Find missing elements of a range

Given an array arr[0..n-1] of distinct elements and a range [low, high], find all numbers that are in range, but not in array. The missing elements should be printed in sorted order.

Examples:

```
Input: arr[] = {10, 12, 11, 15},
       low = 10, hight = 15
Output: 13, 14

Input: arr[] = {1, 14, 11, 51, 15},
       low = 50, hight = 55
Output: 50, 52, 53, 54
```

We strongly recommend you to minimize your browser and try this yourself first.

There can be following two approaches to solve the problem.

1. **Use Sorting :** Sort the array, then do binary search for 'low'. Once location of low is find, start traversing array from that location and keep printing all missing numbers.

```
// A sorting based C++ program to find missing
// elements from an array
#include<bits/stdc++.h>
using namespace std;

// Print all elements of range [low, high] that
// are not present in arr[0..n-1]
void printMissing(int arr[], int n, int low,
                                  int high)
{
   // Sort the array
   sort(arr, arr+n);

   // Do binary search for 'low' in sorted
   // array and find index of first element
   // which either equal to or greater than
   // low.
   int *ptr = lower_bound(arr, arr+n, low);
   int index = ptr - arr;

   // Start from the found index and linearly
   // search every range element x after this
   // index in arr[]
   int i = index, x = low;
   while (i < n && x<=high)
   {
       // If x doesn't math with current element
       // print it
       if (arr[i] != x)
          cout << x << " ";

       // If x matches, move to next element in arr[]
       else
          i++;

       // Move to next element in range [low, high]
       x++;
   }

   // Print range elements thar are greater than the
   // last element of sorted array.
   while (x <= high)
     cout << x++ << " ";
}

// Driver program
int main()
{
   int arr[] = {1, 3, 5, 4};
   int n = sizeof(arr)/sizeof(arr[0]);
   int low = 1, high = 10;
   printMissing(arr, n, low, high);
   return 0;
}
```

Output:

```
2 6 7 8 9 10
```

2. **Use Hashing :** Create a hash table and insert all array items into the hash table. Once all items are in hash table, traverse through the range and print all missing elements.

```
// A hashing based C++ program to find missing
// elements from an array
#include<bits/stdc++.h>
using namespace std;

// Print all elements of range [low, high] that
// are not present in arr[0..n-1]
void printMissing(int arr[], int n, int low,
                                  int high)
{
   // Insert all elements of arr[] in set
   unordered_set<int> s;
   for (int i=0; i<n; i++)
      s.insert(arr[i]);

   // Traverse throught the range an print all
   // missing elements
   for (int x=low; x<=high; x++)
      if (s.find(x) == s.end())
         cout << x << " ";
}

// Driver program
int main()
{
   int arr[] = {1, 3, 5, 2};
   int n = sizeof(arr)/sizeof(arr[0]);
   int low = 1, high = 10;
   printMissing(arr, n, low, high);
   return 0;
}
```

Output:

```
2 6 7 8 9 10
```

Which approach is better?

Time complexity of first approach is O(nLogn + k) where k is number of missing elements (Note that k may be more than nLogn if array is small and range is big)

Time complexity of second solution is O(n + (high-low+1)).

If the given array has almost elements of range i.e., n is close to value of (high-low+1), then second approach is definitely better as there is no Log n factor. But if n is much smaller than range, then first approach is better as it doesn't require extra space for hashing. We can also modify first approach to print adjacent missing elements as range to save time. For example if 50, 51, 52, 53, 54, 59 are missing, we can print them as 50-54, 59 in first method. And if printing this way is allowed, the first approach takes only O(n Log n) time.