HEAP

SUBMITTED BY:

JASKIRAT SINGH

2020CSC1008

SUBMITTED TO:

MRS. AYUSHI GUPTA

Code

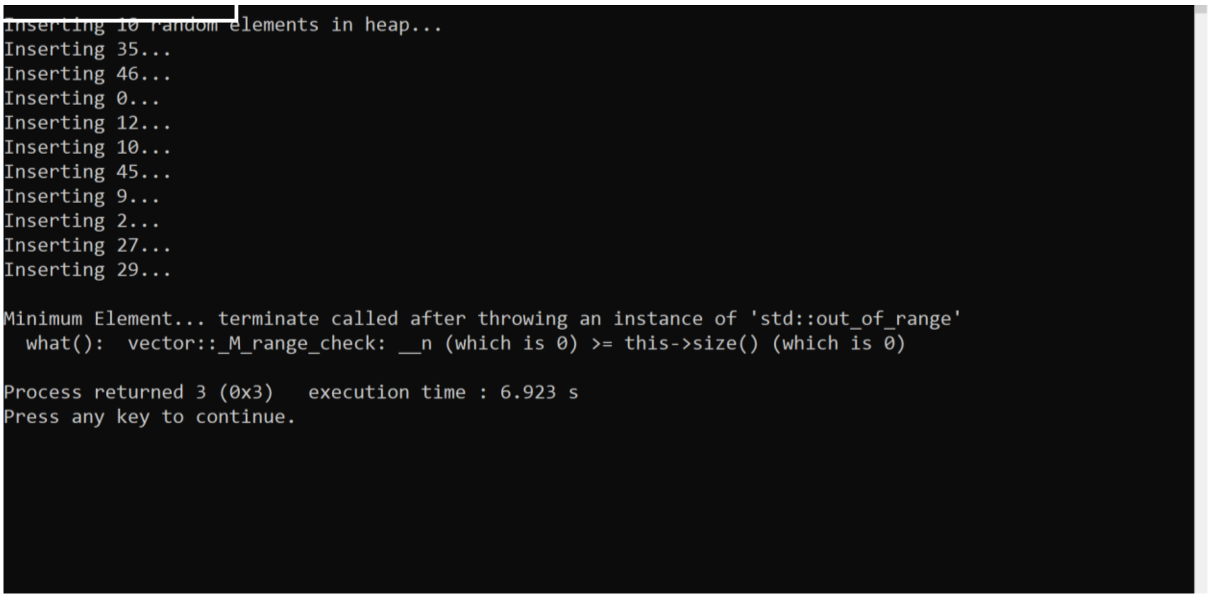
```
#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <vector>
```

```
using namespace std;


class CompleteTree
{ // left-complete tree interface
private:
    vector<int> V; // tree contents
public:
    int last() { return V.size() - 1; }
    int left(int i) { return 2 * i + 1; }
    int right(int i) { return 2 * i + 2; }
    int parent(int i) { return (i - 1) / 2; }
    bool hasLeft(int i) { return left(i) <= last(); }
    bool hasRight(int i) { return right(i) <= last(); }
    bool isRoot(int i) { return i == 0; }
    int root() { return 0; }
    void addLast(int e) { V.push_back(e); }
    void removeLast() { V.pop_back(); }
    void swap(int &x, int &y)
    {
        int temp = move(x);
        x = move(y);
        y = move(temp);

    }

    int elemAtIdx(int i) { return V.at(i); }

};


class HeapPriorityQueue

{

private:

    CompleteTree T;


public:

    void insert(int e)

    {

        T.addLast(e);     // add e to heap

        int v = T.last(); // e’s position

        while (!T.isRoot(v))

        { // up-heap bubbling

            int u = T.parent(v);

            if (T.elemAtIdx(u) < T.elemAtIdx(v))

                break;    // if v in order

            T.swap(v, u); // . . .else swap with parent
            v = u;
        }
    }
    int min()
    {
        int x = T.elemAtIdx(0);
        return x;
    }

    void removeMin()
    {
        if (T.last() == 0)
        {
            T.removeLast();
        } // . . .remove it
        else
        {
            int u = T.root();
            int v = T.last();
            T.swap(u, v);   // swap last with root
            T.removeLast(); // . . .and remove last
```

```
}

      while (T.hasLeft(u))
      { // down-heap bubbling
         int smaller = T.left(u);
         if (T.hasRight(u))
         {
            int x = T.right(u);
            if (T.elemAtIdx(x) < T.elemAtIdx(smaller))
               smaller = x;
         }
         if (T.elemAtIdx(smaller) < T.elemAtIdx(u))
         {
            T.swap(u, smaller); // . . .then swap
            u = smaller;
         }
         else
            break;
      }
   }
 }
};
```

```
int main()

{

    HeapPriorityQueue hpq;

    cout << "\nInserting 10 random elements in heap... ";

    srand(time(0));

    for (int i = 0; i < 10; i++)

    {

        int random = rand() % 50;

        cout << "\nInserting " << random << "...";

    }


    cout<<"\n\nMinimum Element... "<<hpq.min();


    cout << "\n\nRemoving minimum element...";

    hpq.removeMin();


    cout<<"\n\nMinimum Element after Deletion ... "<<hpq.min();


    return 0;

}
```

Output



```
Inserting 10 random elements in heap...
Inserting 35...
Inserting 46...
Inserting 0...
Inserting 12...
Inserting 10...
Inserting 45...
Inserting 9...
Inserting 2...
Inserting 27...
Inserting 29...

Minimum Element... terminate called after throwing an instance of 'std::out_of_range'
  what():  vector::_M_range_check: __n (which is 0) >= this->size() (which is 0)

Process returned 3 (0x3)   execution time : 6.923 s
Press any key to continue.
```