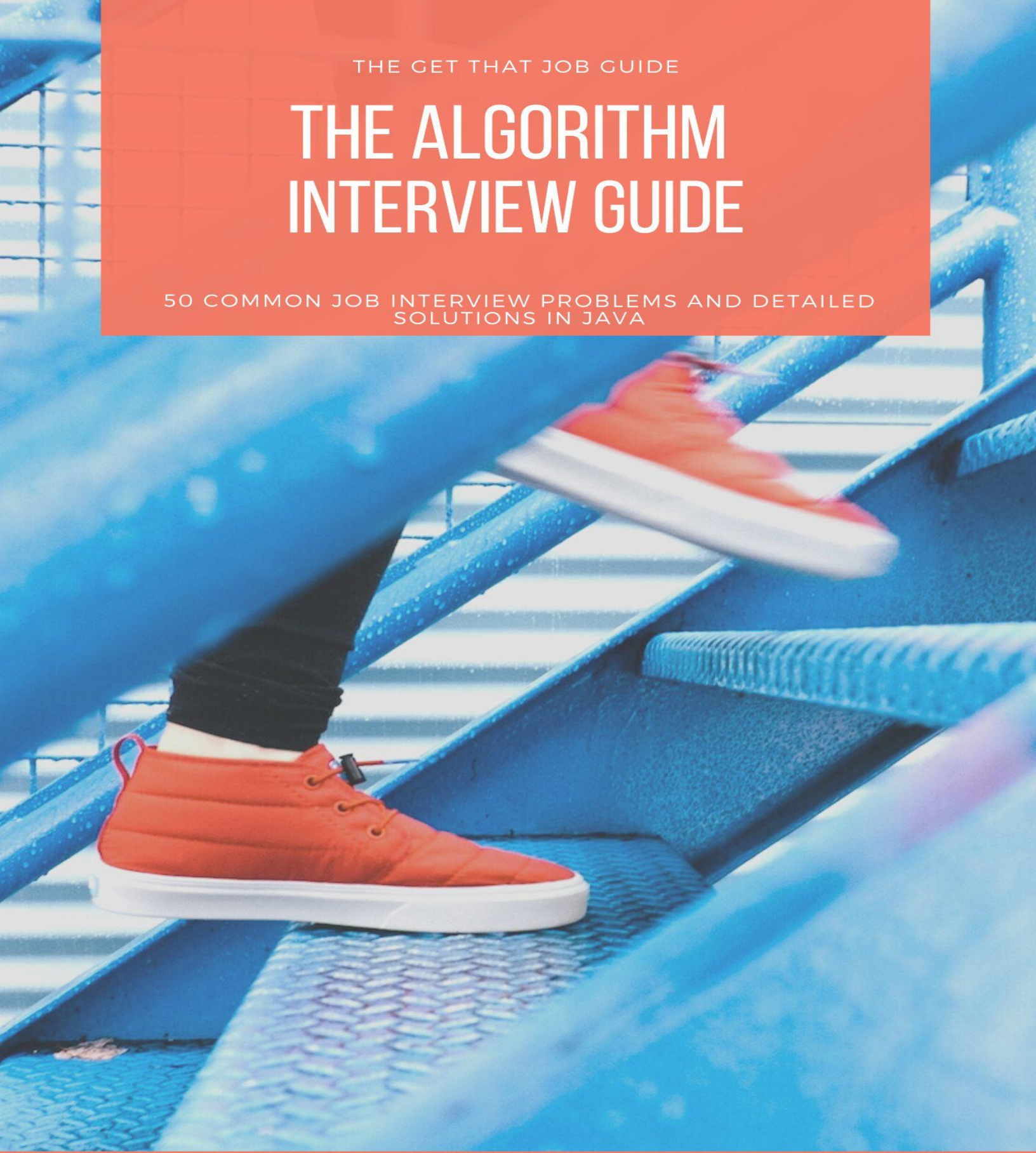
THE GET THAT JOB GUIDE

THE ALGORITHM INTERVIEW GUIDE

50 COMMON JOB INTERVIEW PROBLEMS AND DETAILED SOLUTIONS IN JAVA



RAJESH MOHANAN

A Step by step approach in understanding the solutions to the common Problems

The Algorithm Interview Guide

An interview guide with a detailed step-by-step explanation to the most commonly asked questions and patterns.

Rajesh Mohanan

This book is for sale at http://leanpub.com/algorithm

This version was published on 2023-04-16

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Introduction

Good knowledge of Algorithm and Data Structures is a basic necessity to pass your interview and land a good job. These problems are asked not just at entry-level but even at managerial levels as well.

