

Cracking The Leetcode in C++

Shuopeng Zhang

Published
with GitBook



Table of Contents

Introduction	0
Linked List	1
Add Two Numbers(2)	1.1
Remove Nth Node From End of List(19)	1.2
Merge Two Sorted Lists(21)	1.3
Merge k sorted lists(23)	1.4
Swap Nodes in Pairs(24)	1.5
Remove Duplicates from Sorted List II(82)	1.6
Remove Duplicates from Sorted List(83)	1.7
Partition List(86)	1.8
Convert Sorted List to Binary Search Tree(109)	1.9
Copy List with Random Pointer(138)	1.10
Linked List Cycle(141)	1.11
Insertion Sort List(147)	1.12
Intersection of Two Linked Lists(160)	1.13
Remove Linked List Elements(203)	1.14
Reverse Linked List(206)	1.15
Palindrome Linked List(234)	1.16
Odd Even Linked List(328)	1.17
Graph	2
Clone Graph(133)	2.1
Strings	3
Integer to RomanLongest Substring Without Repeating Characters(3)	
Longest Palindromic Substring(5)	3.2 3.1
Integer to Roman (12)	3.3
Roman to Integer(13)	3.4
Longest Common Prefix(14)	3.5

Letter Combinations of a Phone Number(17)	3.6
Valid Parentheses(20)	3.7
Generate Parentheses(22)	3.8
Implement strStr()(28)	3.9
Multiply Strings(43)	3.10
Group Anagrams(49)	3.11
Length of Last Word(58)	3.12
Add Binary(67)	3.13
Edit Distance(72)	3.14
Decode Ways(91)	3.15
Restore IP Addresses(93)	3.16
Valid Palindrome(125)	3.17
Reverse Words in a String(151)	3.18
Compare Version Numbers(165)	3.19
Verify Preorder Serialization of a Binary Tree(331)	3.20
Maximum Product of Word Lengths(318)	3.21
Basic Calculator(224)	3.22
Palindrome Pairs(336)	3.23
Array	4
Two Sum(1)	4.1
3Sum(15)	4.2
Remove Duplicates from Sorted Array(26)	4.3
Remove Element(27)	4.4
Next Permutation(31)	4.5
Search for a Range(34)	4.6
Search Insert Position(35)	4.7
Combination Sum(39)	4.8
Combination Sum II(40)	4.9
Rotate Image(48)	4.10
Maximum Subarray(53)	4.11

Spiral Matrix(54)	4.12
Jump Game(55)	4.13
Merge Intervals(56)	4.14
Unique Paths(62)	4.15
Unique Paths II(63)	4.16
Minimum Path Sum(64)	4.17
Plus One(66)	4.18
Set Matrix Zeroes(73)	4.19
Search a 2D Matrix(74)	4.20
Sort Colors(75)-----	4.21
Subsets(78)	4.22
Word Search(79)	4.23
Remove Duplicates from Sorted Array II(80)	4.24
Merge Sorted Array(88)	4.25
Subsets II(90)	4.26
Construct Binary Tree from Preorder and Inorder Traversal(105)	4.27
Construct Binary Tree from Inorder and Postorder Traversal(106)	4.28
Pascal's Triangle(118)	4.29
Pascal's Triangle II(119)	4.30
Maximum Product Subarray(152)	4.31
Find Minimum in Rotated Sorted Array(153)	4.32
Find Minimum in Rotated Sorted Array II(154)	4.33
Majority Element(169)	4.34
Combination Sum III(216)	4.35
Contains Duplicate(217)	4.36
Contains Duplicate II(219)	4.37
Median of Two Sorted Arrays(4)	4.38
Summary Ranges(228)	4.39
Find Peak Element(162)	4.40
Longest Increasing Path in a Matrix(329)	4.41

Game of Life(289)	4.42
H-Index(274)	4.43
Search a 2D Matrix II(240)	4.44
Reconstruct Itinerary(332)	4.45
Count of Range Sum(327)	4.46
Insert Interval(57)	4.47
Best Time to Buy and Sell Stock with Cooldown(309)	4.48
Tree	5
Binary Tree Inorder Traversal(94)	5.1
Validate Binary Search Tree(98)	5.2
Same Tree(100)	5.3
Symmetric Tree(101)	5.4
Binary Tree Level Order Traversal(102)	5.5
Binary Tree Zigzag Level Order Traversal(103)	5.6
Maximum Depth of Binary Tree(104)	5.7
Construct Binary Tree from Preorder and Inorder Traversal(105)	5.8
Construct Binary Tree from Inorder and Postorder Traversal (106)	5.9
Binary Tree Level Order Traversal II(107)	5.10
Convert Sorted Array to Binary Search Tree(108)	5.11
Balanced Binary Tree(110)	5.12
Minimum Depth of Binary Tree(111)	5.13
Path Sum(112)	5.14
Path Sum II(113)	5.15
Flatten Binary Tree to Linked List(114)	5.16
Populating Next Right Pointers in Each Node(116)	5.17
Binary Tree Maximum Path Sum(124)	5.18
Sum Root to Leaf Numbers(129)	5.19
Invert Binary Tree(226)	5.20
Binary Tree Paths(257)	5.21
Kth Smallest Element in a BST(230)	5.22

