Find K Closest Elements

Given a sorted array, two integers `k` and `x`, find the `k` closest elements to `x` in the array. The result should also be sorted in ascending order. If there is a tie, the smaller elements are always preferred.

Example 1:

```
Input: [1,2,3,4,5], k=4, x=3
Output: [1,2,3,4]
```

Example 2:

```
Input: [1,2,3,4,5], k=4, x=-1
Output: [1,2,3,4]
```

Note:

1. The value k is positive and will always be smaller than the length of the sorted array.
2. Length of the given array is positive and will not exceed 10^4
3. Absolute value of elements in the array and x will not exceed 10^4

Solution 1

O(nlog(n)) Time Solution:

```
public List<Integer> findClosestElements(List<Integer> arr, int k, int x) {
     Collections.sort(arr, (a,b) -> a == b ? a - b : Math.abs(a-x) - Math.abs(b-x))
;
     arr = arr.subList(0, k);
     Collections.sort(arr);
     return arr;
}
```

O(n) Time Solution:

```
public List<Integer> findClosestElements(List<Integer> arr, int k, int x) {
    List<Integer> less = new ArrayList<>(), greater = new ArrayList<>(),
                  lessResult = new LinkedList<>(), greaterResult = new LinkedList<>
();

    for (Integer i : arr) {
        if (i <= x) less.add(i);
        else greater.add(i);
    }

    Collections.reverse(less);
    int  i = 0, j = 0, n = less.size(), m = greater.size();
    for (int size=0;size<k;size++) {
        if (i < n && j < m) {
            if (Math.abs(less.get(i) - x) <= Math.abs(greater.get(j) - x)) lessResul
t.add(less.get(i++));
            else greaterResult.add(greater.get(j++));
        }
        else if (i < n) lessResult.add(less.get(i++));
        else greaterResult.add(greater.get(j++));
    }

    Collections.reverse(lessResult);
    lessResult.addAll(greaterResult);
    return lessResult;
}
```

Note that above solution can be improved using binary search under the assumption that we have O(1) access to elements in input list.

written by compton_scatter original link here

Solution 2

I binary-search for where the resulting elements start in the array. It's the first index `i` so that `arr[i]` is better than `arr[i+k]` (with "better" meaning closer to or equally close to `x`). Then I just return the `k` elements starting there.

```ruby
def find_closest_elements(arr, k, x)
  arr[(0..arr.size).bsearch { |i| x - arr[i] <= (arr[i+k] || 1/0.0) - x }, k]
end
```

I think that's simpler than binary-searching for `x` and then expanding to the left and to the right like I've seen in other binary search solutions.

Edit: I also implemented it in Go now, to make it O(log n).

Here's a Java version without using the library's binary search:

```java
public List<Integer> findClosestElements(List<Integer> arr, int k, int x) {
    int lo = 0, hi = arr.size() - k;
    while (lo < hi) {
        int mid = (lo + hi) / 2;
        if (x - arr.get(mid) > arr.get(mid+k) - x)
            lo = mid + 1;
        else
            hi = mid;
    }
    return arr.subList(lo, lo + k);
}
```

written by StefanPochmann original link here

Solution 3

The idea is to find the first number which is equal to or greater than `x` in `arr`. Then, we determine the indices of the start and the end of a subarray in `arr`, where the subarray is our result. The time complexity is `O(logn + k)`.

In the following code, `arr[index]` is the first number which is euqal to or geater than `x` (if all numbers are less than `x`, `index` is `arr.size()`), and the result is `arr[i+1, i+2, ... j]`.

Java version:

```
    public List<Integer> findClosestElements(List<Integer> arr, int k, int x) {
        int index = Collections.binarySearch(arr, x);
        if(index < 0) index = -(index + 1);
        int i = index - 1, j = index;
        while(k-- > 0){
            if(i<0 || (j<arr.size() && Math.abs(arr.get(i) - x) > Math.abs(arr.get(j
) - x) ))j++;
            else i--;
        }
        return arr.subList(i+1, j);
    }
```

C++ version:

```
    vector<int> findClosestElements(vector<int>& arr, int k, int x) {
        int index = std::lower_bound(arr.begin(), arr.end(), x) - arr.begin();
        int i = index - 1, j = index;
        while(k--) (i<0 || (j<arr.size() && abs(arr[i] - x) > abs(arr[j] - x) ))? j+
+: i--;
        return vector<int>(arr.begin() + i + 1, arr.begin() + j );
    }
```

written by Hao Cai original link here