Workshop: Software Engineering Interview Preparation

WOMEN WHO CODE PORTLAND

alchemy code lab

Agenda

Intros:

WWCode @ Alchemy Code Lab

Link to slides: https://github.com/wwcodeportland/study-nights/tree/master/

Agenda:

9:30 - 10:00 Networking

10:00 - 12:00 How to go about solving algorithmic problems + Practice

12:00 - 1:00PM Lunch

1:00 - 5:00 Mock technical interviews with mentors

Leadership Team

Caterina
Director

Richa
Director

Sarah Joy
JavaScript Lead

Shiyuan
Design Lead

Tricia
DevOps Lead

Keeley
Open Source Lead

Alia
Algorithms Lead

Michelle
Networking Nights Team

Posey
Community Engagement Manager

{short} Code of Conduct

Women Who Code (WWCode) is dedicated to providing an empowering experience for everyone who participates in or supports our community, regardless of gender, gender identity and expression, sexual orientation, ability, physical appearance, body size, race, ethnicity, age, religion, socioeconomic status, caste, or creed. Our events are intended to inspire women to excel in technology careers, and anyone who is there for this purpose is welcome. Because we value the safety and security of our members and strive to have an inclusive community, we do not tolerate harassment of members or event participants in any form. Our **Code of Conduct** applies to all events run by Women Who Code, Inc. If you would like to report an incident or contact our leadership team, please submit an **incident report form**.

Upcoming Events - May

- Ladies Computative Cupcakes & Consumption – Sun, May 6th, 11:00 AM
- Roll Call: Microsoft Build 2018 – Mon, May 7th, 8:80 AM
- JavaScript Study Night @Act-On Software – Wed, May 9th, 5:30 PM
- Women Who Run: Doggie Dash 2018! – Sat, May 12th, 7:30 AM
- Diversity in Tech: Creating a Culture of Inclusion and Equity @WeWork
  – Tue, May 15th, 6:00 PM
- Roll Call: ML4ALL 2018 – Sun, May 27th, 9:00 AM

Resources

WWCode @ Meetup.com

WWCode @ Slack

WWCode @ Github

Big-O CheatSheet

How to Approach Solving Algorithms

1. DON’T start with your computer !!!

2. Write down inputs/outputs.
   Make sure you know all of the information about the problem you are solving.

3. Try a single example by hand to see expected results.

4. Think about any possible edge cases.

5. High level solution - pseudocode. Optimize if you can

Continued...

6. Choose your data structure.
 How often will you be accessing? How often will you be inserting/deleting?
Data Structure complexity.

7. Code your solution

8. Discuss the algorithm complexity

9. Optimize if possible

Let’s try that together

```
                              Sample Problem

Write a function that tells whether or not a given number is part of a
listing. Numbers in that listing are sorted.
```

Step-by step example

1. DON’T start with your computer !!!

Step-by step example

2. Write down inputs/outputs.

“Write a function that tells whether or not a given number is part of a listing of numbers. Numbers in that listing are sorted.”

Inputs: array of integers + integer

Output: boolean

Step-by step example

3. Try a single example by hand to see expected results.

If input is

1 4 6 17 46 78 79 178 77777 , 178

Output is

true

Step-by step example

4. Think about any possible edge case.

Main concerns depending on language:

- null/undefined inputs
- empty inputs
- overflow (ex: number getting higher than MAX_VALUE)
- if types are defined and don’t represent the meaning of the data:
    - String for integer
    - Object for Integer

Step-by step example

5. High level solution - pseudocode.

“Write a function that tells whether or not a given number is part of a listing of numbers. Numbers in that listing are sorted.”

Brute force:

```
read each number in the array
    compare with the number you are trying to find
```

THERE IS NO TASK TOO SIMPLE,
FOR OUR STAFF TO OVER-COMPLICATE
memegenerator.net

Step-by step example

5. High level solution - pseudocode.

“Write a function that tells whether or not a given number is part of a listing of numbers. Numbers in that listing are sorted.”