Minimum Height Trees(310)	5.23
Count of Smaller Numbers After Self(315)	5.24
The Skyline Problem(218)	5.25
Design	6
LRU Cache(146)	6.1
Min Stack(155)	6.2
Binary Search Tree Iterator(173)	6.3
Peeking Iterator(284)	6.4
Find Median from Data Stream(295)	6.5
Implement Trie (Prefix Tree)(208)	6.6
Implement Stack using Queues(225)	6.7
Depth-first Search (Not Started)	7
Number of Islands(200)	7.1
Math	8
Power of Three(326)	8.1
Perfect Squares(279)	8.2
Power of Two(231)	8.3
Ugly Number	8.4
Ugly Number II(264)	8.5
Super Ugly Number(313)	8.6
Patching Array(330)	8.7
Hash Table(Not started)	9
Two Pointers(Not Started)	10
Trie(Not started 3)	11
Head(Not Started)	12
Greedy(Not Started)	13
Devide And Conquer(Not Started)	14
Sort(Not Started 12)	15
Breadth-first Search (Not started 18)	16
Stack(Not Started 19)	17

Binary Search(Not Started 24)	18
Backtracking(Not Started 29)	19

Cracking Leetcode in C++ - By Shuopeng Zhang

My solution to Leetcode in C++.

I am actively working on this book at this moment (March 7, 2016). My plan is to finish ~150 questions before June 2016. There is no further plan to work on this book after that.

Contact

Author: Shuopeng Zhang 张硕鹏

Email me at chang.shoup {at} gmail.com

Linked List

Add Two Numbers(2)

Question

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

Link: <https://leetcode.com/problems/add-two-numbers/>

Solution

Recursive Version

```
class Solution {
public:
    ListNode* addTwoNumbersHelper(ListNode* l1, ListNode* l2, int &carry) {
        if(l1 == nullptr && l2 == nullptr && carry == 0) return nullptr;

        int sum = (l1 == nullptr ? 0 : l1->val) +
                  (l2 == nullptr ? 0 : l2->val) + carry;

        ListNode* head = new ListNode(sum % 10);
        head->next = addTwoNumbersHelper((l1 == nullptr ? l1 : l1->next),
                                         (l2 == nullptr ? l2 : l2->next),
                                         carry);

        return head;
    }

    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        return addTwoNumbersHelper(l1, l2, 0);
    }
};
```

Iterative Version

```
class Solution {
public:
    struct ListNode* addTwoNumbers(struct ListNode* l1,
                                   struct ListNode* l2) {
        struct ListNode* rtn = new ListNode(0), *cur = rtn;
        int carry = 0;
        while(l1 || l2){
            int sum = carry;
            if(l1 != NULL){
                sum += l1->val;
                l1 = l1->next;
            }
            if(l2 != NULL){
                sum += l2->val;
                l2 = l2->next;
            }
            cur->val = sum%10;
            carry = sum/10;
            if(l1 || l2 || carry!=0){
                cur->next = new ListNode(0);
                cur = cur->next;
            }
        }
        if(carry != 0){
            cur->val = carry;
        }
        return rtn;
    }
};
```

Remove Nth Node From End of List(19)

Question

Given a linked list, remove the n-th node from the end of list and return its head.

For example,

Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.

Note: Given n will always be valid. Try to do this in one pass.

link: <https://leetcode.com/problems/remove-nth-node-from-end-of-list/>

Solution

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        if(n <= 0 || head == nullptr) return head;
        ListNode *dummy_head = new ListNode(0), *right = head,
            *left = nullptr;
        dummy_head->next = head;

        while(right->next != nullptr){
            if(--n == 0) left = dummy_head;
            right = right->next;
            left = (left == nullptr? left:left->next);
        }

        if(n == 1) left = dummy_head;
        if(left != nullptr) left->next = left->next->next;

        return dummy_head->next;
    }
};
```

Merge Two Sorted Lists(21)

Question

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

Link: <https://leetcode.com/problems/merge-two-sorted-lists/>

Solution

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if(l1 == nullptr) return l2;
        if(l2 == nullptr) return l1;
        if(l1->val > l2->val) swap(l1, l2);
        l1->next = mergeTwoLists(l1->next, l2);
        return l1;
    }
};
```

Merge k sorted lists.(23)

Question

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

Solution

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if(l1 == nullptr) return l2;
        if(l2 == nullptr) return l1;
        if(l1->val > l2->val) swap(l1, l2);
        l1->next = mergeTwoLists(l1->next, l2);
        return l1;
    }

    ListNode* mergeKLists(vector<ListNode*>& lists, int start, int end) {
        if(start == end) return lists[start];
        if(end - start == 1) return mergeTwoLists(lists[start], lists[end]);
        if(end < start) return nullptr;

        ListNode *left = mergeKLists(lists, start, (end + start)/2);
        *right = mergeKLists(lists, 1 + (end + start)/2, end);

        return mergeTwoLists(left, right);
    }

    ListNode* mergeKLists(vector<ListNode*>& lists) {
        return mergeKLists(lists, 0, lists.size()-1);
    }
};
```


Swap Nodes in Pairs(24)

Question

Given a linked list, swap every two adjacent nodes and return its head.

For example, Given 1->2->3->4, you should return the list as 2->1->4->3.

Your algorithm should use only constant space. You may not modify the values in the list, only nodes itself can be changed.

link: <https://leetcode.com/problems/swap-nodes-in-pairs/>

Solution

```
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        if(head == NULL || head->next == NULL) return head;
        ListNode *rest = swapPairs(head->next->next),
                *first = head, *second = head->next;
        second->next = first;
        first->next = rest;
        return second;
    }
};
```

Remove Duplicates from Sorted List II(82)

Question

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

For example, Given 1->2->3->3->4->4->5, return 1->2->5.

Given 1->1->1->2->3, return 2->3.

Link: <https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/>

Solution

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if(head == NULL || head->next == NULL) return head;
        if(head->val == head->next->val){
            ListNode* cur = head->next->next;
            while(cur != NULL && cur->val == head->val)
                cur = cur->next;

            return deleteDuplicates(cur);
        }else{
            head->next = deleteDuplicates(head->next);
            return head;
        }
    }
};
```

Remove Duplicates from Sorted List(83)

Question

Given a sorted linked list, delete all duplicates such that each element appear only once.

