a) Implementation Binomial Heap

i) Insert Function: Inserting a key into binomial heap

```
list < Node *> insert TreeinHeap(list <Node*>_heap, Node *tree)
 {
   list < Node * > temp;
   temp.push.back (tree);
   temp = unionBinomialHeap(*heap, temp);
    return adjust (temp);
     }

List<Node *>  removeMinFromTree(Node *tree)
{
list<Node *> heap;
Node  * temp  = tree -> child ;
 Node  * lo;
 while (temp)
  {
    lo = temp;
    temp= temp-> sibling;
    lo -> sibling = NULL;
    heap.push_front(lo);
   }
  return heap;
 }
list<Node *> insert ( list <Node*>_head, int key)
   {   Node  * temp = newNode(key);
       return  insertTreeinHeap(head , temp);
   }
```


Clf.

IRM19CS403
CHIRANJEEVI

ii) Node* getMin (list <Node*> _heap)

```
{
  list <Node*>:: iterator it = heap_begin();
   Node *temp = *it;
  while (it+1 != heap.end())
  {
   if ((*it)->data < temp->data)
        temp= *it
        it++;
    }
  return temp;
```

iii) list <Node *> ExtractMin (list <Node*> _heap)

```
{
list <Node*> new_heap, lo;
Node  *temp;
 temp= getMin(_heap);
 list< Node*> :: iterator it;
 it = _heap.begin();
 while(it!=_heap.end())
 { if(*it!=temp)
     {
       new_heap.push_back (*it);
     }
    it++;
  }
 lo: removeMinFromTree (temp);
 new_heap = UnionBinaryHeap (new_heap_lo);
  new_heap= adjust (new_heap);
  return new_heap;
}
```


Ch