Ajay Nitter
1BM19CS006

Binomial Heap

```
struct Node {
    int data, degree;
    Node *child, *sibling, *parent;
};

Node* merge(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*> insert(list<Node*> head, int key) {
    Node* temp = new Node(key);
    return insertATreeinHeap(head, temp)
}

list<Node*> extractMin((list<Node*> heap) {
    list<Node*> new_heap, 10;
    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++;
    }
```

Ajay Mittur
IBM18CS006

Binomial Heap

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

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

list<Node*> insertATreeInHeap(list<Node*> heap,
                              Node* tree) {
    list<Node*> temp;
    temp.push_back(tree);
    temp = unionBinomialHeap(heap, temp);
    return temp;
}

list<Node*> removeMinFromTreeReturnBHeap(Node*
                                         tree)
    list<Node*> heap;
    Node *temp = tree->child, *lo;
    while(temp) {lo = temp; temp = temp->sibling;
    lo->sibling = NULL; heap.push_front(lo);}
    return heap; }
```