For example, Given 1->1->2, return 1->2.

Given 1->1->2->3->3, return 1->2->3.

Link: <https://leetcode.com/problems/remove-duplicates-from-sorted-list/>

Solution

```
class Solution {  
public:  
    ListNode* deleteDuplicates(ListNode* head) {  
        if(head == NULL || head->next == NULL) return head;  
        if(head->val == head->next->val){  
            return deleteDuplicates(head->next);  
        }else{  
            head->next = deleteDuplicates(head->next);  
            return head;  
        }  
    }  
};
```

Partition List(86)

Question

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

For example, Given 1->4->3->2->5->2 and x = 3,

return 1->2->2->4->3->5.

Link: <https://leetcode.com/problems/partition-list/>

Solution

```
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        if(head == nullptr || head->next == nullptr) return head;

        ListNode *left = new ListNode(0), *leftend = left,
                  *right = new ListNode(0), *rightend = right;
        while(head != nullptr){
            if(head->val < x){
                leftend->next = head;
                leftend = leftend->next;
            }else{
                rightend->next = head;
                rightend = rightend->next;
            }
            head = head->next;
        }
        rightend->next = nullptr;
        leftend->next = right->next;
        return left->next;
    }
};
```

Convert Sorted List to Binary Search Tree(109)

Question

Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

Solution

```
class Solution {
public:
    TreeNode* helper(ListNode* head, int size){
        if(size<=0) return NULL;
        if(head->next == NULL) return new TreeNode(head->val);

        int left_size = size/2, mid = 1 + size/2,
            right_size = size - mid, pos = 1;

        ListNode *left_list = head, *mid_node = NULL,
            *right_list = NULL, *cur = head;

        while(pos < mid){
            cur = cur->next;
            pos++;
        }

        TreeNode* tree = new TreeNode(cur->val);
        TreeNode* rtree = helper(cur->next, right_size);
        cur = NULL;
        TreeNode* ltree = helper(head, left_size);
        tree->left = ltree;
        tree->right = rtree;
        return tree;
    }
};
```

```
    }

    TreeNode* sortedListToBST(ListNode* head) {
        if(head == NULL) return NULL;

        ListNode* cur = head;

        int size = 0;
        while(cur != NULL){
            size++;
            cur = cur->next;
        }
        return helper(head, size);
    }
};
```

Copy List with Random Pointer(138)

Question

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.

Link: <https://leetcode.com/problems/copy-list-with-random-pointer/>

Solution


```
class Solution {
public:
    RandomListNode *copyRandomList(RandomListNode *head) {
        if(head == nullptr) return nullptr;

        RandomListNode *rtn = new RandomListNode(0),
            *cur = head, *cur_rtn = rtn;
        unordered_map<RandomListNode *, RandomListNode *> map;

        while(cur){
            cur_rtn->next = new RandomListNode(cur->label);
            map[cur] = cur_rtn->next;
            cur_rtn = cur_rtn->next;
            cur = cur->next;
        }

        cur = head, cur_rtn = rtn->next;
        while(cur){
            cur_rtn->random = map[cur->random];
            cur = cur->next;
            cur_rtn = cur_rtn->next;
        }

        return rtn->next;
    }
};
```

Linked List Cycle(141)

Question

Given a linked list, determine if it has a cycle in it.

Follow up:

Can you solve it without using extra space?

Link: <https://leetcode.com/problems/linked-list-cycle/>

Solution

```
class Solution {
public:

    bool hasCycle(ListNode *head) {
        if(head == NULL) return false;

        ListNode * fast = head;

        while(head != NULL){
            head = head->next;
            if(fast->next != NULL && fast->next->next != NULL){
                fast = fast->next->next;
            }else{
                return false;
            }
            if(head != NULL && head->next == fast->next)
                return true;
        }

        return false;
    }
};
```

Insertion Sort List(147)

Question

Sort a linked list using insertion sort.

Link: <https://leetcode.com/problems/insertion-sort-list/>

Solution

```
class Solution {
public:
    ListNode* insert(ListNode* head, ListNode* node){
        if(head == NULL) return node;
        if(head->val >= node->val){
            node->next = head;
            return node;
        }else{
            head->next = insert(head->next, node);
            return head;
        }
    }

    ListNode* insertionSortList(ListNode* head) {
        if(head == NULL) return NULL;

        ListNode *newhead = head, *node = NULL,
            *cur = head->next;

        newhead->next = NULL;

        while(cur){
            node = cur;
            cur = cur->next;
            node->next = NULL;
            newhead = insert(newhead, node);
        }
        return newhead;
    }
};
```

Intersection of Two Linked Lists(160)

Question

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:



begin to intersect at node c1.

