Q1. Write a c++ program, to demonstrate priority queue.

```
#include <iostream>
#include <queue>
using namespace std;

int main() {
    priority_queue <int> s;


    s.push(1);
    s.push(2);
    s.push(3);
    s.push(4);
    s.push(5);
    s.push(6);
    s.push(7);


    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }

    return 0;
}
```

Q2. Implement different operations on priority queue .i.e. adding element, removing element, size of priority queue, and print it.

```
#include <iostream>
#include <queue>
using namespace std;

int main()
{
   priority_queue <int> q;

   q.push(10);
   q.push(20);
   q.push(30);

   q.pop();

   cout<<q.size()<<endl;

   while(!q.empty())
   {
      cout<<q.top()<<" ";
      q.pop();
   }
}
```

Q3. Write a c++ program, to demonstrate priority queue having a min element at top.

```
#include <iostream>
#include <queue>

using namespace std;

int main()
{
   priority_queue < int, vector<int>, greater<int> > q;

   q.push(30);
   q.push(10);
   q.push(20);
   q.push(50);

   while(!q.empty())
   {
      cout<<q.top()<<" ";
      q.pop();
   }
}
```

Q4. Write a c++ program, to swap the elements of two priority queues of int type.

```
#include <iostream>
#include <queue>
using namespace std;

void swapping(priority_queue <int> &q1, priority_queue <int> &q2);

int main()
{
   priority_queue <int> q1;
   priority_queue <int> q2;

   for(int i = 1; i < 5; i++)
   {
      q1.push(i * 10);
   }

   for(int i = 1; i < 5; i++)
   {
      q2.push(i * 20);
   }

   swapping(q1, q2);

   cout<<"q1 = ";
   while(!q1.empty())
   {
      cout<<q1.top()<<" ";
      q1.pop();
   }

   cout<<endl<<"q2 = ";
   while(!q2.empty())
   {
      cout<<q2.top()<<" ";
      q2.pop();
   }
}

void swapping(priority_queue <int> &q1, priority_queue <int> &q2)
{
   priority_queue <int> tmp;

   while(!q1.empty())
   {
      tmp.push(q1.top());
      q1.pop();
   }

   while(!q2.empty())
   {
      q1.push(q2.top());
      q2.pop();
   }
```

```
    while(!tmp.empty())
    {
        q2.push(tmp.top());
        tmp.pop();
    }
}
```

Q5. Write a c++ program, to show that priority_queue is by default a Max Heap.
Note:
If elements are printed in descending order, then we have a max heap.

```
#include <iostream>
#include <queue>
using namespace std;

int main()
{
   // Creating a priority_queue (by default max heap)
   priority_queue<int> pq;

   // Inserting elements
   pq.push(30);
   pq.push(10);
   pq.push(50);
   pq.push(20);
   pq.push(40);

   // Printing elements
   cout << "Elements in priority_queue (Max Heap): ";
   while(!pq.empty())
   {
      cout << pq.top() << " ";  // Always gives the largest element
      pq.pop();
   }

   return 0;
}
```

Q6. Write a c++ program, to use priority_queue to implement min heap.

```
#include <iostream>
#include <queue>
//#include <vector>   // vector is needed for min heap
using namespace std;

int main()
{
   // Min Heap using priority_queue
   priority_queue<int, vector<int>, greater<int>> minHeap;

   // Inserting elements
   minHeap.push(30);
   minHeap.push(10);
   minHeap.push(50);
   minHeap.push(20);
   minHeap.push(40);

   // Printing elements (ascending order)
   cout << "Elements in Min Heap (ascending order): ";
   while(!minHeap.empty())
   {
      cout << minHeap.top() << " ";
      minHeap.pop();
   }

   return 0;
}
```