There are many courses and sites that support you to master this. This book is an addition to fill one gap which I thought is needed.

Over the years of interviewing candidates, I have felt that though they are familiar with the questions, they have not mastered the approach to the solutions. Somehow many candidates are not able to write down an execution flow if asked to.

Many a time I have seen them skipping from one popular approach to another as is commonly available but the fact is **they have not understood the approaches or the solutions to a problem**. A slight change in the question and they are at sea.

What I am aiming for with this book is to help these candidates cross a barrier.

What this book offers you is a set of commonly asked questions in algorithms and a step-by-step approach to understanding the solution.

While this will be a guide for you to learn, Nothing will help you master the topic unless you practice various problems repeatedly.

Please write to me for any review that you have about the book - rajeshmohan.gect@gmail.com

Thank you.

O(n) the Complexities

The Big O - The time complexities

The Big O notation is something that you need to have a basic understanding of to make a good mark in your interviews.

Big O helps us understand and compare algorithms based on their time complexities.

Constant time or O(1)

When the run time of the algorithm is not dependant on the input size and has only one instruction to execute, it is termed as constant time or O(1).

A fetch from a HashMap or an array based on an index is O(1).

Linear time or O(n)

When an algorithm runs once based on the input size, we term it linear or of the complexity of O(n).

As an example, the following algorithm runs through the size of the array once.

```
for(int i = 0; i < array.length; i++){
   System.out.println(array[i);
}
```

Logarithmic or O(logn)

If with each iteration, if you are able to bring down the input size, we have logarithmic complexity or O(logn)

Binary Search is a classical example where we reduce the size of input into half in each iteration.

The number of computations gets increased by the log of the input value.

O(n logn)

Here the algorithm takes n times log n time to complete the task. Most of the sorting algorithms like Quick, Heap and Merge as O(n logn) as its best case.

Quadratic or $O(n^2)$

Generally, when we have a nested for loop it is quadratic or $O(n^2)$.

```
for(i = 0; i < length; i++){
   for(j = 1; j < length; j++){
       //do something
   }
}
```

Consider the length to be 10. Then this above code snippet will execute 100 times.
If it is 100, then it will be 10,000 times. The complexity increases quadratically.

Memory

If our algorithm doesn't need extra space then it is O(1).
By extra space, mostly means that we don't create an extra array or collection. Having extra variables should be fine.

If we have an input array and we create a new array of the same size for our processing then we take O(n) memory.

Moving Zeroes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem definition:

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Move 0's - Brute Force

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Moving 0's - in O(n)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

The Majority Element

Problem Definition

> *Given an array of size n, find the majority element. The majority element is the element that appears more than* $n/2$ *times.*

Now before jumping to solve it, it is always a good idea to ask for clarification in case of a face-to-face/phone interview. If this is a round where your solution is going to be auto-graded then mostly, most of the common doubts are provided along with the question itself.

One obvious clarification is if a given array will contain an element that is occurring more than $n/2$ times.

In this case, we are assuming that this is guaranteed.

One other question that can help you a lot is about the time complexity that is expected. If the complexity is not a concern or if the interviewer is not particular about a solution that is O(n), then there is an easier approach.

You can sort the array and return the element in the middle. Since the element you are looking for is the one that is guaranteed to occur more than n/2 times, the middle element will certainly be the answer. Even if the interviewer mentions upfront about an
O(n) solution, I believe it is still better to mention this solution because it is such a simple solution and a clever approach.

Solution without extra space and running time of $O(nlogn)$

```
1  private int majorityElement(int[] nums){
2     if(nums ==null || nums.length ==0)
3              throw new IllegalArgumentException("Empty or null array not expected");
4     Arrays.sort(nums);//Complexity of O(n logn)
5     return nums[nums.length/2];
6  }
```

```
Consider your input array as [1,2,1,2,1,2,1].
When you call Arrays.sort(), the array is sorted to [1,1,1,1,2,2,2]
Length of the array is 7.
nums.length/2 = 3
nums[3] = 1, which is also the majority element.
```

Now if this needs to be solved in $O(n)$ time, you also need to check about the space complexity. ie you need to know if taking extra space is permitted. Mostly this is going to be OK and if so you can use a Map with the number as the key and the frequency of each element as the value. As soon as you reach a value that is more than half the array, you have your solution.