Notes:

If the two linked lists have no intersection at all, return null. The linked lists must retain their original structure after the function returns. You may assume there are no cycles anywhere in the entire linked structure. Your code should preferably run in $O(n)$ time and use only $O(1)$ memory.

link: <https://leetcode.com/problems/intersection-of-two-linked-lists/>

Solution

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA,
                                   ListNode *headB) {
        if(headA == NULL || headB == NULL ) return NULL;

        ListNode * curA = headA, *curB = headB;

        while(curA != NULL || curB != NULL){
            if(curA == curB) return curA;
            if(curA == NULL){
                curA = headB;
            }else{
                curA = curA->next;
            }
            if(curB == NULL){
                curB = headA;
            }else{
                curB = curB->next;
            }
        }
        return NULL;
    }
};
```

Remove Linked List Elements(203)

Question

Remove all elements from a linked list of integers that have value val.

Example Given: 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, val = 6

Return: 1 --> 2 --> 3 --> 4 --> 5

Link: <https://leetcode.com/problems/remove-linked-list-elements/>

Solution

```
class Solution {
public:
    ListNode* removeElements(ListNode* head, int val) {
        if(head == NULL) return NULL;
        while(head != NULL && val == head->val) head = head->next;
        if(head == NULL) return NULL;
        ListNode* newhead = new ListNode(0), *cur = head;

        while(cur->next != NULL)
            cur->next->val == val ? cur->next = cur->next->next :
                                   cur = cur->next;

        return head;
    }
};
```


Reverse Linked List(206)

Question

Reverse a singly linked list.

Link: <https://leetcode.com/problems/reverse-linked-list/>

Solution

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* newhead = new ListNode(0);

        while(head){
            ListNode* tail = newhead->next, *temp = head;
            head = head->next;
            temp->next = tail;
            newhead->next = temp;
        }

        return newhead->next;
    }
};
```

Palindrome Linked List(234)

Question

Given a singly linked list, determine if it is a palindrome.

Follow up: Could you do it in $O(n)$ time and $O(1)$ space?

Link: <https://leetcode.com/problems/palindrome-linked-list/>

Solution

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* newhead = new ListNode(0);

        while(head){
            ListNode* tail = newhead->next, *temp = head;
            head = head->next;
            temp->next = tail;
            newhead->next = temp;
        }

        return newhead->next;
    }

    bool match(ListNode* first, ListNode* second){
        while(first&&second){
            if(first->val == second->val){
                first = first->next;
                second = second->next;
            }else{
                return false;
            }
        }
        return true;
    }
};
```

```
    }

    bool isPalindrome(ListNode* head) {
        if(head == NULL || head->next == NULL) return true;

        ListNode *cur = head, *first = NULL, *second = NULL;
        int size = 0, pos = 1;
        while(cur) {
            cur = cur->next;
            size++;
        }

        cur = head;
        while(cur->next && pos < size/2){
            cur = cur->next;
            pos++;
        }
        first = head;
        if(size%2 == 0){
            second = cur->next;
            cur->next = NULL;
        }else{
            second = cur->next->next;
            cur->next = NULL;
        }
        return match(reverseList(first), second);
    }
};
```

Odd Even Linked List(328)

Question

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in $O(1)$ space complexity and $O(\text{nodes})$ time complexity.

Example: Given 1->2->3->4->5->NULL,

return 1->3->5->2->4->NULL.

Note: The relative order inside both the even and odd groups should remain as it was in the input. The first node is considered odd, the second node even and so on ...

Link: <https://leetcode.com/problems/odd-even-linked-list/>

Solution

```
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if(!head || !head->next || !head->next->next)
            return head;

        ListNode *first = new ListNode(0),
                  *second = new ListNode(0),
                  *cur = head, *firstCur = first,
                  *secondCur = second;

        int pos = 1;

        while(cur){
            if(pos%2 == 1){
                firstCur->next = cur;
                cur = cur->next;
                firstCur = firstCur->next;
            }else{
                secondCur->next = cur;
                cur = cur->next;
                secondCur = secondCur->next;
            }
            pos++;
        }
        secondCur->next = NULL;
        firstCur->next = second->next;
        return first->next;
    }
};
```

Graph

Includes the questions from graph section

Clone Graph(133)

Question

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.

Link: <https://leetcode.com/problems/clone-graph/>

Solution

```

class Solution {
public:
    std::unordered_map<UndirectedGraphNode*,
                    UndirectedGraphNode *> mymap;

    UndirectedGraphNode *cloneGraph(UndirectedGraphNode *node) {
        if(node == NULL) return NULL;

        std::queue<UndirectedGraphNode *> myqueue;
        myqueue.push(node);

        UndirectedGraphNode * G =
            new UndirectedGraphNode(node->label);
        mymap.insert({node, G});

        while(myqueue.size() != 0){
            UndirectedGraphNode
                *left_key_node = myqueue.front(),
                *right_key_node =
                    mymap.find(left_key_node)->second;
            myqueue.pop();

            for(auto& it : left_key_node->neighbors){
                auto it_mymap = mymap.find(it);
                if(it_mymap == mymap.end()){
                    UndirectedGraphNode * copiedNode =
                        new UndirectedGraphNode(it->label);
                    myqueue.push(it);
                    mymap.insert({it, copiedNode});
                    right_key_node->neighbors.push_back(copiedNode);
                }else
                    right_key_node->neighbors.push_back(
                        it_mymap->second);
            }
        }
        return G;
    }
};

```


Strings

Questions can be found here <https://leetcode.com/tag/string/>.

Integer to RomanLongest Substring Without Repeating Characters(3)

Question

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbb" the longest substring is "b", with the length of 1.

link: <https://leetcode.com/problems/longest-substring-without-repeating-characters/>

Solution

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        if(s.empty()) return 0;
        int left = 0, right = 1, s_size = s.size(), max_size = 0;
        unordered_map<char, int> map;
        map[s[left]] = left;
        max_size = map.size();
        while(right < s_size){
            auto it = map.find(s[right]);
            if(it == map.end()){
                map[s[right]] = right;
            }else{
                left = max(left, it->second+1);
                map[s[right]] = right;
            }

            max_size = max(max_size, right-left+1);
            right++;
        }

        return max_size;
    }
};
```

Longest Palindromic Substring(5)

Question

Given a string S, find the longest palindromic substring in S. You may assume that the maximum length of S is 1000, and there exists one unique longest palindromic substring.

