

题目描述

放暑假了，小明决定到某旅游景点游玩，他在网上搜索到了各种价位的酒店（长度为n的数组A），他的心理价位是x元，请帮他筛选出k个最接近x元的酒店（ $n \geq k > 0$ ），并由低到高打印酒店的价格。

输入描述

第一行：n, k, x
第二行：A[0] A[1] A[2]...A[n-1]

输出描述

从低到高打印筛选出的酒店价格

用例

输入	10 5 6 1 2 3 4 5 6 7 8 9 10
输出	4 5 6 7 8
说明	无

输入	10 4 6 10 9 8 7 6 5 4 3 2 1
输出	4 5 6 7
说明	无

输入	6 3 1000 30 30 200 500 70 300
输出	200 300 500
说明	无

题目解析

我的解题思路如下：

首先，我们需要将给定的酒店价格列表price升序。

然后，找到最接近心理价位x的值price[i]，将i作为中心位置。然后，分别遍历中心位置左边位值left=i-1，右边位值right=i+1，比较price[left]和price[right]谁更接近x（即与x的差的绝对值更小），如果price[left]接近，则left--，如果price[right]接近，则right++，如果price[left]和price[right]一样近，则优先选取price[left]

(PS：根据用例2得出的结论，因为用例输出是4 5 6 7，而不是5 6 7 8，其中4和8距离6相同，但是用例2选择了4)

如果left--或者right++之后，发生了越界，则对应的price值为Infinity，表示和中心值距离无穷大，这样就可以只会产生一边延申的效果了。

注意本题不会发生两边同时越界的情况，因为 $n \geq k > 0$ 。

以上就是解题的大致逻辑。

但是上面逻辑还有一个关键点没有实现，那就是一开始如何找到最接近心理价位x的price[i]值。

- 方案1：顺序遍历price数组，找price[i] === x 或者 price[i] < x < price[i+1],
 - 对于price[i] === x情况，则i位置就是中心位置
 - 对于price[i] < x < price[i+1]，则取price[i]， price[i+1]更接近x值得那个位置，如果一样近，则取靠左得 i。

方案1得时间复杂度是O(n)

- 方案2：由于我们已经将price升序了，那么可以借助二分查找，这里得二分查找我们可以参考Java得 Arrays.binarySearch来实现

题目解析

我的解题思路如下：

首先，我们需要将给定的酒店价格列表price升序。

然后，找到最接近心理价位x的值price[i]，将i作为中心位置。然后，分别遍历中心位置左边位left=i-1，右边位置right=i+1，比较price[left]和price[right]谁更接近x（即与x的差的绝对值更小），如果price[left]接近，则left--，如果price[right]接近，则right++，如果price[left]和price[right]一样近，则优先选取price[left]

（PS：根据用例2得出的结论，因为用例输出是4 5 6 7，而不是5 6 7 8，其中4和8距离6相同，但是用例2选择了4）

如果left--或者right++之后，发生了越界，则对应的price值为Infinity[∞]，表示和中心值距离无穷大，这样就可以只会产生一边延申的效果了。

注意本题不会发生两边同时越界的情况，因为n>=k>0。

以上就是解题的大致逻辑。

但是上面逻辑还有一个关键点没有实现，那就是一开始如何找到最接近心理价位x的price[i]值。

- 方案1：顺序遍历price数组，找price[i] === x 或者 price[i] < x < price[i+1]。

- 对于price[i] === x情况，则i位置就是中心位置
- 对于price[i] < x < price[i+1]，则取price[i]， price[i+1]更接近x值得那个位置，如果一样近，则取靠左得i。

方案1得时间复杂度是O(n)

- 方案2：由于我们已经将price升序了，那么可以借助二分查找，这里得二分查找我们可以参考Java得Arrays.binarySearch来实现

关于Java得Arrays.binarySearch，如果要找得值在数组中，则直接返回值在数组中得索引，如果要找的值不在数组中，则返回 -idx-1，其中idx是要查找的值在数组中的有序插入位置。
比如有序数组 [1,3,5,7,9]，现在我们要在该数组中二分查找4的位置，可以发现该数组中根本没有4，但是Java得Arrays.binarySearch方法就会返回 -2 -1，即-3，其中2就是查找值4理应在数组中的插入位置，即如果将4插入到有序数组中，则其位置应该是2，如 [1,3,4,5,7,9]

