main.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Node
{
    int val, degree;
    Node *parent, *child, *sibling;
};

Node *root = NULL;

void binomialLink(Node *h1, Node *h2)
{
    h1->parent = h2;
    h1->sibling = h2->child;
    h2->child = h1;
    h2->degree = h2->degree + 1;
}

Node *createNode(int n)
{
    Node *new_node = new Node;
    new_node->val = n;
    new_node->parent = NULL;
    new_node->sibling = NULL;
    new_node->child = NULL;
    new_node->degree = 0;
    return new_node;
}

```

main.cpp

```
35      if (h2 == NULL)
36          return h1;
37
38      Node *res = NULL;
39
40      if (h1->degree <= h2->degree)
41          res = h1;
42
43      else if (h1->degree > h2->degree)
44          res = h2;
45
46      while (h1 != NULL && h2 != NULL)
47      {
48          if (h1->degree < h2->degree)
49              h1 = h1->sibling;
50
51          else if (h1->degree == h2->degree)
52          {
53              Node *sib = h1->sibling;
54              h1->sibling = h2;
55              h1 = sib;
56          }
57
58          else
59          {
60              Node *sib = h2->sibling;
61              h2->sibling = h1;
62              h2 = sib;
63          }
```

```
72
73      Node *res = mergeBHeaps(h1, h2);
74
75      Node *prev = NULL, *curr = res,
76          *next = curr->sibling;
77      while (next != NULL)
78      {
79          if ((curr->degree != next->degree) ||
80                  ((next->sibling != NULL) &&
81                  (next->sibling)->degree ==
82                  curr->degree))
83          {
84              prev = curr;
85              curr = next;
86          }
87
88          else
89          {
90              if (curr->val <= next->val)
91              {
92                  curr->sibling = next->sibling;
93                  binomialLink(next, curr);
94              }
95              else
96              {
97                  if (prev == NULL)
98                      res = next;
99                  else
100                     prev->sibling = next;
```

main.cpp

```
108 }
109
110 void binomialHeapInsert(int x)
111 {
112     root = unionBHeaps(root, createNode(x));
113 }
114
115 void display(Node *h)
116 {
117     while (h)
118     {
119         cout << h->val << " ";
120         display(h->child);
121         h = h->sibling;
122     }
123 }
124
125 void revertList(Node *h)
126 {
127     if (h->sibling != NULL)
128     {
129         revertList(h->sibling);
130         (h->sibling)->sibling = h;
131     }
132     else
133         root = h;
134 }
135
136 Node *extractMinBHeap(Node *h)
```

main.cpp

```
150                 min = (curr->sibling)->val;
151                 min_node_prev = curr;
152                 min_node = curr->sibling;
153             }
154             curr = curr->sibling;
155         }
156
157         if (min_node_prev == NULL &&
158             min_node->sibling == NULL)
159             h = NULL;
160
161         else if (min_node_prev == NULL)
162             h = min_node->sibling;
163
164         else
165             min_node_prev->sibling = min_node->sibling;
166
167         if (min_node->child != NULL)
168         {
169             revertList(min_node->child);
170             (min_node->child)->sibling = NULL;
171         }
172
173         return unionBHeaps(h, root);
174     }
175
176     Node *findNode(Node *h, int val)
177     {
178         if (h == NULL)
179             return NULL;
```

main.cpp

```
190
191 void decreaseKeyBHeap(Node *H, int old_val, int new_val)
192 {
193     Node *node = findNode(H, old_val);
194
195     if (node == NULL)
196         return;
197
198     node->val = new_val;
199     Node *parent = node->parent;
200
201     while (parent != NULL && node->val < parent->val)
202     {
203         swap(node->val, parent->val);
204         node = parent;
205         parent = parent->parent;
206     }
207 }
208
209 Node *binomialHeapDelete(Node *h, int val)
210 {
211     if (h == NULL)
212         return NULL;
213
214     decreaseKeyBHeap(h, val, INT_MIN);
215
216     return extractMinBHeap(h);
217 }
218
219
```

main.cpp

```
{
    int ele,n;
    cout<<"Enter the no. of elements:"<<endl;
    cin>>n;
    cout<<"Enter the elements to be inserted:"<<endl;
    for(int i=0; i<n; i++)
    {
        cin>>ele;
        binomialHeapInsert(ele);
    }

        cout << "Heap elements after insertion:"<<endl;
        display(root);
    cout<<endl;

        cout << "Enter the number of nodes to be deleted: ";
    cin >> n;
    cout<<"Enter the elements to be deleted: "<<endl;
    for(int i=0; i<n; i++)
    {
        cin>>ele;
        root = binomialHeapDelete(root, ele);
            cout << "After deleting " << ele << ": \n";
            display(root);
            cout << endl;
    }

        return 0;
}

```

```
Enter the no. of elements:
6
Enter the elements to be inserted:
1
99
30
2
15
6
Heap elements after insertion:
6 15 1 2 30 99
Enter the number of nodes to be deleted:
```