**1. Problem Discussion:**

We have a sorted array and two integers k and x, we have to return the k closest integers to x in the array. The result should also be sorted in ascending order. An integer a is closer to x than an integer b if: |a - x| < |b - x|, or |a - x| == |b - x| and a < b Example: Arr = { 10, 20, 30, 40, 50, 60 } K = 3 X = 45

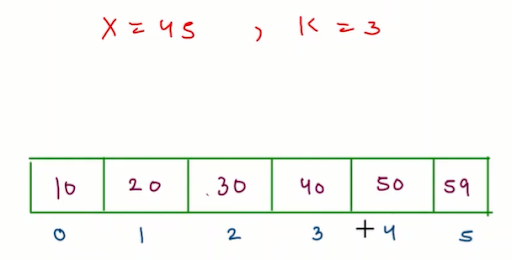
Approach:

Trivial Approach:

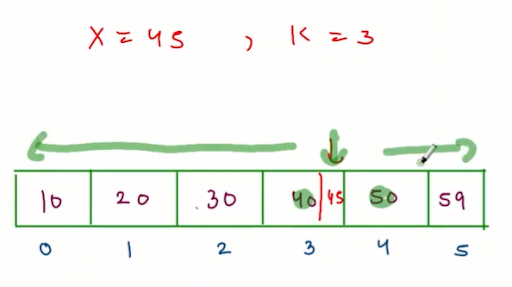
The trivial approach to solve this question will be to use a Pair class having two things one is value i.e. arr[i] and the other is gap i.e. abs(arr[i] - x). Then make a priority queue of max heap type and add the pairs one by one in pq till the size of pq is less than k. When the size is equal to k then check for the peek().gap of the element if it is greater than the element's gap then we will remove the peek pair and add the new pair of that element with gap. At the end we have the k closest element to x in an sorted array. But the elements are in reverse order in the priority queue so we need to sort the array at last. This approach will use O(nlog(k)) time complexity.

Optimised Approach with O(log(n) + klog(k)):

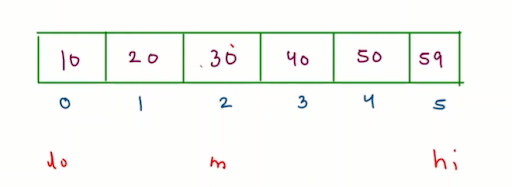
We have to select k closest elements to x in a sorted array. This means we have a sorted array, we can use this property to solve this question in less than O(nlog(k)). In sorted array to find any element we can use binary search which will take log n complexity to find an element in a sorted array.So, the steps to do for solving this question is:

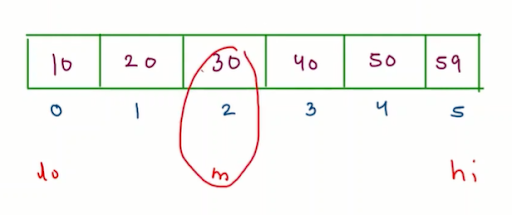
****

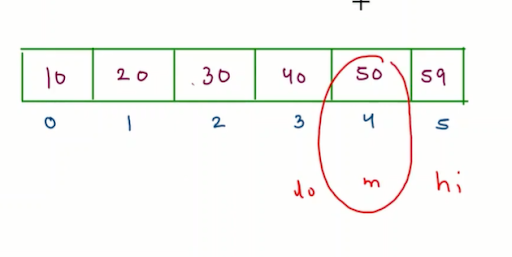
1• To find the element x in the array we will use binary search so that we can get the index of elements x. If x is not present in the array still binary search gives the closest index in the array to x. We will use that index to find k closest elements.

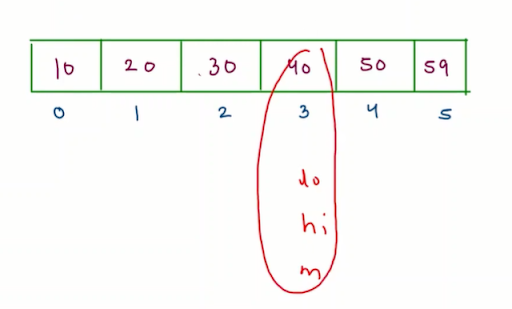
****

2• After finding the closest element index we will segregate the array in two parts : one is left array and second is right array. Left array will have the closest elements to the x from mid - 1 to 0th index. Right array will have the closest elements to the x from mid to arr.length-1.

