main.cpp

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

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

Node* newNode(int key)
{
        Node *temp = new Node;
        temp->data = key;
        temp->degree = 0;
        temp->child = temp->parent = temp->sibling = NULL;
        return temp;
}

Node* mergeBinomialTrees(Node *b1, Node *b2)
{
    if (b1->data > b2->data)
            swap(b1, b2);
        b2->parent = b1;
        b2->sibling = b1->child;
        b1->child = b2;
        b1->degree++;

        return b1;
}
```

main.cpp

```cpp
    list<Node*> _new;
        list<Node*>::iterator it = l1.begin();
        list<Node*>::iterator ot = l2.begin();
        while (it!=l1.end() && ot!=l2.end())
        {
            if((*it)->degree <= (*ot)->degree)
            {
                    _new.push_back(*it);
                    it++;
            }
            else
            {
                    _new.push_back(*ot);
                    ot++;
            }
        }

        while (it != l1.end())
        {
            _new.push_back(*it);
            it++;
        }

        while (ot!=l2.end())
        {
            _new.push_back(*ot);
            ot++;
        }
        return _new;
```

main.cpp

```
66      if (_heap.size() <= 1)
67          return _heap;
68      list<Node*> new_heap;
69      list<Node*>::iterator it1,it2,it3;
70      it1 = it2 = it3 = _heap.begin();
71
72      if (_heap.size() == 2)
73      {
74          it2 = it1;
75          it2++;
76          it3 = _heap.end();
77      }
78      else
79      {
80          it2++;
81          it3=it2;
82          it3++;
83      }
84      while (it1 != _heap.end())
85      {
86          if (it2 == _heap.end())
87              it1++;
88
89          else if ((*it1)->degree < (*it2)->degree)
90          {
91                  it1++;
92                  it2++;
93                  if(it3!=_heap.end())
94                      it3++;
95          }
```

main.cpp

```
96
97              else if (it3!=_heap.end() && (*it1)->degree == (*it2)->degree && (*it1)->degree ==
98              {
99                      it1++;
100                     it2++;
101                     it3++;
102             }
103
104             else if ((*it1)->degree == (*it2)->degree)
105             {
106                     Node *temp;
107                     *it1 = mergeBinomialTrees(*it1,*it2);
108                     it2 = _heap.erase(it2);
109                     if(it3 != _heap.end())
110                         it3++;
111             }
112         }
113         return _heap;
114 }
115
116 list<Node*> insertATreeInHeap(list<Node*> _heap, Node *tree)
117 {
118     list<Node*> temp;
119         temp.push_back(tree);
120         temp = unionBionomialHeap(_heap,temp);
121         return adjust(temp);
122 }
123
124 list<Node*> removeMinFromTreeReturnBHeap(Node *tree)
125 {
```

main.cpp

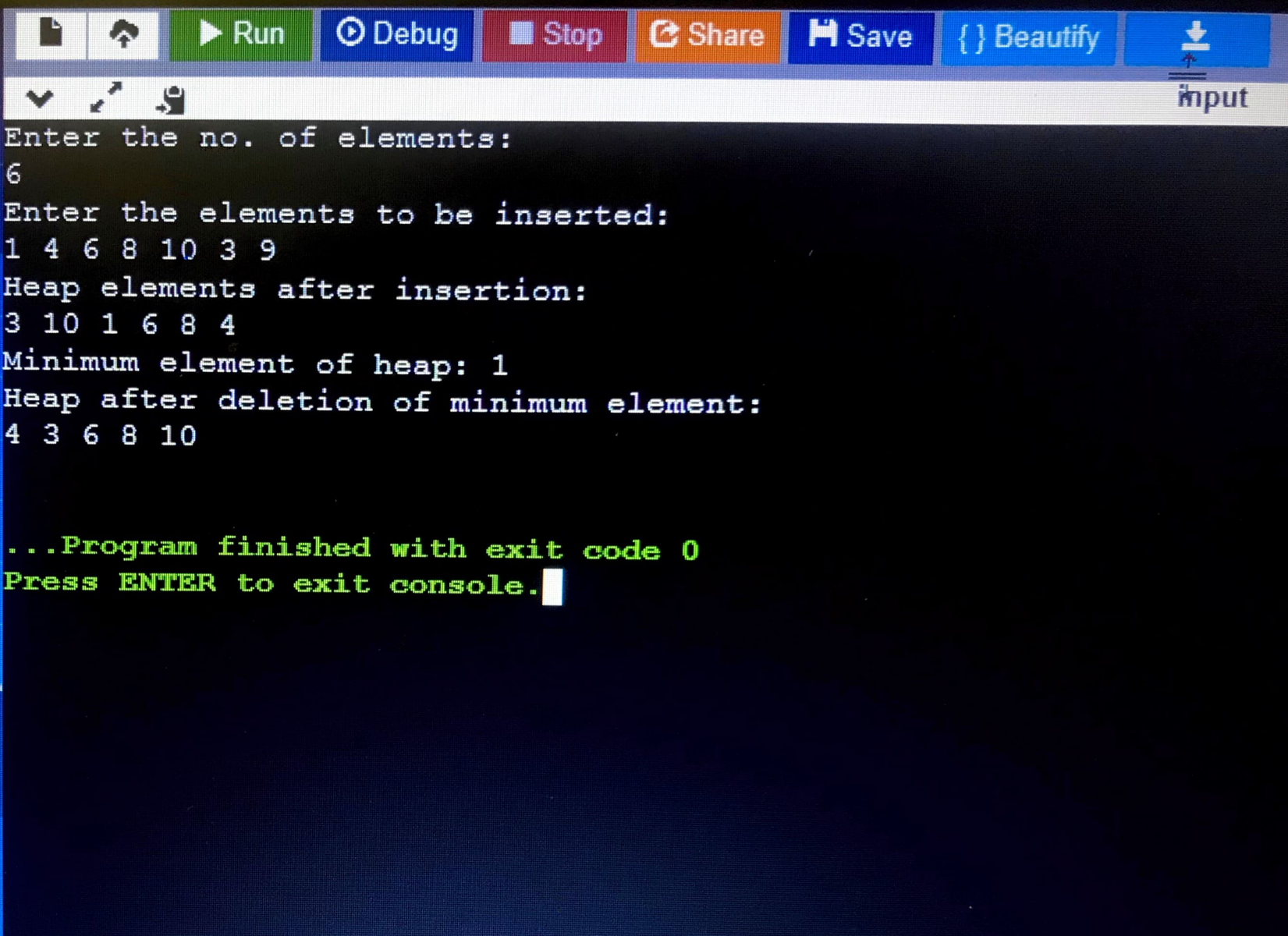
```
129
130         while (temp)
131         {
132             lo = temp;
133             temp = temp->sibling;
134             lo->sibling = NULL;
135             heap.push_front(lo);
136         }
137         return heap;
138 }
139
140 list<Node*> insert(list<Node*> _head, int key)
141 {
142         Node *temp = newNode(key);
143         return insertATreeInHeap(_head,temp);
144 }
145
146 Node* getMin(list<Node*> _heap)
147 {
148         list<Node*>::iterator it = _heap.begin();
149         Node *temp = *it;
150         while (it != _heap.end())
151         {
152             if ((*it)->data < temp->data)
153                 temp = *it;
154             it++;
155         }
156         return temp;
157 }
158
```

main.cpp

