```
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 != NULL) {
if ((curr -> sibling) -> val < min) {
min = (curr -> sibling) -> val;
min_node_prev = curr;
min_node = curr -> sibling;
        }
curr = curr -> sibling;
    }
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;
```

```
if (min_node -> child != NULL) {
revertList(min_node -> child);
(min_node -> child) -> sibling = NULL;
    }

return unionBHeaps(h, root);
}

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);
}

void decreaseKeyBHeap(Node *H, int old_val,
int new_val) {
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;
    }
}

Node *binomialHeapDelete(Node *h, int val) {
if (h == NULL) return NULL;
decreaseKeyBHeap(h, val, INT_MIN);
return extractMinBHeap(h);
}
```