



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Assignment 4

Student Name: Kumar Devashish

UID :22BCS10248

Branch : BE-CSE

Sect./Grp :FL_IOT-602-A

Semester : 6th

Date of Performance :21/02/25

Subject Name : AP Lab

Subject Code :22CSP-351

1. Aim: 1763. Longest Nice Substring

```
class Solution {  
    public String longestNiceSubstring(String s) {  
        if (s.length() < 2) return "";  
  
        for (int i = 0; i < s.length(); i++) {  
            char ch = s.charAt(i);  
            if (s.indexOf(Character.toUpperCase(ch)) == -1 ||  
                s.indexOf(Character.toLowerCase(ch)) == -1) {  
  
                String left = longestNiceSubstring(s.substring(0, i));  
                String right = longestNiceSubstring(s.substring(i + 1));  
                return left.length() >= right.length() ? left : right;  
            }  
        }  
  
        return s;  
    }  
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Output:

The screenshot shows a coding interface for the problem "1763. Longest Nice Substring". The problem description on the left explains that a string is "nice" if every letter it contains appears both in uppercase and lowercase. It provides examples: "aAaBb" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "aBA" is not nice because 'b' appears but 'B' does not. The task is to return the longest substring of *s* that is nice, or an empty string if none exist.

Example 1:
Input: *s* = "YazAay"
Output: "aAa"
Explanation: "aAa" is a nice string because 'A/a' is the only letter of the alphabet in *s*, and both 'A' and 'a' appear. "aAa" is the longest nice substring.

Example 2:
Input: *s* = "Bb"
Output: "Bb"
Explanation: "Bb" is a nice string because both 'B' and 'b' appear. The whole string is a substring.

Example 3:
Input: *s* = "c"
Output: ""
Explanation: There are no nice substrings.

Constraints:
• $1 \leq s.length \leq 100$
• *s* consists of uppercase and lowercase English letters.

The code editor on the right shows a Java solution:

```
class Solution {
    public String longestNiceSubstring(String s) {
        if (s.length() < 2) return "";

        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (s.indexOf(Character.toUpperCase(ch)) == -1 || s.indexOf(Character.toLowerCase(ch)) == -1) {
                String left = longestNiceSubstring(s.substring(0, i));
                String right = longestNiceSubstring(s.substring(i + 1));
                return left.length() > right.length() ? left : right;
            }
        }

        return s;
    }
}
```

The test result at the bottom shows "Accepted" with a runtime of 0 ms. The input for Case 1 is *s* = "YazAay".

2. Aim: 190. Reverse Bits

```
public class Solution {

    public int reverseBits(int n) {

        int result = 0;

        for (int i = 0; i < 32; i++) {

            result = (result << 1) | (n & 1);

            n >>= 1;

        }

        return result;

    }

}
```



Output:

190. Reverse Bits

Reverse bits of a given 32 bits unsigned integer.

Note:

- Note that in some languages, such as Java, there is no unsigned integer type. In this case, both input and output will be given as a signed integer type. They should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using **2's complement notation**. Therefore, in **Example 2** above, the input represents the signed integer **-3** and the output represents the signed integer **-1873741823**.

Example 1:

Input: n = 0000001010010100000111010011100
Output: 964176192 (0011100101110000010100101000000)
Explanation: The input binary string 0000001010010100000111010011100 represents the unsigned integer 43261596, so return 964176192 which its binary representation is 0011100101110000010100101000000.

Example 2:

Input: n = 11111111111111111111111111111101
Output: 3221225471 (1011111111111111111111111111111)
Explanation: The input binary string 11111111111111111111111111111101 represents the unsigned integer 4294967295, so return 3221225471 which its binary representation is 1011111111111111111111111111111.

Constraints:

- The input must be a binary string of length 32

```
public class Solution {
    public int reverseBits(int n) {
        int result = 0;
        for (int i = 0; i < 32; i++) {
            result = (result << 1) | (n & 1);
            n >>= 1;
        }
        return result;
    }
}
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

n = 0000001010010100000111010011100

Output

3. **Aim:** 191. Number of 1 Bits

```
class Solution {

    public int hammingWeight(int n) {

        int count = 0;

        while (n != 0) {

            count += (n & 1);

            n >>= 1;

        }

        return count;

    }

}
```



Output:

The screenshot shows the LeetCode interface for problem 191, "Number of 1 Bits". The problem description asks for a function to return the number of set bits in the binary representation of a positive integer n. The solution is implemented in Java as follows:

```
1 class Solution {
2     public int hammingWeight(int n) {
3         int count = 0;
4         while (n != 0) {
5             count += (n & 1);
6             n >>= 1;
7         }
8         return count;
9     }
10 }
11
```

The test results show that the solution is "Accepted" with a runtime of 0 ms. The input is n = 11, and the output is 3.

