BIT MANIPULATION

No.	Problem Statement	Solution	Time complexity	Space complexity
1	**Single Number**			
	Given a an array of integers 'nums', every element appears twice except for one. Find that single one.	- Idea: **XOR Properties** - n ^ 0 = n - n ^ n = 0 `for(auto x: nums) ans = ans^x`	O(N)	O(1)
2	**Number of 1 Bits**			

	Given an unsigned int return number of '1' bits	Note: 1 = 00000000 00000000 00000000 00000001
		n = 00000000 00000000 00000000 00001011
		`while(n>0)`
		`    if(n&1==1) cnt++`
		`    n=n>>1` --> **right shift**
			O(1)	O(1)

3	**Counting Bits**			
	Given an integer n, return an array ans of length n + 1 such that for each i (0 <= i <= n), ans[i] is the number of 1's in the binary representation of i.	Approach_1: - Loop through 1 to n, converting each to binary notation and counting the no. of 1s	O(N*logN)	O(1)
		Approach_2: - x is **even** -> number of 1's in x == number of 1's in **x/2** - x is **odd** -> number of 1's in x == number of 1's in **x/2 + 1**	O(N)	O(1)
4	**Reverse Bits**			

	Reverse bits of a given 32 bits unsigned integer.
	Ex. n = 10011100 -> 00111001 = 57
		Idea: Push bits **out of 'n' (right shift)** and push bits **into 'ans' (left shift)**
		- Initialize `uint32_t ans=0`
		`for(int i=0; i<32; i++)` // Note: while(n>0) will not work
		`    ans = ans<<1;`
		`    ans = ans ^ (n&1);` // (n&1) gives the last bit of 'n'
		// ans | (n&1) sets the last bit of 'ans'
		`    n = n>>1;`
			O(1)	O(1)

5	**Missing Number**			
	Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array. Ex. nums=[1,2] --> output = 0	Idea: Bitwise XOR to find the missing number: **(a^a = 0) (a^0=a)** 1) Initialize `ans = 0` 2) Iterate over 'nums' to cancel out elements already present in 'nums'. : `ans = ans ^ i ^ nums[i]` 3) Check for 'n' as the loop will iterate through 0 --> n-1: `ans = ans^n`	O(N)	O(1)
6	**Sum of Two Integers**			
	Given two integers a and b, return the sum of the two integers without using the operators + and -.	Idea: Bitwise **XOR** for **addition** , Bitwise **AND** for **carry** `while(b!=0)` `    carry = a & b;` --> Sets 'carry' with bits that are set in both a and b Indicating the positions producing a carry `    a = a ^ b;` --> Performs addition operation without considering carry `    b = carry << 1;` --> 'carry' is shifted one position to the left to calculate the carry for the 'next iteration'	O(N)	O(1)
7	**Reverse Integer**			

BIT MANIPULATION

No.	Problem Statement	Solution	Time complexity	Space complexity

	Given a signed 32-bit integer x, return x with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0.
	Ex. x = 123 -> 321, x = -123 -> -321, x = 120 -> 21
		- INT_MAX: (2,147,483,647) 2^31
		- INT_MIN: (-2,147,483,648) -2^31
		`while(x!=0)`
		`b = x%10;`
		`if(ans>INT_MAX/10 || ans==INT_MAX/10 && b>7) return 0;`
		`if(ans<INT_MIN/10 || ans==INT_MIN/10 && b<-8) return 0;` // IMP: b < -8
		`ans = ans*10 + b;`
		`x = x/10;`
			log10(X)	O(1)

		Extra knowledge: In c++: 'int' is 'signed int' For Signed int INT_MAX: 01111111 11111111 11111111 11111111 (2,147,483,647) INT_MIN: 10000000 00000000 00000000 00000000 (-2,147,483,648) For Unsigned int INT_MAX: 11111111 11111111 11111111 11111111 (4,294,967,295) INT_MIN: 00000000 00000000 00000000 00000000 (0) When we perform a left shift operation on 'signed INT_MIN' i.e '-2147483648 << 1' the leftmost bit (the sign bit) is shifted out of the range of the 32-bit integer, causing an overflow.		