Solution with Extra Space and running time of $O(n)$

```java
public int majorityElement(int[] nums) {
   Map<Integer, Integer> freq = new HashMap<>();
   for(int i : nums){
       Integer count = freq.get(i);
       if(count == null)
           count = 0;
       count++;
       if(count == nums.length/2 + 1)
           return i;
       freq.put(i, count);
   }
   return -1;
}
```

Let us see the working of this algorithm in steps:

```
Let us consider the same array [1,2,1,2,1,2,1]
And let us examine the for loop.
i = 1
   freq map is empty so freq.get(1) gives null.
   count = 1 after the increment.
   1 != 4 so add add the value in the Map.
   freq = {(1=1)}
i = 2
  freq.get(2) gives Null.
   count = 1 after the increment.
   1 != 4 so add add the value in the Map.
   freq = {(1=1), (2=1)}
i = 1
   freq.get(1) gives 1.
```

```
   count = 2 after the increment.
   2 != 4 so add update the value in the Map.
   freq = {(1=2), (2=1)}
i = 2
   freq.get(2) gives 1.
   count = 2 after the increment.
   2 != 4 so add add value in the Map.
   freq = {(1=2), (2=2)}
...
Finally at the last element,
i = 1 and the current freq Map vaues are {(1=3, (2=3))}
   freq.get(1) gives 3.
   count = 4 after the increment.
   4 == 4 so return i, ie 1
```

Now in case if you are asked to solve it in O(n) without using extra space, there is a solution and it is called **Boyre Moore algorithm**. If they need such a solution then this is something you need to know beforehand. I don't think you are expected to come with this solution on your own within the short span of the interview. And so normally they should be OK with either of the above solutions. Now just in case, you need to learn it, let us check it out.

> This is how the author of the algorithm explains it[1]
>
> And they have also provided a step by step explanation[2]

Now having seen the working of the algorithm, let us have the code and also see it applied for our example.

```
1  public int majorityElement(int[] num) {
2     int majority=num[0];
3     int count = 1;
4     for(int i=1; i<num.length;i++){
5         if(count==0){
6             count++;
7             majority=num[i];
8         }
9         else if(majority==num[i]){
10            count++;
11        }
12        else count--;
13    }
```

[1] http://www.cs.utexas.edu/~moore/best-ideas/mjrty/index.html
[2] http://www.cs.utexas.edu/~moore/best-ideas/mjrty/example.html

```
14      return majority;
15  }
```

Let us see how this works for our example

1 2 1 2 1 2 1

```
Initialize the variable majority to nums[0], ie majority = 1.
count = 1

Inside for loop:
i = 1
   decrement count by 1, count = 0 // //else block
i = 2
   increment count by 1, // if(count == 0)
   count = 1
   majority = 1

i = 3
   Decrement count by 1, //else block
   count = 0
i = 4
   increment count by 1, // if(count == 0)
   count = 1
   majority = 1
i = 5
   Decrement count by 1, //else block
   count = 0
i = 6
   increment count by 1, // if(count == 0)
   count = 1
   majority = 1

End of for loop
Return majority, ie 1
```

You can create a different array with a majority element and try working out again to make your understanding better and also to clear some doubts that you might have with this algorithm. When I first encountered this algorithm, I tried solving about 4-5 different variations to make myself understand the concept.

Sub Array With the Maximum Sum Of Size k

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

What is the Sliding Window approach?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Longest Sequence With k Distinct Characters

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Sum of Pairs With Target Sum

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

The Dutch National Flag Problem

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Add 2 Linked Lists

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Remove Duplicates From a Linked List

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

A brief Introduction to Recursion

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Is the Array a Palindrome

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Print the Perimeter of a Binary Tree

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Defintion

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Level Order Successor

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Lowest Common Ancestor of a Binary Search Tree

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

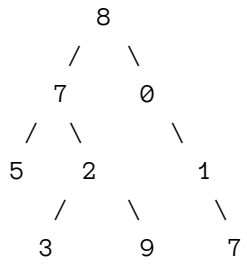
LCA of BST - Recursive

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Lowest Common Ancestor of a Binary Tree

Problem Definition

Given a Binary Tree and 2 nodes in it, find the lowest common ancestor of the nodes.