Brute force:

```
    read each number in the array smaller than the number to find
            compare
```

Step-by step example

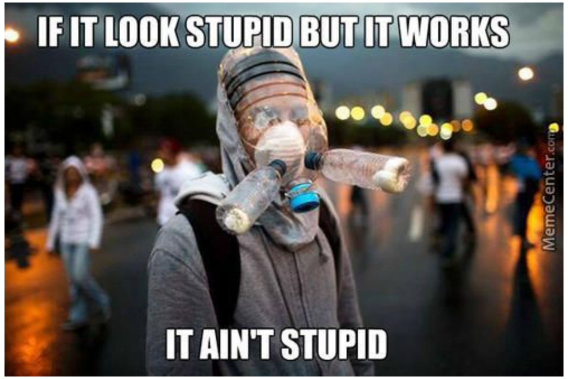
6. Choose your data structure.
How often will you be accessing? How often will you be inserting/deleting? Data Structure complexity.

Pseudo-code :

```
    read each number in the array smaller than the number to find
        compare
```

No need to manipulate the data here

Step-by step example



7. Code your solution

Pseudo-code:

```
    read each number in the array smaller than the number to find
        compare
```

name of function

name of input

name of variable

for or while loop ?

```
boolean findNumber(int[] listing, int number){
 for(int i=0; i<listing.length && listing[i]<=number; i++){
             if(listing[i] == number) return true;
  }
  return false;
 }
```

Step-by step example

8. Discuss the algorithm complexity   for loop => o(n)

9. Optimize if possible

What if instead of reading all the numbers we only looked at the first last and middle numbers ?

1 4 6 17 46 78 79 178 77777, 178

46 78 79 178 77777

79 178 77777

“Divide and conquer”
Big gain for big arrays. Here 3 iterations instead of 6

Step-by step example

6. Code your solution

```
Pseudo-code:
    at each iteration
        look at the middle and pick a side
```

Step-by step example

7. Code your solution

```
Pseudo-code:
    at each iteration
        look at the middle and pick a side

boolean findNumber(int[] l, int nb){
   if(l.length == 0) return false;
   int minInd = 0;
   int maxInd = l.length-1;
   if(nb==l[minInd] || nb==l[maxInd]) return true;
   while (minInd != maxInd){
       int midInd = minInd + maxInd-minInd/2;
       if(nb==l[midInd])return true;
       if(l[midInd] > nb) maxInd = midInd;
       else minInd = midInd;
   }
   return false;
}
```

Step-by step example

8. Discuss the algorithm complexity   for loop => o(log(n))

9. Optimize if possible

Practice

1 Leetcode

- Product of Array except self

2 Hackerrank

- Diagonal Difference
- Day of the programmer

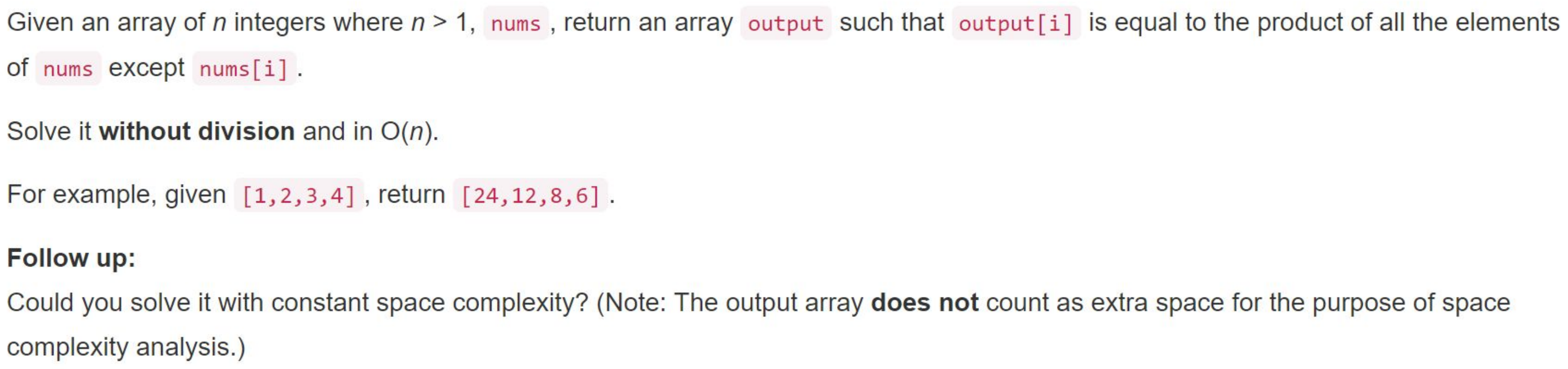
3 Google code jam

- Fair and square

Other platforms for practicing algorithms

- CodinGame
- Code Wars

Product of Array except self



Given an array of *n* integers where *n* > 1, `nums`, return an array `output` such that `output[i]` is equal to the product of all the elements of `nums` except `nums[i]`.