```
164             temp = getMin(_heap);
165             list<Node*>::iterator it;
166             it = _heap.begin();
167             while (it != _heap.end())
168             {
169                 if (*it != temp)
170                 {
171                         new_heap.push_back(*it);
172                 }
173                 it++;
174             }
175             lo = removeMinFromTreeReturnBHeap(temp);
176             new_heap = unionBionomialHeap(new_heap,lo);
177             new_heap = adjust(new_heap);
178             return new_heap;
179     }
180
181     void printTree(Node *h)
182     {
183             while (h)
184             {
185                 cout << h->data << " ";
186                 printTree(h->child);
187                 h = h->sibling;
188             }
189     }
190
191     void printHeap(list<Node*> _heap)
192     {
193             list<Node*> ::iterator it;
```

main.cpp

```
200     cout<<endl;
201 }
202
203 int main()
204 {
205     int ele,n;
206         list<Node*> _heap;
207     cout<<"Enter the no. of elements:"<<endl;
208     cin>>n;
209     cout<<"Enter the elements to be inserted:"<<endl;
210     for(int i=0; i<n; i++)
211     {
212         cin>>ele;
213         _heap = insert(_heap,ele);
214     }
215
216         cout << "Heap elements after insertion:"<<endl;
217         printHeap(_heap);
218
219         Node *temp = getMin(_heap);
220         cout << "Minimum element of heap: "<< temp->data <<endl;
221
222         _heap = extractMin(_heap);
223         cout << "Heap after deletion of minimum element: "<<endl;
224         printHeap(_heap);
225
226         return 0;
227 }
228
229
```


Run
Debug
Stop
Share
Save
{ } Beautify
Input
Enter the no. of elements:
6
Enter the elements to be inserted:
1 4 6 8 10 3 9
Heap elements after insertion:
3 10 1 6 8 4
Minimum element of heap: 1
Heap after deletion of minimum element:
4 3 6 8 10
...Program finished with exit code 0
Press ENTER to exit console.