方案2的时间复杂度是O(logN)

由于本题没有给出数量级，因此可以选择简单的方案1先试试。下面算法源码使用了方案2。

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12 });
13 if (lines.length === 2) {
14   const [n, k, x] = lines[0].split(" ").map(Number);
15   const prices = lines[1].split(" ").map(Number);
16
17   console.log(getResult(n, k, x, prices));
18
19   lines.length = 0;
20 }
21 });
22
23 function getResult(n, k, x, prices) {
24   // 数组升序
25   prices.sort((a, b) => a - b);
26
27   // 二分查找数组中最接近心理价位x的元素的位置idx
28   let idx = binarySearch(prices, x);
29
30   // 如果idx<0，说明在数组中没有找到对应值，因此此时idx是有序插入位置
31   if (idx < 0) {
32     idx = -idx - 1; // 将idx转换为有序插入位置
33
34     // 如果idx的插入位置越界，则只会是右边界越界
35     if (idx === prices.length) return prices.slice(idx - k, idx).join(" ");
36
37     let diff1 = Math.abs(x - prices[idx]);
38     let diff2 = Math.abs(x - prices[idx - 1]);
39     if (diff1 > diff2) {
40       // 选择最接近的值得位置作为中心位置
41       idx -= 1;
42     }
43   }
44
45   // 中心位置的左右k个最近元素，即返回结果集合
```

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 2) {
14     const [n, k, x] = lines[0].split(" ").map(Number);
15     const prices = lines[1].split(" ").map(Number);
16
17     console.log(getResult(n, k, x, prices));
18
19     lines.length = 0;
20   }
21 });
22
23 function getResult(n, k, x, prices) {
24   // 数组升序
25   prices.sort((a, b) => a - b);
26
27   // 二分查找数组中最接近心理价位x的元素的位置idx
28   let idx = binarySearch(prices, x);
29
30   // 如果idx<0, 说明在数组中没有找到对应值, 因此此时idx是有序插入位置
31   if (idx < 0) {
32     idx = -idx - 1; // 将idx转换为有序插入位置
33
34     // 如果idx的插入位置越界, 则只会是右边界越界
35     if (idx === prices.length) return prices.slice(idx - k, idx).join(" ");
36
37     let diff1 = Math.abs(x - prices[idx]);
38     let diff2 = Math.abs(x - prices[idx - 1]);
39     if (diff1 > diff2) {
40       // 选择更接近的值的位位置作为中心位置
41       idx -= 1;
42     }
43   }
44
45   // 中心位置的值算1个接近心里价位的值, 因此已经找到1个, 所以k--
46   k--;
47
48   let left = idx;
49   let right = idx;
50   while (k) {
51     // 从中心位置向两边发散
52     let diff1 = Math.abs(x - (prices[left - 1] ?? Infinity));
53     let diff2 = Math.abs(x - (prices[right + 1] ?? Infinity));
54
55     // 那边值更接近, 则范围向哪边扩大
56     if (diff1 <= diff2) {
57       left--;
58     } else {
59       right++;
60     }
61
62     // 每次都能找到一个更接近的心里价位
63     k--;
64   }
65
66   // left, right范围就是题解
67   return prices.slice(left, right + 1).join(" ");
68 }
69
70 function binarySearch(prices, key) {
71   let low = 0;
72   let high = prices.length - 1;
73
74   while (low <= high) {
75     let mid = (low + high) >> 1;
76     let midVal = prices[mid];
77
78     if (midVal < key) {
79       low = mid + 1;
80     } else if (midVal > key) {
81       high = mid - 1;
82     } else {
83       return mid;
84     }
85   }
86   return -low - 1;
87 }
```