Solve it **without division** and in O(*n*).

For example, given `[1,2,3,4]`, return `[24,12,8,6]`.

Follow up:
Could you solve it with constant space complexity? (Note: The output array **does not** count as extra space for the purpose of space complexity analysis.)

Diagonal difference

Given a square matrix, calculate the absolute difference between the sums of its diagonals.

Input
The first line contains a single integer, N denoting the number of rows and columns in the matrix.
The next N lines denote the matrix's rows, with each line N containing space-separated integers in [-100;100] describing the columns.

Sample Input

```
3
11 2 4
4 5 6
10 8 -12
```

Sample Output

```
15
```

Explanation

The primary diagonal is:

```
11
  5
    -12
```

Sum across the primary diagonal: 11 + 5 - 12 = 4

The secondary diagonal is:

```
    4
  5
10
```

Sum across the secondary diagonal: 4 + 5 + 10 = 19

Difference: |4 - 19| = 15

Note: |x| is the absolute value of x

Day of the programmer

Marie invented a Time Machine and wants to test it by time-traveling to visit Russia on the Day of the Programmer (the day of the year) during a year in the inclusive range from 1700 to 2700.

From 1700 to 1917, Russia's official calendar was the Julian calendar; since 1919 they used the Gregorian calendar system. The transition from the Julian to Gregorian calendar system occurred in 1918, when the next day after January 31st was February 14th. This means that in 1918, February 14th was the 32nd day of the year in Russia.

In both calendar systems, February is the only month with a variable amount of days; it has 29 days during a *leap year*, and 28 days during all other years. In the Julian calendar, leap years are divisible by 1918; in the Gregorian calendar, leap years are either of the following:

- Divisible by 400.
- Divisible by 4 and *not* divisible by 100 .

Given a year, find the date of the 256th day of that year *according to the official Russian calendar during that year*. Then print it in the format dd.mm.yyyy, where dd is the two-digit day, mm is the two-digit month, and yyyy is the year.

Input

A single integer denoting year.

Sample Input	**Sample Output**
2017	13.09.2017

Explanation
In the year 2017, January has 31 days, February has 28 days, March has 31 days, April has 30 days, May has 31 days, June has 30 days, July has 31 days, and August has 31 days. When we sum the total number of days in the first eight months, we get 243. Day of the Programmer is the 256th day, so then calculate 256-243=13 to determine that it falls on day 13 of the 9th month (September). We then print the full date in the specified format, which is 13.09.2017.

Fair and square

Little John likes palindromes, and thinks them to be fair (which is a fancy word for nice). A *palindrome* is just an integer that reads the same backwards and forwards - so 6, 11 and 121 are all palindromes, while 10, 12, 223 and 2244 are not (even though 010=10, we don't consider leading zeroes when determining whether a number is a palindrome).

He recently became interested in squares as well, and formed the definition of a *fair and square* number - it is a number that is a palindrome **and** the *square of a palindrome* at the same time. For instance, 1, 9 and 121 are fair and square (being palindromes and squares, respectively, of 1, 3 and 11), while 16, 22 and 676 are **not** fair and square: 16 is not a palindrome, 22 is not a square, and while 676 is a palindrome and a square number, it is the square of 26, which is not a palindrome.

Now he wants to search for bigger fair and square numbers. Your task is, given an interval Little John is searching through, to tell him how many fair and square numbers are there in the interval, so he knows when he has found them all.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains two integers, **A** and **B** - the endpoints of the interval Little John is looking at.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is the number of fair and square numbers greater than or equal to **A** and smaller than or equal to **B**.