Chandan CBagan
1BM18CS026

```
struct Node
{
  int data, degree;
  Node * child, sibbling, * parent;
}
list <Node*> insert (list <Node *>_head, int key)
{
   Node * temp = new node (key);
   return insertATreeInHeap (_head, temp);
}
Node * getmin ( list <Node*>_heap)
{  list<Node *>:: iterator it = _heap.begin();
   Node *temp = *it;
   while (it) = _heap.end())
   {  if (*it) → data < temp → data)
         temp = *it;
      it++;
   }
   return temp;
}
list <Node *> insertATree In Heap (list <Node *>_heap_
Node * tree)
{
   list <Node *> temp;
   temp.push_back (tree);
   temp = unionBinomialHeap (_heap, temp);
   return adjust (temp);
}
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 returnHeap(temp);
new_heap = unionBinomialHeap(new_heap, lo)
new_heap = adjust(new_heap)
return new_heap;
}

Node *mergeBinomialTrees(Node *b1, Node *b2)
{
    if (b1->data > b2->data)
        swap(b1, b2)
    b2->parent = b1;
    b2->sibling = b1->child;
    b1->child = b2;
    b1->degree++;
    return b1;
}

list<Node*> unionBinomialheap(list<Node*> l1,
                              list<Node*> l2)
{
    list<Node*> _new
    list<Node*>::iterator it = l1.begin();
    list<Node*>::iterator it = l2.begin();
while (it != l1.end() && it != l2.end())
{
    if (*it)->degree <= (*it)->degree)
    {
        _new.push_back(*it);
        it++;
    }
    else
        _new.push_back(*it);
        it++;
}}
```

```
while (it1 != l1.end())
{
    new.push_back(*it1);
    it1++;
}
else{
    new.push_back(*it2);
    it2++;
}
}
while (it1 != l1.end())
{
    new.push_back(*it1);
    it1++;
}
while (it2 != l2.end())
{
    new.push_back(*it2);
    it2++;
}
return new;
}
```