

9) Binomial Heap :

(i) struct Node {

int data, degree;

Node * child, * sibling, * parent;

};

```
list <Node*> insert (list <Node*> _heap, int key)
{
    Node *temp = new Node (key);
    return insert In Heap (-heap, 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*> insert A Tree In Heap (list <Node*> _heap,
Node * tree) {
    list <Node*> temp;
    temp.push_back (tree);
    temp = union Binomial Heap (-heap, temp);
    return adjust (temp);
}
```



```

list <Node* > extract Min ( list <Node* > - heap ) {
    list <Node* > new_heap ,
    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 ++;
    }
    lo = remove Min From Tree . Return B Heap ( temp );
    new_heap = union Binomial Heap ( new_heap , lo );
    new_heap = adjust ( new_heap );
    return new_heap ;
}

```

```

Node * merge Binomial Trees ( 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*> union Binomial Heap (list <Node*> l1,
list <Node*> 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++;

}

}

while (it != l1.end()) {

-new.push_back(*it);

it++;

}

while (ot != l2.end()) {

-new.push_back(*ot);

ot++;

}

return -new;

}

~~list~~


```
list <Node* > removeMin From Tree Return BHeap (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;  
}
```