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>=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值置为 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],

1. 对于price[i] === x情况，则i位置就是中心位置
2. 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

引，如果要找的值不

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值置为 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],

1. 对于price[i] === x情况，则i位置就是中心位置
2. 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

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const [n, k, x] = lines[0].split(" ").map(Number);
    const prices = lines[1].split(" ").map(Number);

    console.log(getResult(n, k, x, prices));

    lines.length = 0;
  }
});

function getResult(n, k, x, prices) {
  // 数组升序
  prices.sort((a, b) => a - b);

  // 二分查找数组中最接近心理价位x的元素的位置idx
  let idx = binarySearch(prices, x);

  // 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
  if (idx < 0) {
    idx = -idx - 1; // 将idx转换为有序插入位置

    // 如果idx的插入位置越界，则只会是右边界越界
    if (idx === prices.length) return prices.slice(idx - k, idx).join(" ");

    let diff1 = Math.abs(x - prices[idx]);
    let diff2 = Math.abs(x - prices[idx - 1]);
    if (diff1 > diff2) {
      // 选择最接近的值的位置作为中心位置
      idx -= 1;
    }
  }

  // 中心位置的值算1个接近心理价位的值，因此已经找到1个，所以
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const [n, k, x] = lines[0].split(" ").map(Number);
    const prices = lines[1].split(" ").map(Number);

    console.log(getResult(n, k, x, prices));

    lines.length = 0;
  }
});

function getResult(n, k, x, prices) {
  // 数组升序
  prices.sort((a, b) => a - b);

  // 二分查找数组中最接近心理价位x的元素的位置idx
  let idx = binarySearch(prices, x);

  // 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
  if (idx < 0) {
    idx = -idx - 1; // 将idx转换为有序插入位置

    // 如果idx的插入位置越界，则只会是右边界越界
    if (idx === prices.length) return prices.slice(idx - k, idx).join(" ");

    let diff1 = Math.abs(x - prices[idx]);
    let diff2 = Math.abs(x - prices[idx - 1]);
    if (diff1 > diff2) {
      // 选择最接近的值的位置作为中心位置
      idx -= 1;
    }
  }

  // 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--
  k--;

  let left = idx;
  let right = idx;
  while (k) {
    // 从中心位置向两边发散
    let diff1 = Math.abs(x - (prices[left - 1] ?? Infinity));
    let diff2 = Math.abs(x - (prices[right + 1] ?? Infinity));

    // 那边值更接近，则范围向哪边扩大
    if (diff1 <= diff2) {
      left--;
    } else {
      right++;
    }

    // 每次都能找到一个最接近的心里价位
    k--;
  }

  // left,right范围就是题解
  return prices.slice(left, right + 1).join(" ");
}

function binarySearch(prices, key) {
  let low = 0;
  let high = prices.length - 1;

  while (low <= high) {
    let mid = (low + high) >>> 1;
    let midVal = prices[mid];

    if (midVal < key) {
      low = mid + 1;
    } else if (midVal > key) {
      high = mid - 1;
    } else {
      return mid;
    }
  }
  return -low - 1;
}
```

Java算法源码

```java
import java.util.Arrays;
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;
import java.util.StringJoiner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();
    int k = sc.nextInt();
    int x = sc.nextInt();

    int[] prices = new int[n];
    for (int i = 0; i < n; i++) {
      prices[i] = sc.nextInt();
    }

    System.out.println(getResult(n, k, x, prices));
  }

  public static String getResult(int n, int k, int x, int[] prices) {
    // 数组升序
    Arrays.sort(prices);

    // 二分查找数组中最接近心理价位x的元素的位置idx
    int idx = Arrays.binarySearch(prices, x);

    // 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
    if (idx < 0) {
      idx = -idx - 1; // 将idx转换为有序插入位置

      // 如果idx的插入位置越界，则只会是右边界越界
      if (idx == prices.length) {
        int[] ans = Arrays.copyOfRange(prices, idx - k, idx);
        StringJoiner sj = new StringJoiner(" ");
        for (int an : ans) {
          sj.add(an + "");
        }
        return sj.toString();
      }

      int diff1 = Math.abs(x - prices[idx]);
      int diff2 = Math.abs(x - prices[idx - 1]);
      if (diff1 > diff2) {
        // 选择最接近的值的位置作为中心位置
        idx -= 1;
      }
    }

    // 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--
    k--;

    int left = idx;
    int right = idx;
    while (k > 0) {
      int leftVal = left == 0 ? Integer.MAX_VALUE : prices[left - 1];
      int rightVal = right == n - 1 ? Integer.MAX_VALUE : prices[right + 1];

      // 从中心位置向两边发散
      int diff1 = Math.abs(x - leftVal);
      int diff2 = Math.abs(x - rightVal);

      // 那边值更接近，则范围向哪边扩大
      if (diff1 <= diff2) {
        left--;
      } else {
        right++;
      }

      // 每次都能找到一个最接近的心里价位
      k--;
    }

    // left,right范围就是题解
    int[] ans = Arrays.copyOfRange(prices, left, right + 1);
    StringJoiner sj = new StringJoiner(" ");
    for (int an : ans) {
      sj.add(an + "");
    }
    return sj.toString();
  }
}
```

Python算法源码

```python
import sys

# 输入获取
n, k, x = map(int, input().split())
prices = list(map(int, input().split()))
```

Python算法源码

```python
import sys

# 输入获取
n, k, x = map(int, input().split())
prices = list(map(int, input().split()))


# 算法入口
def getResult(n, k, x, prices):
    # 数组升序
    prices.sort()

    # 二分查找数组中最接近心理价位x的元素的位置idx
    idx = binarySearch(prices, x)

    # 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
    if idx < 0:
        idx = -idx - 1 # 将idx转换为有序插入位置

        # 如果idx的插入位置越界，则只会是右边界越界
        if idx == len(prices):
            return " ".join(map(str, prices[(idx - k):idx]))

        diff1 = abs(x - prices[idx])
        diff2 = abs(x - prices[idx - 1])

        if diff1 > diff2:
            # 选择最接近的值的位置作为中心位置
            idx -= 1

    # 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--
    k -= 1

    left = idx
    right = idx
    while k > 0:
        # 从中心位置向两边发散
        leftVal = sys.maxsize if left == 0 else prices[left - 1]
        rightVal = sys.maxsize if right == n - 1 else prices[right + 1]

        diff1 = abs(x - leftVal)
        diff2 = abs(x - rightVal)

        # 那边值更接近，则范围向哪边扩大
        if diff1 <= diff2:
            left -= 1
        else:
            right += 1

        # 每次都能找到一个最接近的心里价位
        k -= 1

    # left,right范围就是题解
    return " ".join(map(str, prices[left:(right + 1)]))


# 二分查找实现
def binarySearch(arr, key):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) >> 1
        midVal = arr[mid]

        if midVal < key:
            low = mid + 1
        elif midVal > key:
            high = mid - 1
        else:
            return mid

    return - low - 1


# 调用算法
print(getResult(n, k, x, prices))
```