Interview Questions

Interview Question 1 - Gold Stars

Alice is a teacher with a class of *n* children, each of whom has been assigned a numeric rating. The classroom is seated in a circular arrangement, with Alice at the top of the circle. She has a number of gold stars to give out based on each child's rating, but with the following conditions:

- Each child must receive at least one gold star
- Any child with a higher rating than his or her immediate neighbor should get more stars than that neighbor

Assuming *n* >= 3, what is the minimum total number of stars Alice will need to give out?

Write a program which takes as its input an `int[]` containing the ratings of each child, ordered by seating position, and returns an `int` value for the minimum total number of stars that Alice will need to give out.

Hint: this problem can be solved with an algorithm that runs in *O(n)* time.

For example:

In a class of three children with ordered ratings of `{1, 2, 2}`, Alice will need to give out `{1, 2, 1}`stars accordingly, for a total number of `4` stars overall.

NOTE: You should be able to implement this in pure Java using no imports, helper functions, or collections classes!

Bonus 1

In the above example, child #3 has the same rating as child #2 but gets fewer stars. To be equitable, the number of stars should be `{1, 2, 2}`, resulting in a total number of `5` stars overall.

Modify the algorithm so that any child with fewer stars than an immediate neighbor with an equal rating gets at least as many stars as that neighbor.

Interview Question 2 - Text Justification

Given an natural language text, write an algorithm which will format the text by splitting it into lines of a specified length, with each line justified such that the text is evenly spaced according to specific rules.

Inputs:

- `String text` - the text to format
- `int length` - the line length for the output

Output: a `String[]` that meets the following requirements:

- Each line should have exactly `length` characters, including whitespace
- Each line should contain as many words as will fit, and the algorithm should return as few lines as possible
- Each line should be padded with spaces, distributed as evenly as possible
- If the number of spaces required to pad a line do not divide evenly between words, add the extra spaces between the leftmost words first
- The last line of the output should be left justified, with no extra spaces between words and all padding added to the right.

Example:

Given the inputs:

```
String text = "This is an example of text justification.";
int length = 16;
```

...the output array should be:

```
String[] output = {
  "This    is    an",
  "example  of text",
  "justification.  "
};
```

Hint: you can use `String.split(" ")` to tokenize the input into words.

Remember to avoid helper functions and libraries, though you can use methods in`java.lang.String` and classes in `java.lang` and `java.util`

Bonus 1

Add support for both left and right justification via a parameter. Use an `enum` to specify values for all three use cases, with the above us case as the default. For left and right justification, the last line should be justified the same as the rest of the text.

Using the above example inputs, the outputs for these two variants would look like this:

Left justification:

```
String[] output = {
  "This is an      ",
  "example of text ",
  "justification.  "
};
```

Right justification:

```
String[] output = {
  "      This is an",
  " example of text",
  "  justification."
};
```

NOTE: You are not allowed to use any libraries or specialized functions to complete the challenges below.

Challenge 1 - Deleting duplicates from a sorted array

This problem is concerned with deleting repeated elements from a sorted array.

Write a program which takes as input a sorted `int[]` and updates it such that:

- all duplicates have been removed and
- all remaining valid elements have been shifted left to fill the emptied indices
- all remaining empty indices have values set to `0`

- the function returns the number of remaining valid elements (the array size minus the number of removed elements)

For example, given an input array with the values `{2,3,5,5,7,11,11,11,11,13}`, after the function completes, the values in the array should be `{2,3,5,7,11,13,0,0,0}`, and the function should return `6`.

Hint: There is an *O(n)* time and *O(1)* space solution.

Challenge 2 - Enumerate all primes <= `n`

A prime number (or a prime) is an integer greater than 1 that has no positive divisors other than 1 and itself.

Write a program which takes as input an `int` value `n` and returns an array of `int` containing all unique primes <= `n`.

Example: if the value of `n` is `8`, the function should return: `{2,3,5,7}`

Hint: One well-known algorithm for doing this is over 2,000 years old, but it's not the most efficient.

Remember, you are not allowed to use any primality testing functions.

Challenge 3 - Spiral Order

A matrix is a two-dimensional array of *r* rows, each with *c* columns, such that the total number of elements in the matrix is *r * c*.

