K Closest Elements

Given a sorted array arr[] and a value X, find the k closest elements to X in arr[].

Examples:

```
Input: K = 4, X = 35
       arr[] = {12, 16, 22, 30, 35, 39, 42,
                45, 48, 50, 53, 55, 56}
Output: 30 39 42 45
```

Note that if the element is present in array, then it should not be in output, only the other closest elements are required

In the following solutions, it is assumed that all elements of array are distinct.
A **simple solution** is to do linear search for k closest elements.
1) Start from the first element and search for the crossover point (The point before which elements are smaller than or equal to X and after which elements are greater). This step takes O(n) time.
2) Once we find the crossover point, we can compare elements on both sides of crossover point to print k closest elements. This step takes O(k) time.
The time complexity of the above solution is O(n).

An **Optimized Solution** is to find k elements in O(Logn + k) time. The idea is to use Binary Search to find the crossover point. Once we find index of crossover point, we can print k closest elements in O(k) time.

C++ Java

```java
// Java program to find k closest elements to a given value
class KClosest
{
    /* Function to find the cross over point (the point before
       which elements are smaller than or equal to x and after
       which greater than x)*/
    int findCrossOver(int arr[], int low, int high, int x)
    {
        // Base cases
        if (arr[high] <= x) // x is greater than all
            return high;
        if (arr[low] > x)  // x is smaller than all
            return low;

        // Find the middle point
        int mid = (low + high)/2;  /* low + (high - low)/2 */

        /* If x is same as middle element, then return mid */
        if (arr[mid] <= x && arr[mid+1] > x)
            return mid;

        /* If x is greater than arr[mid], then either arr[mid + 1]
          is ceiling of x or ceiling lies in arr[mid+1...high] */
        if(arr[mid] < x)
            return findCrossOver(arr, mid+1, high, x);

        return findCrossOver(arr, low, mid - 1, x);
    }

    // This function prints k closest elements to x in arr[].
    // n is the number of elements in arr[]
    void printKclosest(int arr[], int x, int k, int n)
    {
        // Find the crossover point
        int l = findCrossOver(arr, 0, n-1, x);
        int r = l+1;   // Right index to search
        int count = 0; // To keep track of count of elements
                       // already printed
```

```
        // If x is present in arr[], then reduce left index
        // Assumption: all elements in arr[] are distinct
        if (arr[l] == x) l--;

        // Compare elements on left and right of crossover
        // point to find the k closest elements
        while (l >= 0 && r < n && count < k)
        {
            if (x - arr[l] < arr[r] - x)
                System.out.print(arr[l--]+" ");
            else
                System.out.print(arr[r++]+" ");
            count++;
        }

        // If there are no more elements on right side, then
        // print left elements
        while (count < k && l >= 0)
        {
            System.out.print(arr[l--]+" ");
            count++;
        }


        // If there are no more elements on left side, then
        // print right elements
        while (count < k && r < n)
        {
            System.out.print(arr[r++]+" ");
            count++;
        }
    }

    /* Driver program to check above functions */
    public static void main(String args[])
    {
        KClosest ob = new KClosest();
        int arr[] = {12, 16, 22, 30, 35, 39, 42,
                     45, 48, 50, 53, 55, 56
                    };
        int n = arr.length;
```

```
            int x = 35, k = 4;
            ob.printKclosest(arr, x, 4, n);
        }
    }
    /* This code is contributed by Rajat Mishra */
```


Output:

```
39 30 42 45
```

The time complexity of this method is O(Logn + k).