Link: <https://leetcode.com/problems/longest-palindromic-substring/>

Solution

```
class Solution {
public:
    string longestPalindrome(string s) {
        int total_size = s.size(), start = 0, max_len = -1;
        if(total_size <= 1) return s;

        for(int i = 0; i < total_size; i++){
            int j = i, k = i;
            if(total_size - i <= max_len/2) break;
            while(k < total_size - 1 && s[k + 1] == s[k]) k++;
            i = k;
            while(k+1 <= total_size - 1 &&
                j-1 >= 0 &&
                s[k+1] == s[j-1]){ k++; j--;}
            if(k - j + 1 > max_len) {max_len = k-j+1; start = j;}
        }

        return s.substr(start, max_len);
    }
};
```

Integer to Roman (12)

Question

Given an integer, convert it to a roman numeral. Input is guaranteed to be within the range from 1 to 3999.

Link: <https://leetcode.com/problems/integer-to-roman/>

Solution

```
class Solution {
public:
    vector<vector<string>> s = {{ "I", "V", "X"},
                                {"X", "L", "C"},
                                {"C", "D", "M"},
                                {"M"}};

    string convert(int num, vector<string>& strs){
        if(num == 0) return {};
        string rtn;
        if(num <= 3){
            while(num-- > 0) rtn.append(strs[0]);
        }else if(num == 4) rtn = strs[0] + strs[1];
        else if(num == 9) rtn = strs[0] + strs[2];
        else{
            rtn.append(strs[1]);
            num -= 5;
            while(num-- > 0) rtn.append(strs[0]);
        }
        return rtn;
    }

    string intToRoman(int num) {
        string rtn = "";
        for(int i = 3; i >= 0; i--){
            rtn.append(convert(((int)(num/pow(10.0, i))%10),
                               s[i]));
        }
        return rtn;
    }
};
```

Roman to Integer(13)

Question

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

Link: <https://leetcode.com/problems/roman-to-integer/>

Solution

Solution 1

This code runs about 48~54 ms


```
class Solution {
public:

    int romanToInt(string s) {
        vector<int> vec = {};
        int rtn = 0;
        for(char c : s){
            if(c == 'M'){
                vec.push_back(1000);
            }else if(c == 'D'){
                vec.push_back(500);
            }else if(c == 'C'){
                vec.push_back(100);
            }else if(c == 'L'){
                vec.push_back(50);
            }else if(c == 'X'){
                vec.push_back(10);
            }else if(c == 'V'){
                vec.push_back(5);
            }else if(c == 'I'){
                vec.push_back(1);
            }else assert(1 == 0); // Invalid char
        }

        for(int i = 0; i < vec.size(); i++){
            if(i != vec.size() - 1 && vec[i] < vec[i+1])
                rtn -= vec[i];
            else rtn += vec[i];
        }
        return rtn;
    }
};
```

Solution 2

This code runs about 68 ms

```
class Solution {
public:
    unordered_map<char, int> map = {{'M', 1000}, {'D', 500},
                                    {'C', 100}, {'L', 50},
                                    {'X', 10}, {'V', 5},
                                    {'I', 1}};

    int romanToInt(string s) {
        int rtn = 0;
        int s_size = s.size();
        for(int i = 0; i < s_size; i++){
            if(i != s_size - 1 && map[s[i]] < map[s[i+1]])
                rtn -= map[s[i]];
            else rtn += map[s[i]];
        }
        return rtn;
    }
};
```

Longest Common Prefix(14)

Question

Write a function to find the longest common prefix string amongst an array of strings.

Link: <https://leetcode.com/problems/longest-common-prefix/>

Solution

Solution 1

The basic idea is divide-and-conquer. This code runs about 8 ms.

```
class Solution {
public:
    string conquer(string& str1, string& str2){
        for(int i = 0; i < min(str1.size(), str2.size()); i++){
            if(str1[i] != str2[i]){
                return str1.substr(0, i);
            }
        }

        return str1.size() > str2.size() ? str2:str1;
    }

    string devide(vector<string>& strs, int start, int end){
        if(start > end) return "";
        if(start == end) return strs[start];

        int mid = (start + end) / 2;
        string str1 = devide(strs, start, mid), str2 =
            devide(strs, mid+1, end);
        return conquer(str1, str2);
    }

    string longestCommonPrefix(vector<string>& strs) {
        return devide(strs, 0, strs.size() - 1);
    }
};
```

Solution 2

This code runs about 4-8 ms

```
class Solution {
public:
    int helper(string oristr, string newstr){
        for(int i = 0; i < min(oristr.size(),newstr.size());i++){
            if(oristr[i]!=newstr[i]) return i;
        }
        return min(oristr.size(), newstr.size());
    }

    string longestCommonPrefix(vector<string>& strs) {
        if(strs.size() == 0) return "";
        string rtn = strs[0];
        for(int i = 1; i < strs.size() && rtn!=""; i++){
            rtn = rtn.substr(0,helper(rtn, strs[i]));
        }
        return rtn;
    }
};
```

Letter Combinations of a Phone Number(17)

Question

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.