For nodes 5 and 2, the LCA is 7

Similarly, for 3 and 9, it is 8 which also happens to be the root node

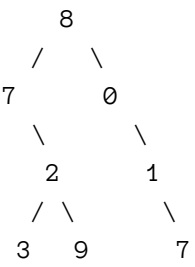
And for 0 and 7, it is 0 and not 8 as a node is also a descendant of itself

Well, this is a very popular question in the interview circuit for many top companies. The point to keep a note of is that there are no pointers from a child to its parent. Had there been, then we could have tried going up the tree till you find a common point. Now there are ways to store the parent nodes in a data structure and then iterate through it to figure out the common node.

But What I am presenting here is a very simple straightforward recursive solution. If you don't get it first, please work your way through a couple of more examples the way I explain it here. The solution is simple

What we do is iterate the left side till you reach a leaf or one of the given nodes. Once you have a value for the left node, we do the same to the right node.
Once we have both left and right nodes, we backtrack to the common ancestor
It might sound a bit complex but let us use an example to understand this further.

```java
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
   if (root == null || root == p || root == q) return root;
   TreeNode left = lowestCommonAncestor(root.left, p, q);
   TreeNode right = lowestCommonAncestor(root.right, p, q);
   return left == null ? right : right == null ? left : root;
}
```



Let us find the ancestors of 3 and 1

```
lowestCommonAncestor(8, 3, 1)
    left = lowestCommonAncestor(7, 3, 1)
        left = lowestCommonAncestor(5, 3, 1)
            left = lowestCommonAncestor(null, 3, 1)
           root == null
                return null
            left = null
            right = lowestCommonAncestor(null, 3, 1)
                   root == null
                            return null
            right = null
            return null //if left is null, return right
       left = null
       right = lowestCommonAncestor(2, 3, 1)
               left = lowestCommonAncestor(3, 3, 1)
                   root == left
                       return 3
               left = 3
               right = lowestCommonAncestor(9, 3, 1)
                   left = lowestCommonAncestor(null, 3, 1)
                       root == null
                           return null
                   right = lowestCommonAncestor(null, 3, 1)
                       root == null
                           return null
```

```
                return null
            right = null
            return left //left is not null, right is null. return left
            return 3
    right = 3
    return right
    return 3//we have the left side covered. now move to right
left = 3
right = lowestCommonAncestor(0, 3, 1)
        left = lowestCommonAncestor(null, 3, 1)
            root == null
                return null
        left = null
        right = lowestCommonAncestor(1, 3, 1)
            root == right
                return 1
        right = 1
        return right//left is null, return right
        return 1
right = 1
return root//if left is not null and right is not null, then return root
return 8
```

Sum of All Paths of a Binary Tree

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Top k Frequent Number

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Maximum Profit

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Longest Palindromic Subsequence

Problem Definition

Longest Common Subsequence

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Counting Negative numbers from a Sorted Matrix

Problem Definition

> *Given an m x n matrix that is sorted in non-increasing order both row-wise and column-wise, return the number of negative numbers in the grid.*

Let me explain this with sample input and the expected output

Input:
[8,7,6,**-1**]
[7,6,1,**-1**]
[1,1,**-1**,**-2**]
[**-1**,**-1**,**-2**,**-3**]

Output = 8

Let us check another one

[1,**-1**]
[**-1**,**-1**]

Output = 3

Now at one level, the question appears easy. You can actually traverse the matrix, visit each element and see if they are negative. You can actually achieve the result in O(mn) time.

But then there has to be something more to this, right? What would be an easier way to get this than O(mn)?
For that let us think what would be a better complexity that we can think that would be better than O(mn)?

Think about it a bit before you proceed. The fact that the matrix is sorted is significant in getting a better solution.

The approach we can take is that of Binary Search. Note that Binary Search and its variants are a commonly used technique in solving many problems other than the regular search that you would have studied.

A quick refresher:
So how does Binary Search work?

Consider you have a sorted array and you are asked to see if an element exists or not.

1 4 5 6 8 9 10 12 13 15 16 17 18 19 21 23 24

Let us say that we need to see if 16 exists in this array or not.

The normal Binary Search, will see the mid-value of the array and compare that with 19.

The mid-value here is 13.
13 is lesser than 16, so we know that 16 is coming towards the right side of 13. So we can effectively ignore the numbers to the left of 13.

Now if you can re-imagine your array as

13 15 16 17 18 19 21 23 24

Now the mid element of this array is 18

18 is greater than 16, and so we can ignore whatever numbers are there to the right of 18

Now the array is |13|15|16|17|18|

The mid-value is 16 and we now know that the element exists.

What it took were 3 steps.

So now let us see how to employ the same technique to our problem.

