Chandan C Bagan
1BM18CS026

// Decrease key by new value in Binomial heap.

```
void decreaseKeyBheap (Node *H, int old_val, int new_val)
{
    // 1. Check element is present or not
    // 2. Return if node is not present
    // 3. Reduce value to minimum
    // 4. update the heap according to reduced value

    Node * node = findNode (H, old_val);
    if (Node == NULL)
      return;
    node -> val = new_val;
    Node * parent = node -> parent;
    while (parent != NULL && node -> val < parent -> val)
    { swap (Node -> val, parent -> val);
      Node = parent;
      parent = parent -> parent;
    }
}

// function to search for an element
Node * findNode (Node * h, int val)
{
    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);
}
```

```
// Function to delete an element from Binomial heap.
Node * binDelete (Node *h, int val)
{
    // 1. Check if heap is empty or not
    // 2. Reduce the value of element to minimum
    // 3. Delete the minimum element from heap.
    if (h == NULL)
        return NULL;
    decreaseKeyBheap (h, val, INT_MIN);
    return ExtractMin Heap(h);
}
// Function to extract minimum value from binomial
Node * extractMinBHeap (Node *h)
{
    if (h == NULL)
        return NULL;
    Node * min_node_prev = NULL;
    Node * min_node = h;
    int min = h->val;
    Node * curr = h;
    while (curr->sibling) -> val < min)
    {
        if ((curr->sibling != NULL)
        {
            min = (curr->sibling) -> val;
            min_node_prev = curr;
            min_node = curr->sibling;
        }
        curr = curr->sibling;
    }
    if (min_node_prev == NULL && min_node->sibling == NU
        h = NULL;
```

```
else if (min_node_prev == NULL)
    h = min_node -> sibling;
else
    min_node_prev -> sibling = min_node -> sibling;
if (min_node -> child != NULL)
{
    reverseList (min_node -> child);
    (min_node -> child) -> sibling = NULL;
}
return unionBheaps (h, root);
}
```