Input: Digit string "23" Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

Note: Although the above answer is in lexicographical order, your answer could be in any order you want.

Link: <https://leetcode.com/problems/letter-combinations-of-a-phone-number/>

Solution

```
class Solution {
public:

    vector<vector<string> > map = {{ "a", "b", "c"}, {"d", "e", "f"},
                                    {"g", "h", "i"}, {"j", "k", "l"},
                                    {"m", "n", "o"}, {"p", "q", "r", "s"},
                                    {"t", "u", "v"}, {"w", "x", "y", "z"}];

    vector<string> helper(string digits){
        if(digits.size() == 0) return {};
        if(digits.size() == 1) return map[(int)digits[0] - '2'];

        vector<string> rtn = {},
            temp = helper(digits.substr(1, string::npos));

        for(string c : map[(int)digits[0] - '2'])
            for(string r: temp) rtn.push_back(c+r);

        return rtn;
    }

    vector<string> letterCombinations(string& digits) {
        return helper(digits);
    }

};
```

Valid Parentheses(20)

Question

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(" and "([])" are not.

Link: <https://leetcode.com/problems/valid-parentheses/>

Solution

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> stk;
        for(char c : s){
            if(c == '(' || c == '{' || c == '[') stk.push(c);
            else if(!stk.empty() &&
                    ((c == ')' && stk.top() == '(') ||
                     (c == '}' && stk.top() == '{') ||
                     (c == ']' && stk.top() == '[')))
                stk.pop();
            else return false;
        }
        return stk.empty();
    }
};
```


Generate Parentheses(22)

Question

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:

\$\$\$"((()))", "(()())", "(())()", "()(())", "()()()"\$\$\$

Link: <https://leetcode.com/problems/generate-parentheses/>

Solution

```
class Solution {
public:
    vector<string> result = {};
    void helper(string str, int open, int close){
        if(open == 0 && close == 0) result.push_back(str);
        if(open > 0) helper(str + '(', open - 1, close);
        if(open < close) helper(str + ')', open, close - 1);
    }

    vector<string> generateParenthesis(int n) {
        helper("", n, n);
        return result;
    }
};
```

Implement strStr()(28)

Question

Implement strStr().

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Solution

```
class Solution {
public:

    int strStr(string haystack, string needle) {
        int h_size = haystack.size(), n_size = needle.size();
        if((h_size == 0 && n_size == 0) ||
            (h_size != 0 && n_size == 0)) return 0;

        for(int i = 0; i <= h_size - n_size; i++){
            for(int j = 0; j < n_size; j++){
                if(haystack[i+j] != needle[j]) break;
                if(j + 1 == n_size) return i;
            }
        }
        return -1;
    }
};
```

Multiply Strings(43)

Question

Given two numbers represented as strings, return multiplication of the numbers as a string.

Note: The numbers can be arbitrarily large and are non-negative.

Solution

```
class Solution {
public:
    string multiply(string num1, string num2) {
        int n1_size = num1.size(), n2_size = num2.size();
        if(n1_size == 0 || n2_size == 0) return "0";
        vector<int> storage(n1_size + n2_size, 0);
        string rtn = "";

        for(int i = 0; i < n1_size; i++)
            for(int j = 0; j < n2_size; j++)
                storage[i+j + 1] +=
                    (num1[i] - '0') * (num2[j] - '0');
        reverse(storage.begin(), storage.end());
        int carry = 0;
        for(int i = 0; i < n1_size + n2_size; i++){
            int sum = carry + storage[i];
            rtn.append(to_string(sum%10));
            carry = sum/10;
        }

        reverse(rtn.begin(), rtn.end());

        // remove starting '0's
        int pos = 0;
        while(pos < rtn.size())
            if(rtn[pos] == '0' && rtn.size() != 1)
                rtn = rtn.substr(1);
            else break;

        return rtn;
    }
};
```

Group Anagrams(49)

Question

Given an array of strings, group anagrams together.

For example, given: ["eat", "tea", "tan", "ate", "nat", "bat"], Return:

```
[
  ["ate", "eat", "tea"],
  ["nat", "tan"],
  ["bat"]
]
```

Note: For the return value, each inner list's elements must follow the lexicographic order. All inputs will be in lower-case.

Solution

```
class Solution {
public:
    vector<vector<string> > groupAnagrams(vector<string>& strs) {
        vector<vector<string> > rtn = {};
        if(strs.empty()) return rtn;

        unordered_multimap<string, string> map;
        unordered_set<string> set;
        for(int i = 0; i < strs.size(); i++){
            string copy = strs[i];
            sort(copy.begin(), copy.end());
            map.insert({copy, strs[i]});
            set.insert(copy);
        }

        for (auto it = set.begin(); it != set.end(); it++) {
            auto range = map.equal_range(*it);
            vector<string> temp;
            for (auto it2=range.first;it2!=range.second;it2++)
                temp.push_back(it2->second);
            sort(temp.begin(), temp.end());
            rtn.push_back(temp);
        }

        return rtn;
    }
};
```

Length of Last Word(58)

Question

Given a string *s* consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string.

If the last word does not exist, return 0.

Note: A word is defined as a character sequence consists of non-space characters only.

For example, Given *s* = "Hello World", return 5.

Link: <https://leetcode.com/problems/length-of-last-word/>

Solution

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        int start = 0, end = s.size()-1;
        if(s.empty()) return 0;

        while(end >= 0 && s[end] == ' ') end--;

        if(end == -1) return 0;
        start = end;

        while(start >=0 && s[start] != ' ') start--;

        return end - start;
    }
};
```

Add Binary(67)

Question

Given two binary strings, return their sum (also a binary string).

