Q1. Create a c++ program using multiset and returns an iterator to the first element in the multiset –> O(1)

```
#include <iostream>
#include <set>
#include <iterator>
using namespace std;

int main() {
   multiset <int> s = {10, 20, 30, 10, 20, 30};

   multiset <int>::iterator it = s.begin();

   cout<<*it;

}
```

Q2. Create a c++ program using multiset and returns an iterator to the theoretical element that follows the last element in the multiset –> O(1)

```
#include <iostream>
#include <set>
#include <iterator>
using namespace std;

int main()
{
   multiset <int> s = {10, 20, 30, 40};

   multiset  <int>::iterator it = s.end();

}
```

Q3. Create a c++ program using multiset and returns the number of elements in the multiset –> O(1)

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

int main()
{
   multiset <int> s = {10, 20, 30, 70, 10, 20};

   cout<<s.size();
}
```

Q4. Create a c++ program using multiset and returns the maximum number of elements that the multiset can hold –> O(1)

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

int main()
{
   multiset <int> s;

   cout<<s.max_size();
}
```

Q5. Create a c++ program using multiset and returns whether the multiset is empty –> O(1)

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

int main()
{
   multiset <int> s;

   if(s.empty())
      cout<<"multiset empty";
   else
      cout<<"multiset not empty";
}
```

Q6. Create a c++ program using multiset and inserts the element x in the multiset –> O(log n)

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

int main()
{
   multiset <int> s;

   int x;

   cout<<"Enter element = ";
   cin>>x;

   s.insert(x);


}
```

Q7. Create a c++ program using multiset and removes all the elements from the multiset –> O(n)

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

int main()
{
    multiset <int> s = {10, 20, 30, 40, 50};

    s.clear();
}
```

Q8. Create a c++ program using multiset and removes all the occurrences of x –> O(log n)

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

int main()
{
   multiset <int> s = {10, 20, 30, 10, 20, 30, 40, 50};

   s.erase(10);
   s.erase(20);
   s.erase(30);

   for(int x : s) cout<<x<<" ";
}
```

Q9. Create a c++ program using multiset and remove only one instance of element from multiset having same value

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

int main() {
   multiset <int> s = {10, 20, 30, 20, 40, 10};

   multiset <int>::iterator j = s.begin();

   for(auto i = s.begin(); i != s.end(); i++ )
   {
      j = i;
      j++;

      if((*i) == *(j))
      {
         s.erase(j);
      }
      else
      {
         break;
      }
   }

   for(int x : s) cout<<x<<" ";
}
```

Q10. Unlike a set, a multiset may contain multiple occurrences of same number. The multiset equivalence problem states to check if two given multisets are equal or not. For example let A = {1, 2, 3} and B = {1, 1, 2, 3}. Here A is set but B is not (1 occurs twice in B), whereas A and B are both multisets. More formally, “Are the sets of pairs defined as \(A' = \{ (a, frequency(a)) | a \in \mathbf{A} \}\) equal for the two given multisets?” Given two multisets A and B, write a program to check if the two multisets are equal.

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

int main() {
   multiset <int> A = {1, 2, 3};
   multiset <int> B = {1, 1, 2, 3};

   if(A == B)
   {
      cout<<"multiset are equal";
   }
   else
   {
      cout<<"multiset are not equal";
   }
}
```