Binomial heap

```
Insert ( head, Key):
{
    Node temp = new Node (Key)
    list <Node *> f
    f. push-back (temp)
    f = union(head, Key)
    return adjust (f)
}

adjust (list <Node *> head)
{
    if ( heap.size <= 1) return heap
    list <Node *> newheap
    auto it1, it2 , it3;
    it1 = it2 = it3 = heap.begin()
    if ( heap.size() == 2)
    {
        it2 = it1
        it2++
        it3 = heap.end()
    }
```

```
        else
        {
            it2++
            it3 = it2
            it3++
        }
    While (it1 != heapend())
{
        if (it2) == heapend())
        it1++
        else if(it1->degree < it2->degree)
        {   it1++.
            it2++
            if (it3 != heapend)
                it3++
        }
        else if (it3 != heapend && (it1->deg
            == it2->deg) && (it1->deg == it3->deg)
{           it1++
            it2++
            it3++
        }
        return heap
}
```

```
getmin( list<Node*> heap)
{
    auto it = heap.begin()
    while(it != heap.end())
    {
        if(*it -> data < temp -> data)
          temp = *it
          it++
    }
    return temp
}

extractMin( list<Node*> heap)
{
    list<Node*> newhead, io;
    Node *temp;
    while(it != heap.end())
    {
        if(*it != temp)
        {
            newhead.push_back(*it)
        }
        it++
    }
Io = removemin(temp)
new_heap = unionBH, new_heap(adjust)
return new_heap }
```