Ankitha
1BM18CS016

```
Node *root = NULL   - global
struct Node
{   int val, degree;
    Node *parent, *child, *sibling;
};
void binomialHeapInsert(int x)
{
    root = unionHeap(root, createNode(x));
        // creating new node and inserting do union
           of this node with root.
}
void decreaseKeyBHeap(Node *H, int old_val, int new_val)
{
    Node *node = findNoode(H, old_val);
    if (node == NULL)   // element is not present
        return;
    // Reduce the value to minimum
    node->val = new_val;
    Node *parent = node->parent;
    // update the heap according to reduced value
    while (parent != NULL && node->val < parent->val)
    {
        swap(node->val, parent->val);
        node = parent;
        parent = parent->parent;
    }
}
```

```
Node * binomialHeapDelete(Node *h, int val)
{
    if (h == NULL)  // If heap is empty
        return NULL;
    decreaseKeyBHeap(h, val, INT_MIN);
    // Reduce the value of element to minimum

    // Delete the min element from heap;
    return extractMinBHeap(h);
}
Node * extractMinBHeap(Node *h)
{
    if (h == NULL)
        return NULL;
    Node *min_node_prev = NULL;
    Node *min_node = h;
    // finding min value
    int min = h->val;
    Node *curr = h;
    while (curr->sibling != NULL)
    {
        if ((curr->sibling)->val < min)
        {
            min = (curr->sibling)->val;
            min_node_prev = curr;
            min_node = curr->sibling
        }
        curr = curr->sibling;
    }
    // If there is single Node
    if (min_node_prev == NULL && min_node->sibling == NULL)
        h = NULL;
```

```
else if (min_node_prev == NULL)
    h = min_node -> sibling ;

else
    min_node_prev -> sibling = min_node->sibling;
  // Removing min

//set root as children
  if (min_node -> child != NULL)
{
        reverseList(min_node -> child);
        (minnode -> child) -> sibling = NULL;
}
  Do union of root h and children
  return unionBHeaps(h, root);
}
```