Java算法源码

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3 import java.util.StringJoiner;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8
9         int n = sc.nextInt();
10        int k = sc.nextInt();
11        int x = sc.nextInt();
12
13        int[] prices = new int[n];
14        for (int i = 0; i < n; i++) {
15            prices[i] = sc.nextInt();
16        }
17
18        System.out.println(getResult(n, k, x, prices));
19    }
20
21    public static String getResult(int n, int k, int x, int[] prices) {
22        // 数组排序
23        Arrays.sort(prices);
24
25        // 二分查找数组中最接近心理价位x的元素的位置idx
26        int idx = Arrays.binarySearch(prices, x);
27
28        // 如果idx<0, 说明在数组中没有找到对应值, 因此此时idx是有序插入位置
29        if (idx < 0) {
30            idx = -idx - 1; // 将idx转换为有序插入位置
31
32            // 如果idx的插入位置越界, 则只会是右边界越界
33            if (idx == prices.length) {
34                int[] ans = Arrays.copyOfRange(prices, idx - k, idx);
35                StringJoiner sj = new StringJoiner(" ");
36                for (int an : ans) {
37                    sj.add(an + "");
38                }
39                return sj.toString();
40            }
41
42            int diff1 = Math.abs(x - prices[idx]);
43            int diff2 = Math.abs(x - prices[idx - 1]);
44            if (diff1 > diff2) {
45                // 选择更接近的值的位位置作为中心位置
46                idx -= 1;
47            }
48        }
49
50        // 中心位置的值得算1个接近心里价位的值, 因此已经找到1个, 所以k--
51        k--;
52
53        int left = idx;
54        int right = idx;
55        while (k > 0) {
56            int leftVal = left == 0 ? Integer.MAX_VALUE : prices[left - 1];
57            int rightVal = right == n - 1 ? Integer.MAX_VALUE : prices[right + 1];
58
59            // 从中心位置向两边发散
60            int diff1 = Math.abs(x - leftVal);
61            int diff2 = Math.abs(x - rightVal);
62
63            // 那边值更接近, 则范围向那边扩大
64            if (diff1 <= diff2) {
65                left--;
66            } else {
67                right++;
68            }
69
70            // 每次都能找到一个更接近的心里价位
71            k--;
72        }
73
74        // left, right范围就是题解
75        int[] ans = Arrays.copyOfRange(prices, left, right + 1);
76        StringJoiner sj = new StringJoiner(" ");
77        for (int an : ans) {
78            sj.add(an + "");
79        }
80        return sj.toString();
81    }
82 }
```

Python算法源码

```
1 import sys
2
3 # 输入获取
4 n, k, x = map(int, input().split())
5 prices = list(map(int, input().split()))
```

Python算法源码

```
1 import sys
2
3 # 输入获取
4 n, k, x = map(int, input().split())
5 prices = list(map(int, input().split()))
6
7
8 # 算法入口
9 def getResult(n, k, x, prices):
10     # 数组升序
11     prices.sort()
12
13     # 二分查找数组中最接近心理价位x的元素的位置idx
14     idx = binarySearch(prices, x)
15
16     # 如果idx<0, 说明在数组中没有找到对应值, 因此此时idx是有序插入位置
17     if idx < 0:
18         idx = -idx - 1 # 将idx转换为有序插入位置
19
20     # 如果idx的插入位置越界, 则只会是右边界越界
21     if idx == len(prices):
22         return " ".join(map(str, prices[(idx - k):idx]))
23
24     diff1 = abs(x - prices[idx])
25     diff2 = abs(x - prices[idx - 1])
26
27     if diff1 > diff2:
28         # 选择最接近的值的中心位置
29         idx -= 1
30
31     # 中心位置的值得1个接近心里价位的值, 因此已经找到1个, 所以k--
32     k -= 1
33
34     left = idx
35     right = idx
36     while k > 0:
37         # 从中心位置向两边发散
38         leftVal = sys.maxsize if left == 0 else prices[left - 1]
39         rightVal = sys.maxsize if right == n - 1 else prices[right + 1]
40
41         diff1 = abs(x - leftVal)
42         diff2 = abs(x - rightVal)
43
44         # 那边值更接近, 则范围向那边扩大
45         if diff1 <= diff2:
46             left -= 1
47         else:
48             right += 1
49
50     # 每次都能找到一个最接近的心里价位
51     k -= 1
52
53     # left, right范围就是题解
54     return " ".join(map(str, prices[left:(right + 1)]))
55
56
57 # 二分查找实现
58 def binarySearch(arr, key):
59     low = 0
60     high = len(arr) - 1
61
62     while low <= high:
63         mid = (low + high) >> 1
64         midVal = arr[mid]
65
66         if midVal < key:
67             low = mid + 1
68         elif midVal > key:
69             high = mid - 1
70         else:
71             return mid
72
73     return - low - 1
74
75
76 # 调用算法
77 print(getResult(n, k, x, prices))
```