Date :

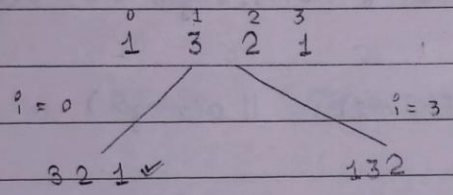
CODING QUESTIONS FOR INFOSYS

1. You are given a string "number" representing in a positive integer & a character digit. Return the resulting string after removing exactly one occurrence of digit from number such that the ~~max~~ value of the resulting string in decimal form is maximized. The test cases are generated such that digit occurs at least once in the number.

I/P : number = "1321"

digit = "1"

O/P : 321



Time Complexity → O(n) [Greedy Solution]

```
public static String removeDigitGreedy (String number, char digit){
        int n = number.length();
// If next digit is greater → remove earlier occurrence
    for(int i=0; i < n-1; i++){
        if(number.charAt(i) == digit && number.charAt(i+1) >
                                        number.charAt(i)) {
            return number.substring(0, i) + number.substring(i+1);
        }
    }

// Otherwise remove the RIGHTMOST occurrence
    for(int i = n-1; i >= 0; i--) {
        if ( number.charAt(i) == digit ) {
            return number.substring(0,i) + number.substring(i+1);
        }
    }
```

Date :

return number; // should never happen (digit always present)

}

Example → Input : number = "1231" digit = "1" Output : 1231

For i = 0 :

number.charAt(0) == 1 && number.charAt(1) > number.charAt(0)

↓ 1 ↓ 1 ↓ 1

Nothing will happen ; condition not satisfied

For i = 1 :

1 && 2 > 1

return "1" + "231" = 1231

↓ Takes part before i

↳ Takes part after i

2. You are given an integer n. A 0-indexed integer array nums of length n+1 is generated following the rules :

nums[0] = 0

nums[1] = 1

nums[2 * i] = nums[i], when 2 <= 2 * i <= n → Even index

nums[2 * i + 1] = nums[i] + nums[i+1], when 2 <= 2 * i + 1 <= n

↳ Odd index

Return the maximum integer in the array nums.

```
public static int getMaximumGenerated(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;

    int[] nums = new int[n + 1];
    nums[0] = 0;
    nums[1] = 1;

    int max = 1; // since nums[1] = 1
```

Date :

```
for ( int i = 1 ; 2*i <= n ; i++ ) {
                    → common condition
    int evenIdx = 2*i ;
    nums[evenIdx] = nums[i] ;
    if ( nums[evenIdx] > max ) {
        max = nums[evenIdx] ;
    }
```

For even indices ←

```
~~for ( int ~~ ;
    int oddIdx = 2*i + 1 ;
    nums[oddIdx] = nums[i] + nums[i+1] ;
    if ( nums[oddIdx] > max ) {
        max = nums[oddIdx] ;
    }
```

For odd indices ←

```
}
return max ;
}
```

Time Complexity → O(n)

Space Complexity → O(n)

Example : n = 7 Generated nums : [0, 1, 1, 2, 1, 3, 2, 3]

Maximum ⇒ 3

3. You are given an array ARR which has N integers. You want to construct a new array RES using ARR by following the below algorithms :

Initially, RES is empty.

Start at any index of ARR.

Choose a direction (left or right) and iterate over the elements of ARR starting from the chosen index in the chosen direction).

Add each iteration element to the end of the RES. Additionally, it is given that the ARR is cyclic. This means that after the

Date :

last element you will iterate to the first one & vice versa.
The value of RES is the sum of the bitwise XOR value of all the prefixes of it. That means the value of RES can be defined as follows:

value (RES) = RES[0] + (RES[0] ^ RES[1]) + (RES[0] ^ RES[1] ^ RES[2]) + ... + (RES[0] ^ RES[1] ^ RES[2] ... ^ RES[N-1])

Find the maximum possible value of RES.

Example:

I/P : N = 10, ARR = [7 8 5 5 9 2 2 0 16]

O/P : 99

considering RES = [5 8 7 6 1 0 2 2 9 5]

value (RES) = 5 + (5 ^ 8) + (5 ^ 8 ^ 7) + ...

value (RES) = 5 + 13 + 10 + 12 + 13 + 13 + 13 + 13 + 1 + 1

Allowed To:

Start from ANY index

Move in ANY direction (left / right)

Array is CYCLIC

RES length = N (every element used exactly once)

So, every possible RES is simply a rotation (and direction change) of ARR.

Meaning:

You only need to check all N rotations (both directions).

For each rotation, compute the prefix XORs & add them.

Time Complexity → $O(N^2)$ but with micro-optimizations

Space Complexity → $O(N)$

Date :

```
// Function to compute sum of prefix XORs for one given array
private static int valueOfRES (int[] arr) {
    int prefixXor = 0;
    int sum = 0;
    for (int x : arr) {
        prefixXor ^= x;
        sum += prefixXor;
    }
    return sum;
}

// Main Optimized Function
public static int getMaxValue (int[] arr) {
    int n = arr.length;
// Create doubled size array to simulate cyclic rotations
int[] doubled = new int[2 * n];
for(int i = 0; i < n; i++) {
    doubled[i] = arr[i];
    doubled[i + n] = arr[i];
}

int max = 0;

// Check all rotations (left direction)
for (int st = 0; st < n; st++) {
    int[] temp = new int[n];
    for (int i = 0; i < n; i++) {
        temp[i] = doubled[start + i];
    }
    max = Max.max(max, valueOfRES(temp));
}
```

Date :

```
// Reversed array for right direction
int[] reversed = new int[n];
for (int i = 0; i < n; i++) {
    reversed[i] = arr[n - i - 1];
}

// Create doubled array for reversed
int[] doubledRev = new int[2 * n];
for (int i = 0; i < n; i++) {
    doubledRev[i] = reversed[i];
    doubledRev[i + n] = reversed[i];
}

// Check all rotations (right direction)
for (int st = 0; st < n; st++) {
    int[] temp = new int[n];
for (int i = 0; i < n; i++) {
    temp[i] = doubledRev[st + i];
}
max = Math.max(max, valueOfRES(temp));
}
return max;
}
```

Example : n = 5

arr = [1, 2, 3, 4, 5]

Maximum ⇒ 15

Check all cyclic rotations in both directions, compute prefix XORs for each RES, and return the maximum sum.