For each row, we need to count the number of negative numbers.
We will do that mainly by using Binary search.
We will also use some optimization so that we need not go to the Binary Search if the row constitutes only positive or only negative numbers.

Let us see the code now

```
public int countNegatives(int[][] matrix) {
   int rows = matrix.length; //total rows
   int cols = matrix[0].length; //total columns
   int negativeCounts = 0; // variable to hold the count of negative numbers
   int lastNeg = cols - 1; //last negative number index



   //Iterate row wise
    for (int row = 0; row < rows; row++) {
        //If the first element of a row is negative, then the whole row is negative.
        //Add the size of the row to total and proceed to next row
        if (matrix[row][0] < 0) {
                negativeCounts+=cols;
```

```
15                  continue;
16          }
17
18          //If the last element of a row is positive, then there are no negative
19          //numbers in this row to to be counted. Proceed to next row.
20          if (matrix[row][cols - 1] > 0)
21                  continue;
22
23          //Now we know that the current row is a mix of positive and negative and we
24          //need to count it. This is where we employ the Binary Search algorithm. Note
25          //that there is no element that  we are searching for but just finding the
26          //starting of negative number in the row
27          int left = 0, right = lastNeg;
28          while (left <= right) {
29             /*use this way instead of (left + right)/2 to avoid overflow*/
30                  int mid = left + (right - l)/2;
31
32                  if (matrix[row][mid] < 0) {
33                      right = mid - 1;
34                  } else
35                      left = mid + 1;
36                  }
37                  //left points to the first negative element so cols - left is the number
38                  //of negative numbers in this row
39                  negativeCounts += (cols - left);
40                  lastNeg = left;
41          }
42          return negativeCounts;
43      }
44  }
```

Let us see the program flow

```
[8,7,6,-1]
[7,6,1,-1]
[1,1,-1,-2]
[-1,-1,-2,-3]

rows = 4
cols = 4
negativeCounts = 0
lastNeg  = 3
```

```
for loop:
row = 0:
    if : 7 is not less than 0, proceed
    if : -1 is not greater than 0, proceed

    left = 0
    right = 3
    start while loop
    0 <= 3:
        mid = 1
    7 is not < 0
    left = 2
     2 <= 3
    mid = 2
    6 is not < 0
    left = 3
     3 <= 3
    mid = 3
    -1 < 0
              right = 2
     3 >2. Break while
     negativeCounts = 4 - 3 = 1
    lastNeg = 3

row = 1
    if : 8 is not less than 0, proceed
    if : -1 is not greater than 0, proceed

    left = 0
    right = 3
    start while loop
    0 <= 3:
        mid = 1
    6 is not < 0
    left = 2
     2 <= 3
    mid = 2
    1 is not < 0
    left = 3
     3 <= 3
    mid = 3
    -1 < 0
              right = 2
```

```
     3 >2. Break while
     negativeCounts =1 + 4 - 3 = 2
    lastNeg = 3

row = 2
   if : 1 is not less than 0, proceed
    if : -2 is not greater than 0, proceed

    left = 0
    right = 3
    start while loop
    0 <= 3:
        mid = 1
    1 is not < 0
    left = 2
     2 <= 3
    mid = 2
    -1 < 0
              right = 1
     2 > 1, Break while

     negativeCounts = 2 + 4 - 2 = 4
     lastNeg = 2

row = 3
   -1 < 0
    negativeCounts = 4 + 4 = 8

negativeCounts = 8
```

Valid Parenthesis Sequence

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Merging Intervals

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Check for Conflicting Intervals

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

In-place sorting

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Find the Duplicate Number From an Array

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Additional excercise

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Finding Median

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Statement

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Find All the Subsets

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Find all the Permutations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Search in an Infinite Array

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Probleam definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Find the kth Smallest Number

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Find the Number of Islands

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Grouping Anagrams

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Search a rotated Array

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Sum of Deepest Node

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Add 2 binary numbers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Longest Common Prefix

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Common Elements of 3 Arrays

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Flood Fill

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Find the Nth Highest number from a Binary Search Tree

Problem Definition

Evaluate an arithmetic expression

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Make a fully connected network

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Mirror a Binary Tree

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Probleam Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Starting and Ending index of an element in a Sorted Array

Problem Definition

Merge Two Sorted Arrays

Problem Definition

ZigZag String conversion

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Represent Number in words

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Construct Binary Tree from preorder and inorder traversal inputs

Problem Definition

Rotate Image

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Sort a given Array using Quick Sort

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Merge Sort

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.

Problem Definition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/algorithm.