#344 Reverse String:

Keep track of the leftmost character and the end character. Swap until the leftmost index is less than the end index.

Time Complexity: O(n)

public void reverseString(char[] s) {

int left = 0;

int end = s.length-1;

while (left < end) {

//swap

char temp = s[left];

s[left] = s[end];

s[end] = temp;

left ++;

end --;

}

}

#412 FizzBuzz

Create a list of strings. Loop through from 1 to n. If the number is divisible by 15, add “FizzBuzz”. If divisible by 3, add “Fizz”. If divisible by 5, add “Buzz.”

public class Solution {

public List<String> fizzBuzz(int n) {

List<String> list = new ArrayList<>();

for (int i = 1; i <= n; i++) {

if (i % 3 == 0 && i % 5 == 0) {

list.add("FizzBuzz");

} else if (i % 3 == 0) {

list.add("Fizz");

} else if (i % 5 == 0) {

list.add("Buzz");

} else {

list.add(String.valueOf(i));

}

}

return list;

}

}

#136 Single Number

Loops through all the values in the array and uses XOR to determine if the number has occurred or not

Time Complexity: O(n)

class Solution {

public int singleNumber(int[] nums) {

int res = 0;

for (int num : nums){

res ^= num;

}

return res;

}

}

#104 Maximum depth of a binary tree

Check if the root is nullptr and determine the max of the depth of the left child and the depth of the right child.

Time Complexity: O(n)

class Solution {

public int maxDepth(TreeNode root) {

if (root == null) {

return 0;

}

int leftDepth = maxDepth(root.left);

int rightDepth = maxDepth(root.right);

return Math.max(leftDepth, rightDepth) + 1;

}

}

#283 Move Zeros

Move all the nonzero values over in the array, and then replace the last values from the nonzero number to the end with 0’s.

class Solution {

public void moveZeroes(int[] nums) {

int count = 0;

int n = nums.length;

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

if(nums[i] != 0){

nums[count++] = nums[i];

}

}

for (int j = count; j < n; j++){

nums[j] = 0;

}

}

}

#371 Sum of Two Integers

If the numbers are the same, use and to add them together. If the numbers are different, use xor to add the two numbers together

public class Solution {

public int getSum(int a, int b) {

while (a != 0 && b != 0) {

int and = a & b;

int xor = a ^ b;

a = and << 1;

b = xor;

}

if (a!= 0) {

return a;

}

else {

return b;

}

}

}

#206 Reverse Linked List

public class Solution {

public ListNode reverseList(ListNode head) {

ListNode fakeHead = new ListNode(-1);

reverse(head, fakeHead);

return fakeHead.next;

}

//reverse returns the tail of the reversed list

private ListNode reverse(ListNode node, ListNode fakeHead){

if (node == null)

return fakeHead;

else{

ListNode tail = reverse(node.next, fakeHead);

tail.next = node;

node.next = null;

return node;

}

}

}

#171 Excel Sheet Column Number

For each character in the string convert into base 26 through character manipulation

public class Solution {

public int titleToNumber(String s) {

int result = 0;

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

int digit = s.charAt(i) - 'A' + 1;

result = result \* 26 + digit;

}

return result;

}

}

#169 Majority Element

Create Variables that will store a counter and the majority variable. If the value in the majority variable occurs, increment the counter. If not, decrement the counter. If the counter is 0, update the majority variable

public class Solution {

public int majorityElement(int[] num) {

int candidate = num[0];

int counter = 1;

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

if (counter == 0){

candidate = num[i];

counter = 1;

}

else{

if (num[i] == candidate)

counter ++;

else

counter --;

}

}//for i

return candidate;

}

}

#13 Roman to Integer

Create a HashMap to store all the romanNumeral Values. If the curr roman numeral is greater less than the previous one the previous one, correct calculation. Otherwise, add value to the counter.

public class Solution {

public int romanToInt(String s) {

HashMap<Character, Integer> helper = new HashMap<Character, Integer>();

initHelper(helper);

int result = 0;

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

if ( i > 0 && helper.get(s.charAt(i)) > helper.get(s.charAt(i-1))) {

result = result - 2 \* helper.get(s.charAt(i-1));

}

result = result + helper.get(s.charAt(i));

}

return result;

}

private void initHelper(HashMap<Character, Integer> helper) {

helper.put('M', 1000);

helper.put('D', 500);

helper.put('C', 100);

helper.put('L', 50);

helper.put('X', 10);

helper.put('V', 5);

helper.put('I', 1);

}

}

#237 Delete Node in a Linked List

public class Solution {

public void deleteNode(ListNode node) {

node.val = node.next.val;

node.next = node.next.next;

}

}

#121 [Best Time to Buy and Sell Stock](https://leetcode.com/problems/best-time-to-buy-and-sell-stock)

Loop through the array and determine the profit by subtracting the previous price from the current price. If the difference is negative, replace with 0.

public class Solution {

public int maxProfit(int[] prices) {

int p = 0;

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

p += Math.max(0, prices[i] - prices[i-1]);

}

return p;

}

}

#242 [Valid Anagram](https://leetcode.com/problems/valid-anagram)

Create Two HashMaps. Store the frequency of each character. Loop through the string to make sure that the values are the same for both HashMaps.

public class Solution {

public boolean isAnagram(String s, String t) {

HashMap&lt;Character, Integer&gt; as = new HashMap&lt;Character, Integer&gt;();

HashMap&lt;Character, Integer&gt; at = new HashMap&lt;Character, Integer&gt;();

count(as, s);

count(at, t);

if (as.size() != at.size()) {

return false;

}

for (Character c:as.keySet()) {

if (!at.containsKey(c) || !as.get(c).equals(at.get(c))) {

return false;

