**Two Pointer (Easy)**

**Move Zeroes (283)**

java

**public** **void** moveZeroes(**int**[] nums) {

**int** j = 0;

**for** (**int** i = 0; i < nums.length; i++) {

**if** (nums[i] != 0) {

**int** temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

j++;

}

}

}

**Remove Element (27)**

java

**public** **int** removeElement(**int**[] nums, **int** val) {

**int** j = 0;

**for** (**int** i = 0; i < nums.length; i++) {

**if** (nums[i] != val) {

nums[j] = nums[i];

j++;

}

}

**return** j;

}

**Remove Duplicates from Sorted Array (26)**

java

**public** **int** removeDuplicates(**int**[] nums) {

**if** (nums.length == 0) **return** 0;

**int** j = 1;

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

**if** (nums[i] != nums[i - 1]) {

nums[j] = nums[i];

j++;

}

}

**return** j;

}

**Reverse String (344)**

java

**public** **void** reverseString(**char**[] s) {

**int** left = 0, right = s.length - 1;

**while** (left < right) {

**char** temp = s[left];

s[left] = s[right];

s[right] = temp;

left++;

right--;

}

}

**Valid Palindrome (125)**

java

**public** **boolean** isPalindrome(String s) {

s = s.replaceAll("[^A-Za-z0-9]", "").toLowerCase();

**int** left = 0, right = s.length() - 1;

**while** (left < right) {

**if** (s.charAt(left) != s.charAt(right)) **return** false;

left++;

right--;

}

**return** true;

}

**Bit Manipulation (Easy)**

**Single Number (136)**

java

**public** **int** singleNumber(**int**[] nums) {

**int** res = 0;

**for** (**int** num : nums) res ^= num;

**return** res;

}

**Reverse Bits (190)**

java

**public** **int** reverseBits(**int** n) {

**int** res = 0;

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

res <<= 1;

res |= (n & 1);

n >>= 1;

}

**return** res;

}

**Sum of Two Integers (371)**

java

**public** **int** getSum(**int** a, **int** b) {

**while** (b != 0) {

**int** carry = (a & b) << 1;

a = a ^ b;

b = carry;

}

**return** a;

}

**Missing Number (268)**

java

**public** **int** missingNumber(**int**[] nums) {

**int** res = nums.length;

**for** (**int** i = 0; i < nums.length; i++)

res ^= i ^ nums[i];

**return** res;

}

**Number Complement (476)**

java

**public** **int** findComplement(**int** num) {

**int** mask = 1;

**while** (mask <= num) mask <<= 1;

**return** (mask - 1) ^ num;

}

**Dynamic Programming (Easy)**

**Climbing Stairs (70)**

java

**public** **int** climbStairs(**int** n) {

**if** (n <= 2) **return** n;

**int** a = 1, b = 2;

**for** (**int** i = 3; i <= n; i++) {

**int** temp = b;

b = a + b;

a = temp;

}

**return** b;

}

**Min Cost Climbing Stairs (746)**

java

**public** **int** minCostClimbingStairs(**int**[] cost) {

**for** (**int** i = 2; i < cost.length; i++) {

cost[i] += Math.min(cost[i - 1], cost[i - 2]);

}

**return** Math.min(cost[cost.length - 1], cost[cost.length - 2]);

}

**House Robber (198)**

java

**public** **int** rob(**int**[] nums) {

**int** prev = 0, curr = 0;

**for** (**int** num : nums) {

**int** temp = curr;

curr = Math.max(curr, prev + num);

prev = temp;

}

**return** curr;

}

**Fibonacci Number (509)**

java

**public** **int** fib(**int** n) {

**if** (n < 2) **return** n;

**int** a = 0, b = 1;

**for** (**int** i = 2; i <= n; i++) {

**int** temp = b;

b = a + b;

a = temp;

}

**return** b;

}

**Range Sum Query - Immutable (303)**

java

**class** NumArray {

**int**[] sums;

**public** NumArray(**int**[] nums) {

**this**.sums = **new** **int**[nums.length + 1];

**for** (**int** i = 0; i < nums.length; i++) {

sums[i + 1] = sums[i] + nums[i];

}

}

**public** **int** sumRange(**int** left, **int** right) {

**return** sums[right + 1] - sums[left];

}

}