Technique: Binary Search over Integer Space

Level: Easy

Find the square root of an integer X. For example, squareRoot(4) = 2.
If X is not a perfect square, find the integer floor of the square root.

For example,
squareRoot(5) & squareRoot(8) will return 2.
squareRoot(9) will return 3.

Questions to Clarify:
Q. Can the input be a negative number?
A. No, only positive numbers and zero.

Solution:
Keep in mind that while it may look like a mathematical problem, the interviewer is actually asking you to search for the square root.

The brute force approach is to iterate i from 0 to X, until i^2 equals or exceeds X.
This will take O(X) time. We can improve this using binary search.

We can search over the integer space from 0 to X. We can actually search from 0 to X/2, because the floors of square roots don't exceed X/2 (except if X=1).

At each search step, we check if mid's square is less than or greater than X, and move mid accordingly.

Pseudocode:

```
(Note: Never write pseudocode in an actual interview. Unless you're writing a
few lines quickly to plan out your solution. Your actual solution should be in
a real language and use good syntax.)

low = 0, high = X/2
while low <= high
     find mid
     if mid*mid is greater than X:
         high = mid - 1
     else if mid*mid is less than X
         if (mid+1)*(mid+1) > X:
             we've found the floor, return mid
         low = mid + 1
     else // equal to X
         return mid

return -1 // not found, shouldn't happen
```

Test Cases:
Edge Cases: 0
Base Cases: 1, 2, 3
Regular Cases: exact square root, 1 less/more than square root, etc.

Time Complexity: O(log(X))

Space Complexity: O(1)

```
public static int floorSquareRoot(int x) {
    if (x == 0)
        return 0;

    if (x == 1)
        return 1;

    int low = 0;
    int high = x/2;
    while (low <= high) {
        int mid = low + (high - low)/2 ;
        if (square(mid) > x) {
            high = mid - 1;
        } else if (square(mid) < x) {
            if (square(mid+1) > x)
                return mid;
            low = mid + 1;
        } else {
            return mid;
        }
    }

    return -1; // should not happen, you can also throw an exception here
}

/*
 * Helper Function: Ask the Interviewer if they want you to implement.
 * This helper function is more for readability.
 */
private static long square(int x) {
    return x*x;
}
```

Problem: Search for a Peak

Level: Medium

A peak element in an array A is an A[i] where its neighboring elements are less than A[i].
So, A[i - 1] < A[i] and A[i + 1] < A[i].

Assume there are no duplicates. Also, assume that A[-1] and A[length] are negative infinity (-∞).
So A[0] can be a peak if A[1] < A[0].

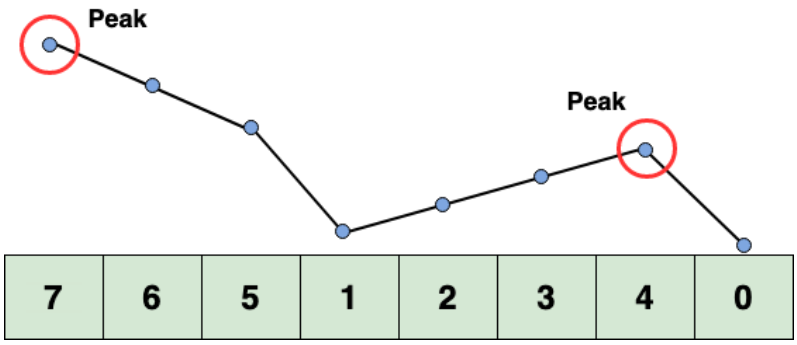
A = [1,3,4,5,2] => Peak = 5
A = [5,3,1] => Peak = 5
A = [1,3,5] => Peak = 5

Questions to Clarify:
Q. Can there be negative numbers in the array
A. Yes, there can be both -ve and +ve numbers

Q. How do we return the output?
A. Return the index of a peak.

Q. What if the array is empty or null?
A. Return -1.

Solution:
It's useful to visualize such problems by drawing a number line. Here is an example of integer array with number lines:



From the above diagram, we see that there can be many peaks in an array. We need to find any one.

Here is an observation: Any element in the array can lead us to a peak. Look at any element in the array. Its slope is either tilted one way or the other. If we follow the slope up, we will eventually reach a peak element.

There are a couple of exceptions of course - if an element is a peak or a valley. If it's a peak,

we've found a result. If it's a valley (1 in the above array), we can go either direction - there's a peak on either side.

So if we sample any element of the array, we can confidently say that there is a peak element either on it's left, or it's right. This smells of binary search.

We look at an element - a[mid]. If the element is sloping up towards the left, we only consider left of mid. If sloping up towards right, we only consider right of mid. If it's a valley, we can go either direction. If it's a peak, we return the result.

Pseudocode:

```
(Note: Never write pseudocode in an actual interview. Unless you're writing a
few lines quickly to plan out your solution. Your actual solution should be in
a real language and use good syntax.)

 low = 0, high = a.length - 1
 while low <= high
     find mid
     left = left neighbor of mid (-Infinity if out of bounds)
     right = right neighbor of mid (-Infinity if out of bounds)

     if left, mid, right are sloping up towards right
         go right
     else if left, mid, right are sloping up towards left
         go left
     else if mid is a valley:
         go either way
     else
         return mid // mid must be peak

  return -1 // not found, should not happen
```

Test Cases:
Edge Cases: empty array, null array
Base Cases: single element
Regular Cases: peak in the middle, peak at a[0], peak at end of array

Time Complexity: O(log(n))

Space Complexity: O(1)

```
public static int findPeak(int[] a) {
    if (a == null || a.length == 0)
        return -1;

    int low = 0;
    int high = a.length - 1;
    while (low <= high) {
        int mid = low + (high - low)/2;

        int left = (mid > 0) ? a[mid-1] : Integer.MIN_VALUE;
        int right = (mid < a.length-1) ? a[mid+1] : Integer.MIN_VALUE;

        if (left < a[mid] && right > a[mid]) {
            low = mid + 1;  // go right
        } else if (right < a[mid] && left > a[mid]) {
            high = mid - 1; // go left
        } else if (right > a[mid] && left > a[mid]) {
            high = mid - 1; // valley, go either way
        } else {
            return mid;
        }
    }

    return -1; // should not happen, you can also throw an exception here
}
```