ADS LAB - 9
Binomial Heap

```
Insert : (input: head, key)
{
    Node * temp = new Node (key);
    List <Node *> t;
    t.push_back ( temp);
    t = Union Bit (head)
    return adjust(t);
}

adjust (list <Node *> heap){
    if (heap.size <=1 ) return heap;
    list <Node *> new_heap;
    auto i+1, i+2, i+3;
    i+1 = i+2 = i+3 = heap.begin();
    if (heap.size() == 2) {
        i+2 = i+1
        i+2++;
        i+3 = heap.end();
    } else {
        i+2++;
        i+3 = i+2;
        i+3++;
    } while (i+1 != heap.end()){
    if (i+2 == heap.end()) i+1++;
    else if (i+1->degree < i+2->degree){
        i+1++, i+2++;
        if (i+3 != heap.end()) i+3++;
    }
else if (*i+1->degree == i+2->degree){
    Node * temp;
```

```
        *it1 = merge(*it1, *it2);
        it2 = heap.erase(it2);
        if (it3 != heap.end()) it3++;
    }
    else if (it3 != heap.end() && *it1->degree ==
            *it2->degree && *it->degree ==
            *it3->degree) {
        *it++, it2++, it3++;
    }
    return heap;
}

Function Get min(list<node*> heap){
    auto it = heap.begin();
    while (it != heap.end()) {
        if (*it->data < temp->data) temp = *it;
        it++;
    }
    return temp;
}
Function extract min (list<Node*> heap) {
    List<Node*> new_heap, lo, Node *temp)
        temp = get min(heap), auto it = heap.begin();
        while (it != heap.end()) {
            if (*it != temp) new_heap push back(*it);
            it++;
    }
    lo = rem(temp);
    new_heap = union B-(new_heap, lo);
    new_hrap = adjust(new_heap);
    return new heap;
}
```