Delete and Decrease key functions in Binomial Heap

1. delete(H): Like binary heap, delete operation first reduces the key to minus infinite, then calls extractMin().

2. decreasekey(H): decreasekey() is also similar to binary heap. We compare the decrease key with its parent and if parent's key is more, we swap keys and recur for parent. We stop when we either reach a node whose parent has smaller key or we hit the root node.

Time complexity of decreasekey() is O(logn)

```
struct Node { } // structure
Node* root = NULL;

int binomialLink(Node* h1, Node* h2)   // link two heaps by making h1 a child of h2
{
    h1->parent = h2;
    h1->sibling = h2->child;
    h2->child = h1;
    h2->degree = h2->degree + 1;
}

Node* createNode(int n){}
Node* mergeBHeaps(Node* h1, Node* h2){}
Node* UnionBHeaps(Node* h1, Node* h2){}
```

```
void binomialHeapInsert(){ }
void display (Node* h) { }

int reverseList(Node *h)
{
    if (h->sibling != NULL)
    {
        reverseList(h->sibling);
        (h->sibling) -> sibling = h;
    }
    else
        root = h;
}

Node* extractMinBHeap(Node *h){ }

Node* findNode(Node *h, int val) // Function to search for an element
{
    if (h == NULL)
        return NULL;
    if (h->val == val)
        return h;

    Node* res = findNode(h->child, val);
    if (res != NULL)
        return res;

    return findNode(h->sibling, val);
}
```

```
void decreaseKeyBHeap(Node* H, int old_val, int new_val)
{
    Node* node = findNode(H, old_val); // check element
                                          present or not

    if (node == NULL)
        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)
        return NULL;
    // Reduce the value of element to minimum
    decreaseKeyBHeap(h, val, INT_MIN);
    // Delete minimum element from heap
    return extractMinBHeap(h);
}
```