

```

1  #include <bits/stdc++.h>
2
3  #include <iostream>
4  using namespace std;
5
6  struct Node {
7      int data, degree;
8      Node *child, *sibling, *parent;
9  };
10
11 Node *newNode(int key) {
12     Node *temp = new Node;
13     temp->data = key;
14     temp->degree = 0;
15     temp->child = temp->parent = temp->sibling = NULL;
16     return temp;
17 }
18
19 Node *mergeBinomialTrees(Node *b1, Node *b2) {
20     if (b1->data > b2->data) swap(b1, b2);
21
22     b2->parent = b1;
23     b2->sibling = b1->child;
24     b1->child = b2;
25     b1->degree++;
26
27     return b1;
28 }
29
30 list<Node *> unionBinomialHeap(list<Node *> l1, list<Node *> l2) {
31     list<Node *> _new;
32     list<Node *>::iterator it = l1.begin();
33     list<Node *>::iterator ot = l2.begin();
34     while (it != l1.end() && ot != l2.end()) {
35         if ((*it)->degree <= (*ot)->degree) {
36             _new.push_back(*it);
37             it++;
38         } else {
39             _new.push_back(*ot);
40             ot++;
41         }
42     }
43
44     while (it != l1.end()) {
45         _new.push_back(*it);
46         it++;
47     }
48
49     while (ot != l2.end()) {
50         _new.push_back(*ot);
51         ot++;
52     }
53     return _new;
54 }
55
56 list<Node *> adjust(list<Node *> _heap) {
57     if (_heap.size() <= 1) return _heap;
58     list<Node *> new_heap;
59     list<Node *>::iterator it1, it2, it3;
60     it1 = it2 = it3 = _heap.begin();
61
62     if (_heap.size() == 2) {
63         it2 = it1;
64         it2++;
65         it3 = _heap.end();
66     } else {
67         it2++;
68         it3 = it2;
69         it3++;
70     }
71     while (it1 != _heap.end()) {
72         if (it2 == _heap.end())
73             it1++;
74
75         else if ((*it1)->degree < (*it2)->degree) {
76             it1++;
77             it2++;
78             if (it3 != _heap.end()) it3++;
79         } else if (it3 != _heap.end() && (*it1)->degree == (*it2)->degree && (*it1)->degree == (*it3)->degree) {
80             it1++;
81             it2++;
82             it3++;
83         }
84
85         else if ((*it1)->degree == (*it2)->degree) {
86             Node *temp;
87             *it1 = mergeBinomialTrees(*it1, *it2);
88             it2 = _heap.erase(it2);
89             if (it3 != _heap.end()) it3++;
90         }
91     }
92     return _heap;
93 }
94
95 list<Node *> insertATreeInHeap(list<Node *> _heap, Node *tree) {
96     list<Node *> temp;
97     temp.push_back(tree);
98     temp = unionBinomialHeap(_heap, temp);
99     return adjust(temp);
100 }

```

```

100 }
101
102 list<Node *> removeMinFromTreeReturnBHeap(Node *tree) {
103     list<Node *> heap;
104     Node *temp = tree->child;
105     Node *lo;
106
107     while (temp) {
108         lo = temp;
109         temp = temp->sibling;
110         lo->sibling = NULL;
111         heap.push_front(lo);
112     }
113     return heap;
114 }
115
116 list<Node *> insert(list<Node *> _heap, int key) {
117     Node *temp = newNode(key);
118     return insertATreeInHeap(_heap, temp);
119 }
120
121 Node *getMin(list<Node *> _heap) {
122     list<Node *>::iterator it = _heap.begin();
123     Node *temp = *it;
124     while (it != _heap.end()) {
125         if ((*it)->data < temp->data) temp = *it;
126         it++;
127     }
128     return temp;
129 }
130
131 list<Node *> extractMin(list<Node *> _heap) {
132     list<Node *> new_heap, lo;
133     Node *temp;
134
135     temp = getMin(_heap);
136     list<Node *>::iterator it;
137     it = _heap.begin();
138     while (it != _heap.end()) {
139         if (*it != temp) new_heap.push_back(*it);
140         it++;
141     }
142     lo = removeMinFromTreeReturnBHeap(temp);
143     new_heap = unionBinomialHeap(new_heap, lo);
144     new_heap = adjust(new_heap);
145     return new_heap;
146 }
147
148 void printTree(Node *h) {
149     while (h) {
150         cout << h->data << " ";
151         printTree(h->child);
152         h = h->sibling;
153     }
154 }
155
156 void printHeap(list<Node *> _heap) {
157     list<Node *>::iterator it;
158     it = _heap.begin();
159     while (it != _heap.end()) {
160         printTree(*it);
161         it++;
162     }
163     cout << endl;
164 }
165
166 int main() {
167     int ch, key;
168     list<Node *> _heap;
169     int n;
170     cout << "Enter n : ";
171     cin >> n;
172     while (n-- > 0) {
173         int k;
174         cin >> k;
175         _heap = insert(_heap, k);
176     }
177
178     cout << "Heap elements after insertion:\n";
179     printHeap(_heap);
180
181     Node *temp = getMin(_heap);
182     cout << "\nMinimum element of heap " << temp->data << "\n";
183
184     _heap = extractMin(_heap);
185     cout << "Heap after deletion of minimum element\n";
186     printHeap(_heap);
187     return 0;
188 }

```



Enter n : 6

11

24

55

66

22

77

Heap elements after insertion:

22 77 11 55 66 24

Minimum element of heap 11

Heap after deletion of minimum element

24 22 55 66 77



Terminal



Enter n : 10

124

44

46

346

24

765

46

88

235

66

Heap elements after insertion:

66 235 24 44 46 346 124 46 88 765

Minimum element of heap 24

Heap after deletion of minimum element

765 44 46 66 235 88 46 346 124