README.md

Session 06.5 - The Warmup

Since we are resuming after a considerable gap of one month, here is a set of assorted puzzlers/problems that should get your gray cells up and running.

Try to solve them without looking it up on the web. First, attempt to come up with a solution on paper (since these are common interview questions as well), and then try implementing them.

Table of Contents

Arrays

1. Rotate a 2D array by 90 degrees.

Example:
Input:

```
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

Output:

```
13 9 5 1
14 10 6 2
15 11 7 3
16 12 8 4
```

Input

```
1 2 3
4 5 6
```

Output:

```
4 1
5 2
6 3
```

2. Given an array of size N-1, with numbers in the range [1, N]. Each number occurs exactly once, with a single number missing. How do you find the missing number?

Eg:

Input:

```
1 8 4 2 3 7 5
```

Output:

```
6
```

Input:

```
1 4 2 3
```

Output:

```
5
```

3. Given an array of integers, arrange the array so that all the odd numbers appear to the left of the array, and all even numbers appear to the right.

Eg: Input

```
1 2 3 4 5 6 7 8
```

Output:

```
1 3 5 7 2 4 6 8
```

The numbers need not be in the same order. For instance, `1 7 5 3 8 4 2 6` is a valid solution for the above input.

Strings

1. Design an algorithm and write code to remove the duplicate characters in a lowercase alphabet (a-z) string without using *ANY* additional buffer. (This includes arrays of constant size)

Eg:
Input

```
abcdebcd
```

Output:

```
abcde
```

2. Given a string, find out if duplicate characters exist within k distance.

Eg: Input:

```
abcdc
2
```

Output:

```
Yes
```

Input:

```
abcdefgc
4
```

Output:

```
No
```

Explanation:
In the first case, the letter c appears within a distance of 2 (indices 2 and 4).
In the second case, the letter c appears twice, at indices 2 and 7. Since k is 4, and no other letter is duplicated, the answer is no.

Recursion

1. Print all combinations of valid pairs of parenthesis of size n

Eg:

Input:

```
2
```

Output:

```
()()
(())
```

Input:

```
3
```

Output:

```
()()()
(())()
()(())
(()())
((()))
```

2. Write a recursive function to check if a string is a palindrome or not.

3. Write a recursive function to calculate `a` raised to `b`, with a better-than-linear time complexity.