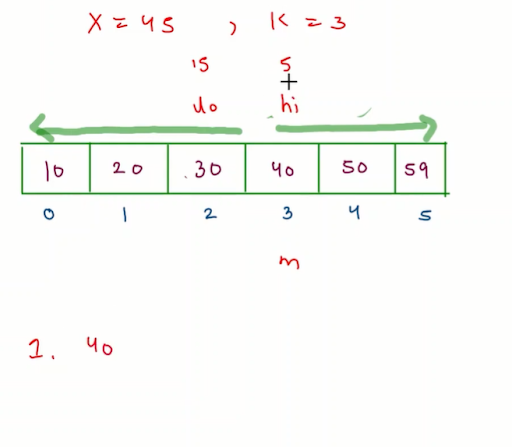
****

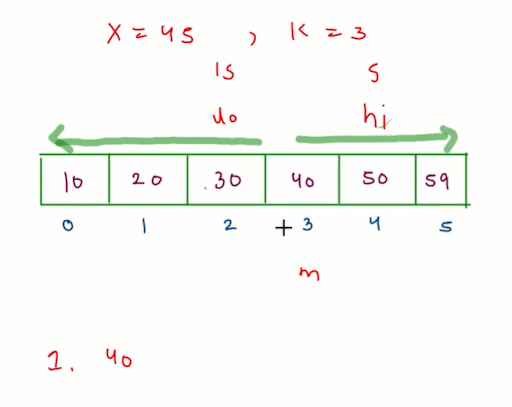
****

****

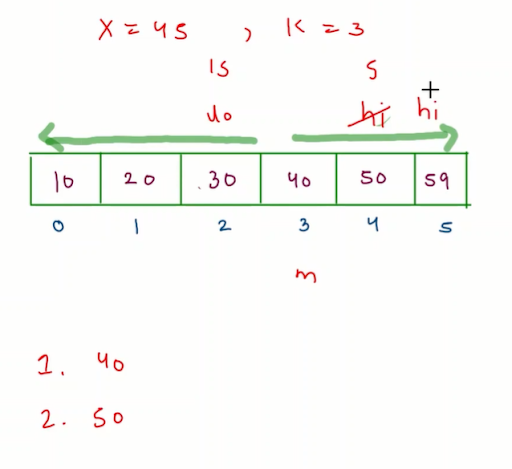
****

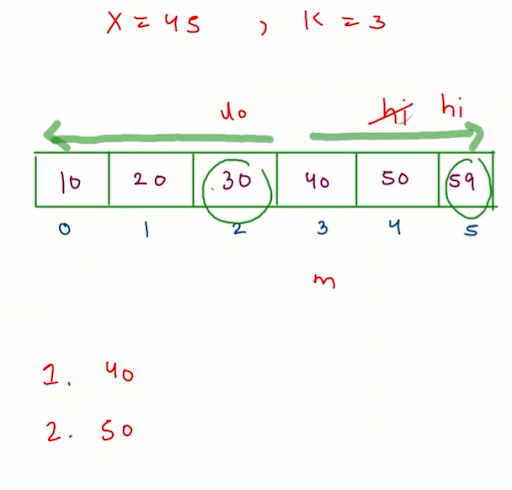
3• Now we will compare the gaps of the left index with x and the right index with x.This means we are going to use a 2 pointer approach in this to find k closest elements. The element of the array with less gap will change the pointer value of li or ri i.e. left index and right index value and we will add the less gap element in the arrayList.

