ARRAY AND STRING EXERCISES

In this series of exercises, we'll get you up and running with arrays and strings.

In the guide, we will quickly highlight the key concepts that you need to know for your interview. If you're unfamiliar with any of these, I highly recommend taking some time to review them.

Then, I've provided you with a series of exercises to help you get up and running quickly. I recommend setting aside 2-3 days to work through these exercises. Simply start from the top and work through them as much as you have time.

Make sure to track the time you spend here in your Interview Prep Tracker. You can categorize this as "Study" time.

For these exercises, we have complete code samples and templates available here.

TABLE OF CONTENTS

KEY STRING AND ARRAY PATTERNS

- **Accessing and Modifying values**
  While this may seem trivial for Arrays and Strings because we're so used to these basic operations, they form the foundation of working with any data structure. Without being able to efficiently access the data, it is impossible to work with the data.

- **Using multiple pointers**
  Multiple pointers are one of the most common ways that we utilize Arrays and Strings. They allow us to do everything from look at substrings to compare two different values in an array, and everything in between.

- **Comparisons**
  Often Arrays and Strings are used to store data fundamental to our larger goal. Therefore, comparing them to find the similarities and differences is very useful.

  Importantly, there are many different types of comparisons we may want to do. Exact comparisons are relatively easy, but fuzzy comparisons can be a lot harder. We will focus on exact comparisons in this section and get to more advanced materials once we discuss recursion.

- **Sliding Windows**
  Sliding windows allow us to efficiently work with subarrays and substrings. Whenever we want to do any sort of comparison, rather than having to recompute for each substring, a sliding window allows us to reuse the work that we've already done.

ARRAY AND STRING EXERCISES

Exercise Set #1 - Accessing and Modifying Values

1. Write a function that takes an integer array and reverses the values in place.

   eg.
   ```
   arr = [1, 2, 3, 4, 5]
   f(arr)
   arr = [5, 4, 3, 2, 1]
   ```

   What is the time and space complexity of your solution?

2. Write a function that takes in a string and removes every odd-indexed character.

   Be careful that this runs in linear time. Remember that if strings are immutable, each time you modify the string is an `O(N)` operation.

   eg.
   ```
   str = “iloveinterviewprep”
   output = “ioenevepe”
   ```

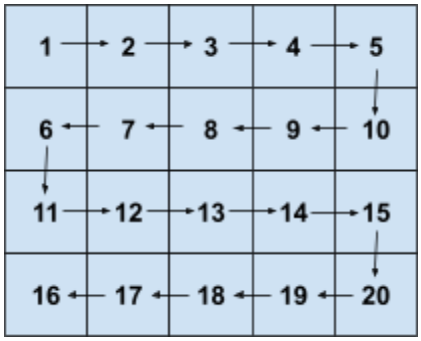
   What is the time and space complexity of your solution?

3. Write a solution to the Zig Zag Conversion problem (Full Problem Definition)

   What is the time and space complexity of your solution?

[Hints for the following exercises on the next page]

4. Given a 2D matrix, write a function to print the values in the following order:

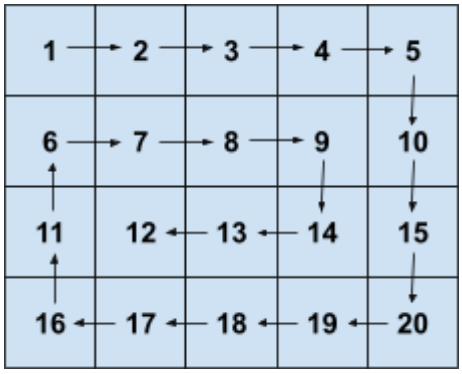



```
output = 1, 2, 3, 4, 5, 10, 9, 8, 7, 6, 11, 12, 13, 14, 15, 20, 19, 18, 17, 16
```

What is the time and space complexity of your solution?

5. Given a 2D matrix, write a function to print the values in the matrix in a clockwise spiral from outside to inside.

.




```
output = 1, 2, 3, 4, 5, 10, 15, 20, 19, 18, 17, 16, 11, 6, 7, 8, 9, 14, 13, 12
```

What is the time and space complexity of your solution?