4. Aim: 53. Maximum Subarray

```
class Solution {

    public int maxSubArray(int[] nums) {

        int maxSum = nums[0], currentSum = nums[0];

        for (int i = 1; i < nums.length; i++) {

            currentSum = Math.max(nums[i], currentSum + nums[i]);

            maxSum = Math.max(maxSum, currentSum);

        }

        return maxSum;

    }

}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Output:

53. Maximum Subarray Solved

Given an integer array `nums`, find the **subarray** with the largest sum, and return its sum.

Example 1:
Input: `nums = [-2,1,-3,4,-1,2,1,-5,4]`
Output: 6
Explanation: The subarray `[4,-1,2,1]` has the largest sum 6.

Example 2:
Input: `nums = [1]`
Output: 1
Explanation: The subarray `[1]` has the largest sum 1.

Example 3:
Input: `nums = [5,4,-1,7,8]`
Output: 23
Explanation: The subarray `[5,4,-1,7,8]` has the largest sum 23.

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^4 \leq \text{nums}[i] \leq 10^4$

Follow up: If you have figured out the $O(n)$ solution, try coding another solution using the **divide and conquer** approach, which is more subtle.

```
class Solution {  
    public int maxSubArray(int[] nums) {  
        int maxSum = nums[0], currentSum = nums[0];  
        for (int i = 1; i < nums.length; i++) {  
            currentSum = Math.max(nums[i], currentSum + nums[i]);  
            maxSum = Math.max(maxSum, currentSum);  
        }  
        return maxSum;  
    }  
}
```

Testcase 1: `[-2,1,-3,4,-1,2,1,-5,4]`

5. Aim: 240. Search a 2D Matrix II

Implementation/ Code:

```
class Solution {  
public:  
    bool searchMatrix(vector<vector<int>>& matrix, int target) {  
        int row = 0, col = matrix[0].size() - 1;  
        while (row < matrix.size() && col >= 0) {  
            if (matrix[row][col] == target) return true;  
            matrix[row][col] > target ? col-- : row++;  
        }  
        return false;  
    }  
};
```

Output:



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

240. Search a 2D Matrix II Solved

Medium Topics Companies

Write an efficient algorithm that searches for a value `target` in an $m \times n$ integer matrix `matrix`. This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

Example 1:

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

Input: `matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]`, `target = 5`
Output: `true`

Example 2:

1	4	7	11	15
---	---	---	----	----

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        for (int[] row : matrix) {
            if (binarySearch(row, target)) {
                return true;
            }
        }
        return false;
    }

    private boolean binarySearch(int[] row, int target) {
        int left = 0, right = row.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (row[mid] == target) {
                return true;
            } else if (row[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return false;
    }
}
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

matrix =

[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]

target =

5

6. Aim: 372. Super Pow

```
class Solution {
```

```
    private static final int MOD = 1337;
```

```
    public int superPow(int a, int[] b) {
```

```
        a %= MOD;
```

```
        int result = 1;
```

```
        for (int digit : b) {
```

```
            result = power(result, 10) * power(a, digit) % MOD;
```

```
        }
```

```
        return result;
```

```
    }
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
private int power(int base, int exp) {  
  
    int res = 1;  
  
    while (exp > 0) {  
  
        if ((exp & 1) == 1) res = res * base % MOD;  
  
        base = base * base % MOD;  
  
        exp >>= 1;  
  
    }  
  
    return res;  
  
}  
  
}
```

Output:

The screenshot displays a coding interface for the problem "372. Super Pow". The left panel shows the problem description, examples, and constraints. The right panel shows the Java code solution and the test results.

Problem Description: 372. Super Pow. Medium. Your task is to calculate $a^b \text{ mod } 1337$ where a is a positive integer and b is an extremely large positive integer given in the form of an array.

Example 1: Input: $a = 2, b = [3]$ Output: 8

Example 2: Input: $a = 2, b = [1,0]$ Output: 1024

Example 3: Input: $a = 1, b = [4,3,3,8,5,2]$ Output: 1

Constraints:

- $1 \leq a \leq 2^{31} - 1$
- $1 \leq b.length \leq 2000$
- $0 \leq b[i] \leq 9$
- b does not contain leading zeros.

Testcase 1: Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

$a =$
2

$b =$
[3]