Snehita. J
1BM19CS109

```
struct Node
{
    int data, degree;
    Node *child, *s, *p;
};

Node* newNode (int key)
{
    Node *t = new Node;
    t->data = key
    t->degree =0;
    t-> child = t->p = t->s = NULL;
    return t;
}

list<Node *> insert ( list <Node*> _h, int key)
{
    Node *t = newNode( key);
    Return InsertATreeInHeap (_h, t);
}

list < Node *> InsertATreeInHeap ( list <Node *> _heap, Node* tree)
{
    list<Node* > t;
    t.push_back( tree);

    t= UnionBinomialHeap(_heap,t);
    return adjust ( temp);

}

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

Snehila

```
list<Node*> adjust (list <Node*> _heap)
{
    it1 = it2 = it3 = _heap.begin();
    while ( it1 != _heap.end())
    {
        if ( it2 == _heap.end())  it1++;
        else if ((*it1)->degree < (*it2)->degree)
        {   it1++;
            it2++;
            if ( it3!= _heap.end())  it3++;
        }
        else {   *it1 = mergeBin(*it1, *it2);
                 it2  = _heap.erase(it2);
                 if ( it3!= _heap.end())  it3++;
        }
       Return  _heap;
}
Node * getMin (list<Node*> _heap)
{   list<Node*> :: it = _heap.begin();
    Node  *temp = *it;
    while ( it != _heap.end())
    {     if ((*it)->data < temp->data)
                      temp = *it;
          it++;
    }
     Return  temp;
}
list<Node*> extractMin (list<Node*> _heap)
{
    while ( it != _heap.end())
    {
        if (*it != temp)
                  newheap.push_back(*it);

        it++;
    }
```

Snehita:

```
  lo = removeMinFromTree(temp);
 new_heap = unionBinomialHeap(new_heap, lo);
 new_heap = adjust(new_heap);
  return new_heap;
}
```