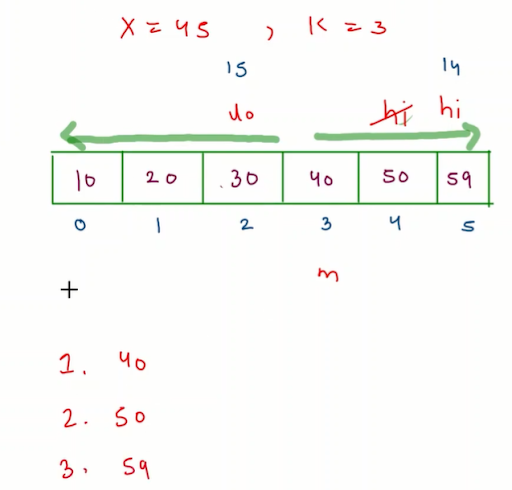
****

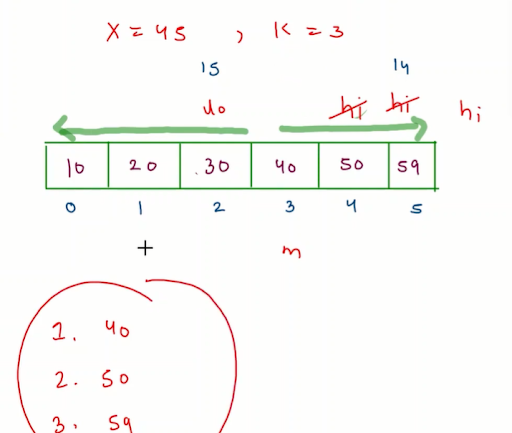
****

4• If the left index element having less gap than right index element with x than we will add left index element in the arrayList and change the li = li -1 else the right index element will be add to the arrayList and change the ri = ri + 1.

****

****

****

****

5• For finding k elements there will be a condition while(li >=0 && ri < arr.length && k >0) we will add the closest element. If the k is still greater than 0 this means some elements are still left so for that we need to check if the left array part is left or right array part for that we need to use two while loops one for left array part and other for right array part. For the left array part condition will be while(li>=0 && k>0) add the elements in the arraylist. And for the right array part condition will be while(ri<arr.length && k>0) add the elements in the arraylist.

6• Now, we have the k closest element but the elements stored in the arraylist are not in ascending order so we need to sort the array for the output format.

Code:

ConsoleJava

import java.util.\*;

import java.io.\*;

public class Main {

/\*find 'k' closest element to 'x' and return answer list\*/

/\*elements in answer list should be in ascending order\*/

public static ArrayList<Integer> findClosest(int[]arr,int k,int x) {

//write your code here

ArrayList<Integer>list = new ArrayList<>();

//first find closest element

int lo = 0;

int hi = arr.length-1;

int mid = 0;

while(lo <= hi) {

mid = (lo + hi)/2;

if(arr[mid] == x) {

break;

}

else if(arr[mid] < x) {

lo = mid+1;

}

else {

hi = mid-1;

}

}

lo = mid-1;

hi = mid;

while(lo >= 0 && hi < arr.length && k-- > 0) {

if(Math.abs(arr[lo]-x) < Math.abs(arr[hi]-x)) {

list.add(arr[lo]);

lo--;

}

else if(Math.abs(arr[lo]-x) > Math.abs(arr[hi]-x)) {

list.add(arr[hi]);

hi++;

}

else {

list.add(arr[lo]);

lo--;

}

}

while(lo >= 0 && k-- > 0) {

list.add(arr[lo]);

lo--;

}

while(hi < arr.length && k-- > 0) {

list.add(arr[hi]);

hi++;

}

Collections.sort(list);

return list;

}

public static void main(String[]args) {

//input work

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int[]arr = new int[n];

for(int i=0; i < n;i++) {

arr[i] = scn.nextInt();

}

int k = scn.nextInt();

int x = scn.nextInt();

ArrayList<Integer>ans = findClosest(arr,k,x);

for(int val : ans) {

System.out.print(val + " ");

}

}

}

Analysis:

Time complexity: This solution will work in O(log(n) + klog(k)) time complexity, where we have done a binary search O(log(n)) and select k elements than we have sort the k elements O(klog(k)). Space complexity: We have used constant space so overall space complexity is O(1).