The spiral order of such a matrix is the list of all its elements starting at index *(0, 0)* and proceeding in clockwise order from the outermost values to innermost values.

Write a program that takes an `int[][]` matrix as its input and returns an `int[]` of all the input's values in spiral order.

Example: Given the following matrix:

```
int[][] matrix = {
  { 1, 2, 3 },
  { 4, 5, 6 },
  { 7, 8, 9 }
};
```

Your program should return `{1,2,3,6,9,8,7,4,5}`

Challenge 4 - Palindrome detection

A palindrome is a word, phrase, or sequence of characters that reads the same backward as forward, e.g., *madam* or *nurses run*.

Write a program which takes a `String` as input and returns a `boolean` value which is `true` if the input is a palindrome and `false` if it is not, considering only alphanumeric characters and ignoring case.

Example:

- `"A man, a plan, a canal: Panama"` is a palindrome and should return `true`
- `"race a car"` is not a palindrome and should return `false`

Challenge 5 - Longest Palindromic Substring

Write a program which takes a `String` as input and returns a `String` which is the longest palindromic substring in the input, given the following assumptions about the input string:

- its maximum length is 1000
- it contains one unique, longest palindromic substring

Examples:

- `"abdbabbdba"` should return `"abdba"`
- `"abdbbbbdba"` should return `"abdbbbbdba"`

Challenge 6 - Longest Common Prefix

Write a program which takes a `String[]` as input and returns a `String` which is the longest common prefix, or an empty string if there is none.

Examples:

- `{"bceefgh", "bcfghijk", "bcefgh"}` should return `"bc"`
- `{"abcdefgh", "aefghijk", "abcefgh"}` should return `"a"`
- `{"", "aefghijk", "abcefgh"}` should return `""`

Interview Tips

When you're working through a problem, make sure you're following the stages below, in order.

1. Take a deep breath and relax. Interviews are stressful for everyone, and there's a time pressure that you don't normally have when you're working, and it's natural to be anxious. Anxiety makes it hard to think clearly, so make an effort to calm yourself, so you can tackle the problem with a clear mind.
2. Understand the problem. Don't jump into the solution immediately. Take a moment to make sure you fully understand the problem and restate the problem to the interviewer to make sure you've understood it correctly. Many of the more complex problems are deliberately ambiguous for edge cases, and the interviewer is looking for your ability to analyze the problem.
3. Create test cases. Before moving on to implementation, create test cases, and write them on the whiteboard. Start with simple test cases before getting into edge cases. Write down at least 5 test cases before moving on. After you finish your solution, your final step will be to run your solution through the test cases. This also helps you confirm that you've successfully understood the problem.
4. Overview strategy before pseudocoding. It's easy to get bogged down in coding or pseudocoding. Discuss your basic plan before you write anything down.
5. Start with a simple strategy before optimizing. It's often better to warm up with a more naive strategy first, even it's not as efficient. Once you have that, you can work on creating a better strategy.
6. Consider sorting or indexing first. The trick for many problems is to organize your data first, either by sorting it or putting the data in a dictionary. This will often make your implementation strategy simpler as well as more efficient.
7. Pseudocode before coding. It's often better to start with pseudocode, so you can work through your strategy quicker. You can mingle code with pseudocode if you don't remember the exact syntax, and make a final pass at the end to fix syntax. Often, your interviewer cares the least about your ability to get correct language syntax, so don't waste too much time on it, so you can spend more time optimizing your solution.
8. Evaluate your correctness. After completing your solution, step through the test cases that you defined earlier and evaluate if they pass. Try to spot any additional test cases that could trip up your solution.
9. Evaluate your efficiency. Evaluate the big O efficiency of your solution, as well as its memory usage.

When you get stuck...

- Take a moment. Once you get stuck, it's easy to start panicking, which guarantees that you're not going to be at your best. If you find yourself going in circles, take a step back, stop thinking about the problem, and try to relax before jumping back into it.
- Don't stop talking. Try to avoid being silent at all times in the interview. Be as chatty as you can about everything you're observing about the problem as you're working toward a solution. Talk aloud about approaches that you're thinking about, even if you think it's not correct. The interviewer generally wants to help you, but it's hard to help someone who's not talking. They can usually give you a good hint if they know exactly what you're thinking. Also, even if you don't solve the problem in the end, you'll be able to showcase your problem solving skills.