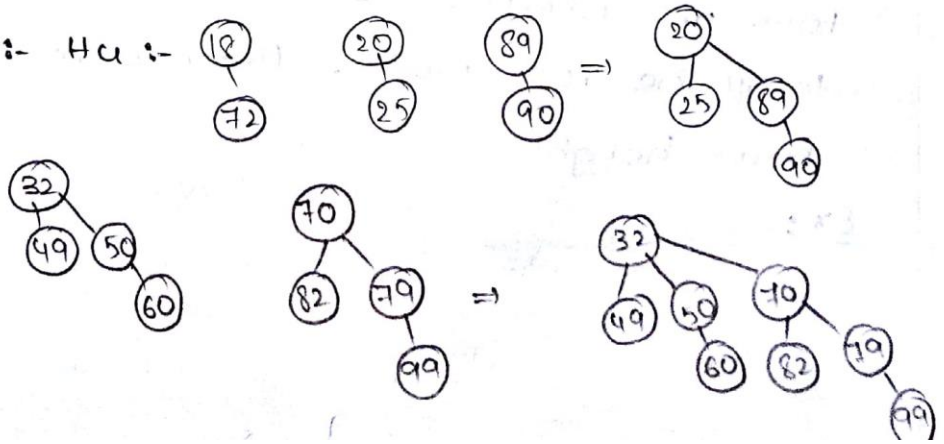
H_4 :- 

Step① :- delete 15

H_4' :- 

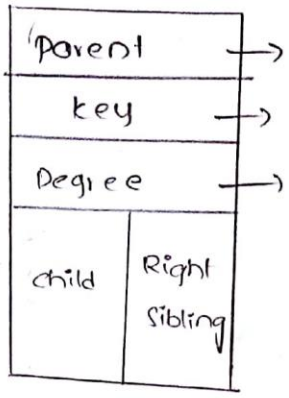
H_4'' :- 

Step② :- H_4 :- 

Step③ :- H_4 :- 

→ Usually the Binomial queue can be represented in the form of linked list where each node contains 5 fields or tuples such as parent, key, Degree, child, Right sibling.

Node Structure



Parent:- The parent field of a node points are stores the address of its parent node.

key:- the key field of a node stores the actual element

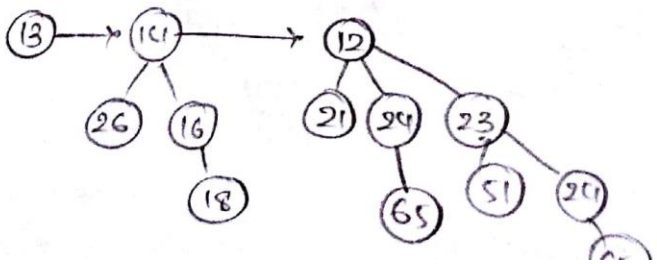
degree:- the degree of a node stores the total no. of children of that specific node.

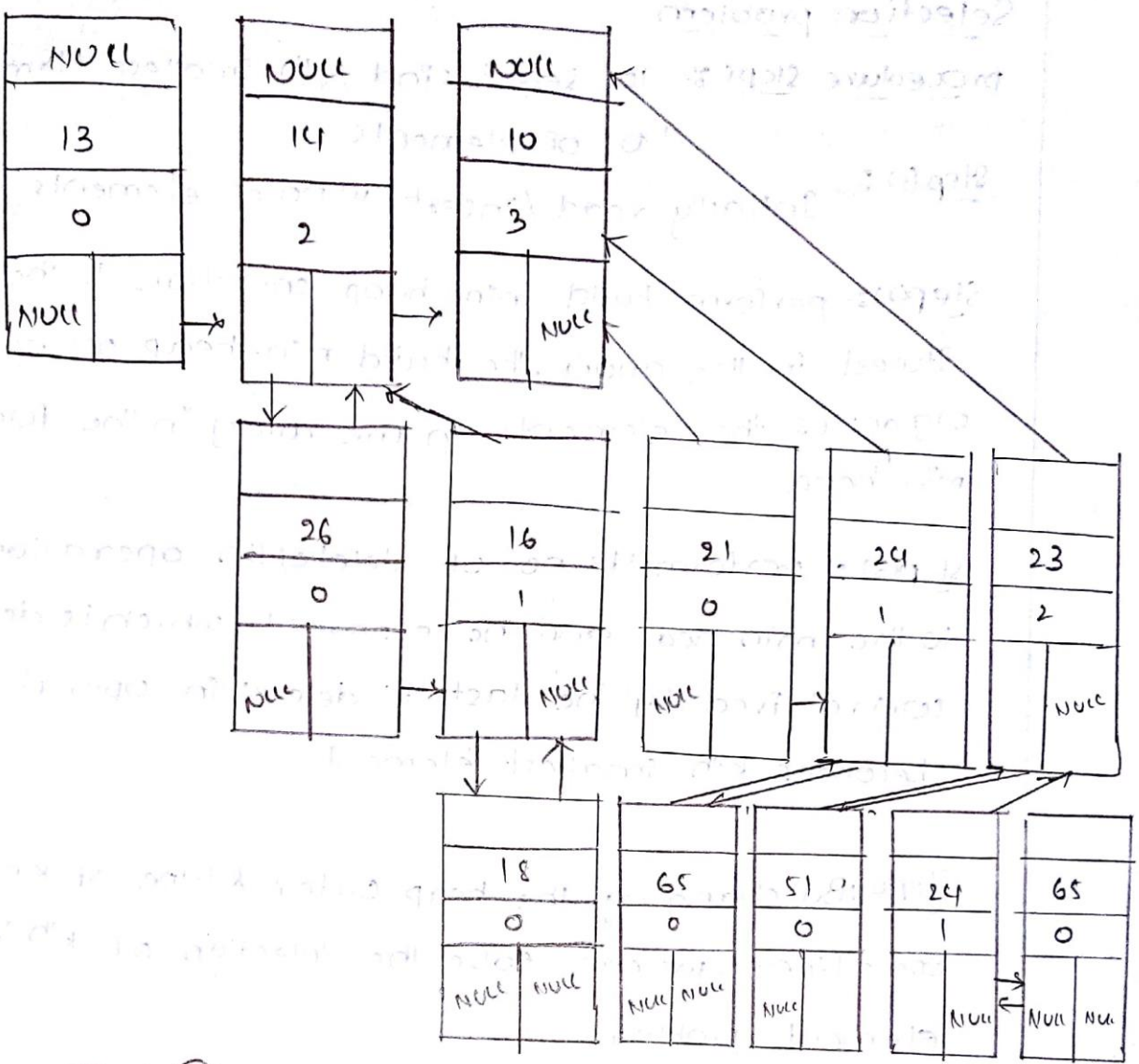
Child:- The child field points are stores the address of any of child node of specific node (most probably from the left child)