}

}

return true;

}

private void count(HashMap&lt;Character, Integer&gt; appear, String s) {

for (int i = 0; i &lt; s.length(); i ++) {

char c = s.charAt(i);

if (!appear.containsKey(c)) {

appear.put(c, 0);

}

appear.put(c, appear.get(c) + 1);

}

}

}

#217 [Contains Duplicate](https://leetcode.com/problems/contains-duplicate)

Create a HashSet to store the non duplicate values in the list. If there is a duplicate return true.

public class Solution {

public boolean containsDuplicate(int[] nums) {

HashSet<Integer> set = new HashSet<Integer>();

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

if (set.contains(nums[i])){

return true;

}

set.add(nums[i]);

}

return false;

}

}

#387 [First Unique Character in a String](https://leetcode.com/problems/first-unique-character-in-a-string)

Store the frequency of each character in an array. Loop through string to determine if the frequency of the character is 1.

class Solution {

public int firstUniqChar(String s) {

if (s == null || s.length() == 0) {

return -1;

}

int[] counts = new int[26];

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

char c = s.charAt(i);

counts[c - 'a']++;

}

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

char c = s.charAt(i);

if (counts[c - 'a'] == 1) {

return i;

}

}

return -1;

}

}

#108 [Convert Sorted Array to Binary Search Tree](https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree)

Find the middle of the array, and split the two parts into the left portion of the tree and the right portion of the tree.

public class Solution {

public TreeNode sortedArrayToBST(int[] nums) {

if (nums.length == 0){

return null;

}

return buildBST(nums, 0, nums.length - 1);

}

private TreeNode buildBST(int[] nums, int start, int end){

if (start > end)

return null;

else{

int mid = ( start + end ) / 2;

TreeNode node = new TreeNode(nums[mid]);

node.left = buildBST(nums, start, mid - 1);

node.right = buildBST(nums, mid + 1, end);

return node;

}

}

}

#268 [Missing Number](https://leetcode.com/problems/missing-number)

Determine the expected sum (1 + 2 + … + n). Find the sum of the numbers in the array. The missing number is the expected sum - the sum calculated in the array.

public class Solution {

public int missingNumber(int[] nums) {

int sum = 0;

int expected = (1 + nums.length) \* nums.length / 2;

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

sum += nums[i];

}

return expected - sum;

}

}

#350 [Intersection of Two Arrays II](https://leetcode.com/problems/intersection-of-two-arrays-ii)

Create a HashMap to store the frequency of each number in the first array. Loop through the second array and determine if the value exists in the HashMap. If it does, store values in the new list and update the values in the HashMap. Return an array of all the values.

public class Solution {

public int[] intersect(int[] nums1, int[] nums2) {

HashMap<Integer, Integer> data = new HashMap<Integer, Integer>();

for (int i = 0; i < nums1.length; i ++){

if (!data.containsKey(nums1[i])) {

data.put(nums1[i], 1);

}

else {

data.put(nums1[i], data.get(nums1[i]) + 1);

}

}

List<Integer> res = new LinkedList<Integer>();

for (int i = 0; i < nums2.length; i ++) {

if (data.containsKey(nums2[i]) && data.get(nums2[i]) != 0) {

res.add(nums2[i]);

data.put(nums2[i], data.get(nums2[i]) - 1);

}

}

int[] result = new int[res.size()];

int i = 0;

for (Integer j:res) {

result[i++] = j;

}

return result;

}

}

#121 Best Time to Buy and Sell Stock

Determine the minimum price and the max price and subtract to determine the maximum profit.

public class Solution {

public int maxProfit(int[] prices) {

if (prices == null || prices.length == 0) {

return 0;

}

int minPrice = prices[0];

int max = 0;

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

max = Math.max(max, prices[i] - minPrice);

minPrice = Math.min(minPrice, prices[i]);

}

return max;

}

}

#21 [Merge Two Sorted Lists](https://leetcode.com/problems/merge-two-sorted-lists)

Determine the smaller value and add the value to the temp list. Keep doing this until one list hits nullptr. Update the rest of the list. Return the temp list.

public class Solution {

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

ListNode fakeNode = new ListNode(-1);

ListNode node = fakeNode;

while (l1 != null && l2 != null){

if (l1.val < l2.val){

node.next = l1;

l1 = l1.next;

node = node.next;

}//if

else{

node.next = l2;

l2 = l2.next;

node = node.next;

}//else

}//while l1 & l2

if (l1 != null){

node.next = l1;

}

else if (l2 != null){

node.next = l2;

}

return fakeNode.next;

}

}

#202 Happy Number

Create HashSet. The Happy Number eventually repeats either it being 1 or the original number n. If one returns true after repeatedly calculating the squares of the digits, otherwise return false.

class Solution {

public boolean isHappy(int n) {

HashSet<Integer> hashSet = new LinkedHashSet<>();// the hash table records each digit in the square sum calculation while (!hashSet.contains(n)) {

hashSet.add(n);

int sum = 0;

while (n > 0) {// calculate the sum of squares

sum += (n % 10) \* (n % 10);

n /= 10;

}

n = sum;//n for

}

return n == 1;

}

}

#118 Pascal’s Triangle

#70 Climbing Stairs

#101 Symmetric Tree

#53 Maximum Subarray

#326 Power of Three

#191 Number of 1 Bits

#198 House Robber

#66 Plus One

#1 Two Sum

#38 Count and Say

#26 Remove Duplicates from Sorted Array

#172 Factorial Trailing Zeroes

#20 Valid Parentheses

#141 Linked List Cycle

#234 Palindrome Linked List