Akshay S Bharadwaj
1BM18CS011

ADS Lab - 9 Writeup (Binomial Heaps)

```
struct Node {
    int data, degree;
    node *child, *sibling, *parent;
};
node* *newNode(int data) {
    node *temp = new Node;
    temp -> data = data;
    temp -> degree = 0;
    temp -> child = temp -> parent = temp -> sibling = NULL;
    return temp;
}
list <node*> insertionoftree (list (node*)heap, node*tree)
{   list <Node*> temp;
    temp.push-back (tree);
    temp = unionofheap (heap, temp);
    return adjust(temp);  // rearranging the heap
}
list< node*> union ofheap (list<node*> l1, list(node*) l2)
{
    list <node*> new;
    list <node*>:: iterator it = l1.begin();
    list < node*> :: iterator ot = l2.beign(),
    while (it != l1.end() && ot != l2.end()) {
        if ((*it) -> degree <= (*ot) -> degree) {
            new. push-back(*it);
            it ++;
        }
        else {
            new. push_back (*ot);
            ot ++;
        } }
        while (it != l1.end()) {
            new.push_back(*it);
            it++;
        }
        while (ot != l2.end()) {
            new.push_back(*ot);
            ot++;
        }
~~}~~   return new;
    }

list<node*> insert (list<node*> head, int data){
    node* *temp = newNode(data);
    return insertion of tree (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*> extractMin (list<node*> heap) {
    list<node*> newheap, lo;
    node *temp;
    temp = getMin(heap);
    list<node*>:: iterator it;
    it = heap.begin();
    while (it != heap end()) {
        if (*it != temp) {
            newheap.push_back(*it);
        }
        it++;
    }
    lo = removeminAnd returnheap(temp),
    newheap = unionofheap(newheap, lo);
    newheap = adjust(newheap);
    return newheap;
}
list<node*> removeminandreturnheap(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;
}
```