Right sibling:- the Right sibling field points or stores the address of right sibling node of specific node.

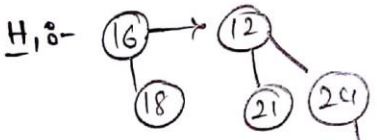
→ In order to represent the binomial queue in the form of linkedlist we need to make corrections to all the root nodes from zero height to max height.

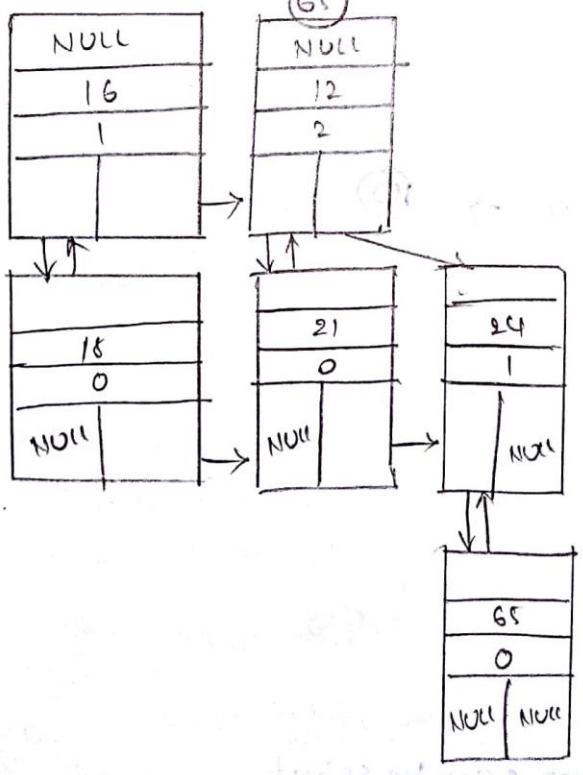
Ex:- H_3:





H.O:-





Applications of priority queue

In General, priority queue can be used in various applications there are 6 applications

1) Graph algorithm (Prims, kruskal)
2) heap sort
3) operating system
4) Huffman coding
5) selection problem
6) event simulation problem

Selection problem

procedure steps :- To select/find kth smallest element in 'N' of elements

Step(1) :- Intially read / Insert 'N' no of elements into array

Step(2) :- perform build min-heap operation to the elements stored in the array. the build min-heap operation organizes the elements in the array in the form of min-heap.

Step(3) :- perform 'k' no of deleteMin operations to the min heap and the element which is deleted (or) retrived by the last 'k' deleteMin operation becomes kth smallest element

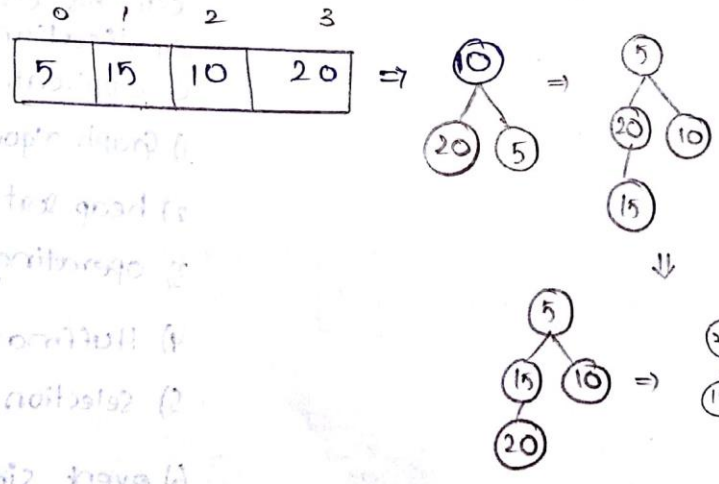
Step(4) :- By changing the heap order k type of k no. of operations. we can solve the selection of k^{th} largest element problem.

Example :-

N = 4

k = 2

10, 20, 5, 15



→ The selection problem can be solved

Time complexity O(blog(n))