1. Add Binary

Difficulty: Easy
Time: 15 mins

Problem Statement:

Given two binary strings `a` and `b`, return their sum as a binary string.

Sample Input:

```
a = "1010"
b = "1011"
```

Sample Output:

```
"10101"
```

```python
def addBinary(a: str, b: str) -> str:
    # Initialize result string and carry
    result = []
    carry = 0

    # Pointers for both strings
    i, j = len(a) - 1, len(b) - 1

    while i >= 0 or j >= 0 or carry:
        # Get current bits, if available, otherwise 0
        bit_a = int(a[i]) if i >= 0 else 0
        bit_b = int(b[j]) if j >= 0 else 0

        # Sum bits and carry
```

```python
        total = bit_a + bit_b + carry

        # Compute new carry
        carry = total // 2

        # Append result of current bit
        result.append(str(total % 2))

        # Move to the next bits
        i -= 1
        j -= 1

    # The result list holds the bits in reverse order
    return ''.join(reversed(result))

# Sample test case
print(addBinary("1010", "1011"))  # Output: "10101"
```

```
10101
```

2. Find the Duplicate Number

Difficulty: Medium
Time: 20 mins

Problem Statement:

Given an array of integers `nums` containing `n + 1` integers where each integer is between 1 and `n` (inclusive), find the duplicate number.

Sample Input:

```
nums = [1, 3, 4, 2, 2]
```

Sample Output:

```
2
```

```python
def findDuplicate(nums):
    # Using Floyd's Tortoise and Hare algorithm
    # Phase 1: Finding the intersection point of two runners
    slow = nums[0]
    fast = nums[0]

    # Move slow by 1 step and fast by 2 steps
    while True:
        slow = nums[slow]
        fast = nums[nums[fast]]
        if slow == fast:
            break

    # Phase 2: Finding the entrance to the cycle
    slow = nums[0]
    while slow != fast:
        slow = nums[slow]
        fast = nums[fast]

    return slow

# Sample test case
print(findDuplicate([1,3,4,2,2]))  # Output: 2
```

```
2
```

3. Counting Bits

Difficulty: Easy
Time: 15 mins

Problem Statement:

Given an integer `n`, return an array of the number of 1-bits in the binary representation of all numbers in the range `[0, n]`.

Sample Input:

```
n = 5
```

Sample Output:

```
[0, 1, 1, 2, 1, 2]
```

```python
def countBits(n: int):
    dp = [0] * (n + 1)  # Initialize DP array

    for i in range(1, n + 1):
        # i >> 1 is i // 2, thus inheriting the count of bits from previous power of two
        dp[i] = dp[i >> 1] + (i & 1)

    return dp

# Sample test case
print(countBits(5))  # Output: [0, 1, 1, 2, 1, 2]
```

```
[0, 1, 1, 2, 1, 2]
```

4. Number of 1 Bits

Difficulty: Easy
Time: 15 mins

Problem Statement:

Write a function that takes an integer `n` and returns the number of `1` bits it has.

Sample Input:

```
n = 00000000000000000000000000001011
```

Sample Output:

```
3
```

```python
def hammingWeight(n: int) -> int:
    count = 0
    while n:
        # Increment count if the last bit is 1
        count += n & 1
        # Right shift n to check the next bit
        n >>= 1
    return count

# Sample test case
print(hammingWeight(11))  # Output: 3 (binary of 11 is 1011)
```

```
3
```

5. Single Number

Difficulty: Easy
Time: 15 mins

Problem Statement:

Given a non-empty array of integers, every element appears twice except for one. Find that single one.

Sample Input:

```
nums = [2, 2, 1]
```

Sample Output:

```
1
```

```python
def singleNumber(nums) -> int:
    result = 0

    for num in nums:
        # XOR operation will cancel out same numbers
        result ^= num
```

```
    return result

# Sample test case
print(singleNumber([4,1,2,1,2]))  # Output: 4
```

```
4
```

6. Missing Number

Difficulty: Easy
Time: 15 mins

Problem Statement:

Given an array containing `n` distinct numbers taken from the range `[0, n]`, find the one that is missing from the array.

Sample Input:

```
nums = [3, 0, 1]
```

Sample Output:

```
2
```

```
def missingNumber(nums) -> int:
    n = len(nums)
    # Sum of first n natural numbers
    total = n * (n + 1) // 2
    # Subtract the sum of the array from the total to find the missing number
    return total - sum(nums)

# Sample test case
print(missingNumber([3, 0, 1]))  # Output: 2
```

```
2
```

7. Reverse Bits

Difficulty: Easy
Time: 15 mins

Problem Statement:

Reverse bits of a given 32 bits unsigned integer.

Sample Input:

```
n = 43261596
```

Sample Output:

```
964176192
```

```python
def reverseBits(n: int) -> int:
    result = 0
    for i in range(32):
        # Shift result to make room for the next bit
        result <<= 1
        # Append the last bit of n to result
        result |= n & 1
        # Shift n to the right to process the next bit
        n >>= 1
    return result

# Sample test case
print(reverseBits(43261596))  # Output: 964176192 (binary representation: 1010010100000111101
```

```
964176192
```