For example,

a = "11"

b = "1"

Return "100".

Solution


```
class Solution {
public:

    string helper(string a, string b, int carry, int a_pos,
                  int b_pos){
        if(a_pos < 0 && b_pos < 0)
            return carry == 1? "1" : "";
        char rtn = ' ';
        int ac = a_pos >= 0 ? a[a_pos] - '0' : 0,
            bc = b_pos >= 0 ? b[b_pos] - '0' : 0,
            sum = carry + ac + bc;
        if(sum == 3){
            rtn = '1'; carry = 1;
        }else if(sum == 2){
            rtn = '0'; carry = 1;
        }else if(sum == 1){
            rtn = '1'; carry = 0;
        }else
            rtn = '0'; carry = 0;

        return helper(a, b, carry, a_pos - 1, b_pos - 1) + rtn;
    }

    string addBinary(string a, string b) {
        return helper(a, b, 0, a.size()-1, b.size()-1);
    }
};
```

Edit Distance(72)

Question

Given two words word1 and word2, find the minimum number of steps required to convert word1 to word2. (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

- a) Insert a character
- b) Delete a character
- c) Replace a character

Solution

```
class Solution {
public:
    int minDistance(string word1, string word2) {
        if(word1.size() == 0 && 0 == word2.size()) return 0;
        if(word1.size() == 0) return word2.size();
        if(word2.size() == 0) return word1.size();

        vector<int> temp(word2.size()+1, 0);
        for (int i = 1; i < temp.size(); i++) temp[i] = i;
        vector<vector<int>> > table(word1.size()+1, temp);
        for (int i = 1; i < table.size(); i++) table[i][0] = i;

        for(int i = 1; i <= word1.size(); i++){
            for(int j = 1; j <= word2.size(); j++){
                if(word1[i-1] != word2[j-1]) table[i][j] =
                    min(table[i-1][j-1]+1, min(table[i-1][j],
                                                table[i][j-1])+1);

                else table[i][j] =
                    min(table[i-1][j-1], min(table[i-1][j],
                                                table[i][j-1])+1);
            }
        }

        return table[word1.size()][word2.size()];
    }
};
```

Decode Ways(91)

Question

A message containing letters from `A-Z` is being encoded to numbers using the following mapping:

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

Given an encoded message containing digits, determine the total number of ways to decode it.

For example,

Given encoded message `"12"` , it could be decoded as `"AB"` (1 2) or `"L"` (12) .

The number of ways decoding `"12"` is `2` .

Solution

```
class Solution {
public:
    unordered_map<int, int> map;
    int helper(string& s, int start){
        if(map.find(start) != map.end()) return map[start];
        if(start >= s.size()) return 0;
        if(start == s.size() - 1)
            return s[start] == '0' ? (map[start] = 0):
                                   (map[start] = 1);
        if(s[start] == '0') return map[start] = 0;

        int two_char = atoi(s.substr(start, 2).c_str()),
            first = helper(s, start + 1),
            second = (two_char <= 26 && two_char >= 10) ?
                    ((start == s.size() - 2) ?
                     1 : helper(s, start + 2)) : 0;

        return (map[start] = first+second);
    }

    int numDecodings(string s) {
        return helper(s, 0);
    }
};
```

Restore IP Addresses(93)

Question

Given a string containing only digits, restore it by returning all possible valid IP address combinations.

For example:

Given "25525511135" ,

return ["255.255.11.135", "255.255.111.35"] . (Order does not matter)

Solution

```
class Solution {
public:
    vector<string> helper(string& s, int start, int dot) {
        if(s.size() - start < dot || s.size() - start > 3 * dot)
            return {};

        int first = atoi(s.substr(start, 1).c_str()),
            second = atoi(s.substr(start, 2).c_str()),
            third = atoi(s.substr(start, 3).c_str());
        if(dot == 1){
            if(third >= 0 && third <= 255 &&
               (s[start] != '0' || start == s.size()-1))
                return {s.substr(start, 3)};
            else
                return {};
        }
        vector<string> rtn(helper(s, start + 1, dot - 1)),
            rtn1, rtn2;
        if((s.size() - start) >= 2 && (second >= 0 &&
            second <= 255) && s[start] != '0')
            rtn1 = helper(s, start + 2, dot - 1);
        if((s.size() - start) >= 3 && (third >= 0 &&
```

```
        third <= 255) && s[start] != '0')
            rtn2 = helper(s, start + 3, dot - 1);

        for(int i = 0; i < rtn.size(); i++)
            rtn[i] = s.substr(start, 1) + "." + rtn[i];

        for(int i = 0; i < rtn1.size(); i++)
            rtn.push_back(s.substr(start, 2) + '.' + rtn1[i]);

        for(int i = 0; i < rtn2.size(); i++)
            rtn.push_back(s.substr(start, 3) + "." + rtn2[i]);

        return rtn;
    }

    vector<string> restoreIpAddresses(string s) {
        return helper(s, 0, 4);
    }
};
```

Valid Palindrome(125)

Question

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

```
For example,  
"A man, a plan, a canal: Panama" is a palindrome.  
"race a car" is not a palindrome.
```

Note: Have you consider that the string might be empty? This is a good question to ask during an interview.