6. Given a 2D matrix, write a function to print the values in the matrix along the diagonals (see explanation below).

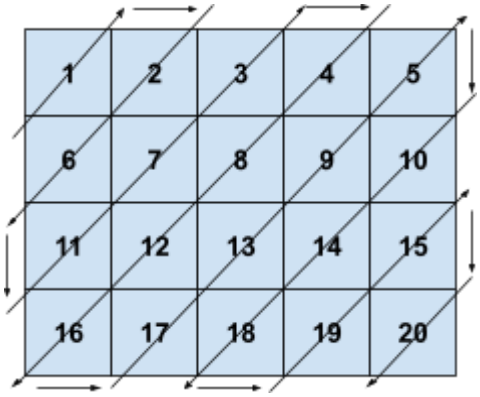
   eg.

   arr =

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

   `output = 1, 2, 6, 11, 7, 3, 4, 8, 12, 16, 17, 13, 9, 5, 10, 14, 18, 19, 15, 20`

   explanation:



   What is the time and space complexity of your solution?

HINTS FOR EXERCISES 1.4 - 1.6

1. How can you break the solution up into multiple steps? For example, with the spiral, can you implement the iteration across to the right, iteration down, iteration across to the left, and iteration up as 4 sequential steps?

2. If you implement each iteration as a separate step, how do you know where each iteration is supposed to start and end?

3. If you are still stuck on these problems, review the solutions here and then attempt them again.

Exercise Set #2 - Using Multiple Pointers

1. Given a string, write a function that prints out all of the substrings.

   eg.
   ```
   str = “abc”
   output:
   a
   ab
   abc
   b
   bc
   c
   ```

   What is the time and space complexity of your solution?

2. Write a function to find duplicate in an array. The array will contain exactly 1 duplicated value. (Use 2 pointers for this. Don’t add values to some extra data structure)

   eg.
   ```
   arr = [1, 2, 3, 2]
   output = 2
   ```

   What is the time and space complexity of your solution?

3. Given a sorted array, find every pair of values in the array that sum up to a given target. If the same pair occurs multiple times, it should only be included once.

   eg.
   ```
   arr = [1, 2, 3, 4, 5]
   target = 5
   output = [[1, 4], [2, 3]]
   ```

   What is the time and space complexity of your solution?

Exercise Set #3 - String and Array Comparisons

1. Given two arrays, compare them to see if they are equal. Equal arrays must contain the same values in the same order.

   eg.
   ```
   arr1 = [1, 2, 3]
   arr2 = [1, 2, 3]
   output = true
   ```

   What is the time and space complexity of your solution?

2. Given two strings, determine if one string is the reverse of the other string.

   eg.
   ```
   str1 = “abcd”
   str2 = “dcba”
   output = true
   ```

   What is the time and space complexity of your solution?

3. Given two strings, determine whether they are anagrams of each other. Anagrams contain the same characters, although not necessarily in the same order.

   eg.
   ```
   str1 = “abc”
   str2 = “bac”
   output = true
   ```

   What is the time and space complexity of your solution?

Exercise Set #4 - Sliding Windows

1. Given an array, compute the sum of each length-k subarray.

   eg.
   ```
   arr = [1, 2, 3, 4, 5]
   k = 3
   output = [6, 9, 12]
   ```

   explanation: output is `[1+2+3, 2+3+4, 3+4+5]`

   What is the time and space complexity of your solution?

2. Given a string, find the length of the longest substring of the string that does not contain any repeated characters. (Full Problem Definition)

   eg.
   ```
   str = “abcdbaba”
   output = 4 (“abcd” or “cdba”)
   ```

   What is the time and space complexity of your solution?

3. Given two strings, `s` and `p`, find all occurrences of anagrams of `p` in `s`. The output should be the starting index of each anagram. (Full Problem Definition)

   eg.
   ```
   s = “abcbcba”
   p = “abc”
   output = [0, 4]
   ```

   What is the time and space complexity of your solution?

4. Given two strings, `s` and `p`, find the smallest substring of `s` that contains all the characters in `p`. (Full Problem Definition)

   eg.
   ```
   s = “aabbccdd”
   p = “abc”
   output = “abbc”
   ```
   What is the time and space complexity of your solution?