Q7. Given two sorted arrays A[] and B[] of sizes N and M respectively, the task is to merge them in a sorted manner using priority_queue.
Example:
Input: A[] = { 5, 6, 8 }, B[] = { 4, 7, 8 }
Output: 4 5 6 7 8 8

```
#include <iostream>
#include <queue>
using namespace std;

int main()
{
   priority_queue < int, vector<int>, greater<int> > q;

   int N, M;
   N = M = 0;

   cout<<"Enter size of 1st array and 2nd array = ";
   cin>>N>>M;

   int A[N], B[M];

   cout<<"Enter element in 1st array"<<endl;
   for(int i = 0; i < N; i++)
   {
      cin>>A[i];
   }

   cout<<"Enter element in 2nd array"<<endl;
   for(int i = 0; i < M; i++)
   {
      cin>>B[i];
   }

   for(int i = 0; i < N; i++)
   {
      q.push( A[i] );
   }

   for(int i = 0; i < M; i++)
   {
      q.push( B[i] );
   }

   cout<<endl<<endl;
   while(!q.empty())
   {
      cout<<q.top();
      q.pop();
   }

}
```

/*Q8. Given an array arr[] of N elements, the task is to perform using priority_queue and the following operation:
●Pick the two largest element from the array and remove these element. If the elements are unequal then insert the absolute difference of the elements into the array.
●Perform the above operations until the array has 1 or no element in it. If the array has only one element left then print that element, else print “-1”.
Example:
Input: arr[] = { 3, 5, 2, 7 }
Output: 1
Explanation:
The two largest elements are 7 and 5. Discard them. Since both are not equal, insert 7 – 5 = 2 into the array. Hence, arr[] = { 3, 2, 2 }
The two largest elements are 3 and 2. Discard them. Since both are not equal, insert 3 – 2 = 1 into the array. Hence, arr[] = { 1, 2 }
The two largest elements are 2 and 1. Discard them. Since both are not equal, insert 2 – 1 = 1 into the array. Hence, arr[] = { 1 }
The only element left is 1. Print the value of the only element left.*/

```
#include <iostream>
#include <queue>
using namespace std;

int* getArr(int &N, priority_queue <int> &q);
void getQ(int* &arr, int &N, priority_queue <int> &q);
int diff(int &a, int &b);

int main()
{
   int N = 0;

   cout<<"Enter array size = ";
   cin>>N;

   int* arr  = new int[N];

   for(int i = 0; i < N; i++)
   {
      cout<<i<<") = ";
      cin>>arr[i];
   }

   priority_queue <int> q;

   while(N > 1 )
   {
      getQ(arr, N, q); // copy arr -> q
      delete[] arr;

      int a = q.top(); q.pop();
      int b = q.top(); q.pop();
      int c = diff(a, b);

      if(c != 0)
      {
         q.push(c);
```

```
        }
        N = q.size();
        arr = getArr(N, q);
    }

    if(arr != nullptr)
    {
        cout<<arr[0];
        delete[] arr;
    }
    else
    {
        cout<<-1;
    }
}

int* getArr(int &N, priority_queue <int> &q)
{
    if(N == 0)
    {
        return nullptr;
    }
    else
    {
        int* arr = new int[N];

        for(int i = 0; i < N; i++)
        {
        arr[i] = q.top(); q.pop(); //insert element in  new array and queue is emtpy
        }

        return arr;
    }

}

void getQ(int* &arr, int &N, priority_queue <int> &q)
{

    for(int i = 0; i < N; i++)
    {
        q.push(arr[i]);
    }
}

int diff(int &a, int &b)
{
    return (a == b) ? 0 : a - b;
}
```

