Binomial Tree

```
Node mergeBinomialtree (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
}
unionBinomialHeap (list l1, list l2)
{
    list<Node*> new
    list<Node*>::iterator it = l1.begin()
    list<Node*>::iterator 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++
    }
```

```
// if there remains some elements in l1 binomial heap
while (it1 = l1.end())
    new.push_back(it)
    it++
while (it1 = l2.end())
    new.push_back(it)
    it++
return new
}

// Insertion
insertion(heap, tree)
{
    // create new heap
    list<Node*> temp;
    temp.push_back(tree)
    temp = union(heap, temp)
    return adjust(temp)
}
// Removing minimum key ele from binomial heap
removeMin(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

// Insert a key into heap
insert ( head, key)
{ Node *temp = newNode (key)
  return insertion ( head, temp)
}
```