Q9. Given three arrays X[], Y[], and Z[] each consisting of N integers, the task is to find the maximum number of triplets (X[i], Y[i], Z[i]) such that (X[i] < Y[i] < Z[i]) for any permutation of the three arrays using priority_queue
Input: X = {9, 6, 14, 1, 8}, Y = {2, 10, 3, 12, 11}, Z = {15, 13, 5, 7, 4}
Output: 3
Explanation:
After rearranging the arrays X[], Y[] and Z[] as {1, 6, 8, 9, 14}, {3, 2, 10, 12, 11}, and {4, 7, 15, 13, 5} respectively. The increasing triplets are {1, 3, 4}, {8, 10, 15} and {9, 12, 13}.
Therefore, the total count of such triplets is 3.

```
#include <iostream>
#include <queue>

using namespace std;
using assending = priority_queue <int, vector<int>, greater<int>>;

int* setArr(int N)
{
   int* arr = new int[N];
   for(int i = 0; i < N; i++)
   {
      cout<<i<<") ";
      cin>>arr[i];
   }

   return arr;
}

void sortArr(int* &arr, int &N, int flag = 0, int element = 0)
{
   assending tmp;

   if(flag == 1)
   {
      for(int i = 0; i < N; i++) //element push in q from array
      {
         if(arr[i] == element)
         {
            continue;
         }
         else
         {
            tmp.push(arr[i]);
         }
      }
      N = tmp.size();
   }
   else
   {
      for(int i = 0; i < N; i++) //element push in q from array
      {
         tmp.push(arr[i]);
      }
   }
```

```
    delete[] arr; //delete actual array

    arr = new int[tmp.size()]; //element push in actual array from q
    for(int i = 0; !tmp.empty(); i++)
    {
        arr[i] = tmp.top();
        tmp.pop();
    }
}

bool traverse(int a, int b)
{
    if(a < b)
    {
        return true;
    }
    else
    {
        return false;
    }
}

int main()
{
    assending Xq;
    assending Yq;
    assending Zq;

    int* X = nullptr;
    int* Y = nullptr;
    int* Z = nullptr;

    int N = 0, Xn, Yn, Zn;

    cout<<"Enter size of array = ";
    cin>>N;

    Xn = Yn = Zn = N;

    cout<<"Enter data for X"<<endl;
    X = setArr(N);

    cout<<"Enter data for Y"<<endl;
    Y = setArr(N);

    cout<<"Enter data for Z"<<endl;
    Z = setArr(N);

    sortArr(X, N);
    sortArr(Y, N);
    sortArr(Z, N);

    Xn = Yn = Zn = N;

    int i = 0;
```

```
    while(i < Xn)
    {
        int flag = 0;

        for(int j = 0; (j < Yn) && (flag == 0); j++)
        {
            if( traverse(X[i], Y[j]) )
            {
                for(int k = 0; (k < Zn) && (flag == 0); k++)
                {
                    if( traverse(Y[j], Z[k]) )
                    {
                        flag = 1;

                        Xq.push(X[i]);
                        Yq.push(Y[j]);
                        Zq.push(Z[k]);

                        sortArr(X, Xn, flag, X[i]);
                        sortArr(Y, Yn, flag, Y[j]);
                        sortArr(Z, Zn, flag, Z[k]);
                    }
                    else
                    {
                        continue;
                    }
                }

            }
            else
            {
                continue;
            }
        }
        if(flag == 0)
        {
            i = i + 1;
        }
        else
        {
            i = 0;
        }
    }

    while(!Xq.empty())
    {
        cout<<"{ "<<Xq.top()<<", "<<Yq.top()<<", "<<Zq.top()<<" }"<<endl;
        Xq.pop();
        Yq.pop();
        Zq.pop();
    }

    delete[] X;
    delete[] Y;
    delete[] Z;
}
```

Q10. Given an array arr[] of size N and a number K, the task is to find the length of the smallest subsequence such that the sum of the subsequence is greater than or equal to number K do it using priority_queue.
Example:
Input: arr[] = {2, 3, 1, 5, 6, 3, 7, 9, 14, 10, 2, 5}, K = 35
Output: 4
Smallest subsequence with the sum greater than or equal to the given sum K is {7, 9, 14, 10}
Input: arr[] = {1, 2, 2, 2, 3, 4, 5, 4, 7, 6, 5, 12}, K = 70
Output:-1
Subsequence with sum greater than equal to the given sum is not possible.

```
#include <iostream>
#include <queue>
using namespace std;

int main() {

    int N = 0;
    int K = 35;
    int sum = 0, cont = 0;

    cout<<"Enter size of array = ";
    cin>>N;

    cout<<"Enter K value = ";
    cin>>K;

    int arr[N]={2, 3, 1, 5, 6, 3, 7, 9, 14, 10, 2, 5};

    priority_queue <int> q;

    for(int i = 0; i < N; i++)
    {
        cout<<i<<") ";
        cin>>arr[i];
        q.push(arr[i]);
    }

    while(!q.empty())
    {
        sum = sum + q.top(); q.pop();
        cont = cont + 1;

        if(sum >= K)
        {
            cout<<cont;
            break;
        }
    }

    if(sum < K)
    {
        cout<<-1;
    }
}
```