For the purpose of this problem, we define empty string as valid palindrome.

Solution

Solution 1 Recursive version


```

class Solution {
public:
    bool helper(string& s, int start, int end){
        if(s.empty() || start > end) return true;
        while(start < end && !isalnum(s[start])) start++;
        while(end > start && !isalnum(s[end])) end--;
        return (start>end || tolower(s[start]) ==
                tolower(s[end])) ?
            helper(s, start+1, end-1) : false;
    }

    bool isPalindrome(string s) {
        return helper(s, 0, s.size()-1);
    }
};

```

Solution 2 Iterative version

```

class Solution {
public:
    bool isPalindrome(string s) {
        int i = 0, j = s.size()-1;
        while(i < j){
            if(isalnum(s[i]) && isalnum(s[j])){
                if(tolower(s[i]) != tolower(s[j]))
                    return false;
                i++; j--;
            }else{
                while(!isalnum(s[i]) && i < j) i++;
                while(!isalnum(s[j]) && i < j) j--;
            }
        }
        return true;
    }
};

```

Reverse Words in a String(151)

Question

Given an input string, reverse the string word by word.

For example, Given s = "the sky is blue", return "blue is sky the".

Clarification:

- * What constitutes a word?

A sequence of non-space characters constitutes a word.

- * Could the input string contain leading or trailing spaces?

Yes. However, your reversed string should not contain leading or trailing spaces.

- * How about multiple spaces between two words?

Reduce them to a single space in the reversed string.

Solution

```
class Solution {
public:
    void reverseWords(string &s) {
        int left = 0, right = 0;
        string rtn = "";
        while(left < s.size()){
            while(left < s.size() && s[left] == ' ') left++;
            right = left + 1;
            while(right < s.size() && s[right] != ' ') right++;
            if(left < s.size() && right <= s.size())
                rtn = s.substr(left, right - left) +
                    (rtn.empty() ? "": ' ' + rtn);
            left = right;
        }
        s = rtn;
    }
};
```

Compare Version Numbers(165)

Question

Compare two version numbers version1 and version2.

If version1 > version2 return 1, if version1 < version2 return -1, otherwise return 0.

You may assume that the version strings are non-empty and contain only digits and the `.` character.

The `.` character does not represent a decimal point and is used to separate number sequences.

For instance, `2.5` is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision.

Here is an example of version numbers ordering:

```
0.1 < 1.1 < 1.2 < 13.37
```

Solution

```
class Solution {
public:
    int helper(string& v1, string& v2, int p1, int p2){
        int d1 = v1.find_first_of('.', p1),
            d2 = v2.find_first_of('.', p2),
            val1 = (p1 == string::npos)? 0 :
                atoi(v1.substr(p1, d1 == string::npos ?
                    d1 : d1 - p1).c_str()),
            val2 = (p2 == string::npos)? 0 :
                atoi(v2.substr(p2, d2 == string::npos ?
                    d2 : d2 - p2).c_str());

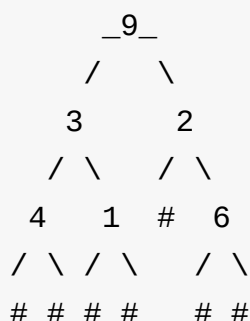
        if(val1 > val2) return 1;
        if(val1 < val2) return -1;
        return (d1 == string::npos && d2 == string::npos) ?
            0 :
            helper(v1, v2, d1 == string::npos? d1 : d1 + 1,
                d2 == string::npos? d2 : d2 + 1);
    }

    int compareVersion(string version1, string version2) {
        return helper(version1, version2, 0, 0);
    }
};
```

Verify Preorder Serialization of a Binary Tree(331)

Question

One way to serialize a binary tree is to use pre-order traversal. When we encounter a non-null node, we record the node's value. If it is a null node, we record using a sentinel value such as #.



For example, the above binary tree can be serialized to the string **"9,3,4,#,#,1,#,#,2,#,6,#,#"**, where # represents a null node.

Given a string of comma separated values, verify whether it is a correct preorder traversal serialization of a binary tree. Find an algorithm without reconstructing the tree.

Each comma separated value in the string must be either an integer or a character '#' representing null pointer.

You may assume that the input format is always valid, for example it could never contain two consecutive commas such as **"1,,3"**.

Example 1:

```
"9, 3, 4, #, #, 1, #, #, 2, #, 6, #, #"
```

Return **true**

Example 2:

```
"1,#"
```

Return **false****Example 3:**

```
"9,#,#,1"
```

Return **false**

Solution

```
class Solution {
public:
    int helper(string& preorder, int start){
        if (start > preorder.size() - 1) return -1;
        if(preorder[start] == '#')
            return start + 2;
        else{
            while(isdigit(preorder[start])) start++;
            int right_start = helper(preorder, start + 1);
            if(right_start < preorder.size())
                return helper(preorder, right_start);
        }
        return -1;
    }

    bool isValidSerialization(string preorder) {
        int rtn = helper(preorder, 0);
        return (rtn == -1 || rtn < preorder.size())? false: true;
    }
};
```

Maximum Product of Word Lengths(318)

Question

Given a string array `words` , find the maximum value of `length(word[i]) * length(word[j])` where the two words do not share common letters. You may assume that each word will contain only lower case letters. If no such two words exist, return 0.

Example 1:

Given `["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]`

Return `16`

The two words can be `"abcw", "xtfn"` .

Example 2:

Given `["a", "ab", "abc", "d", "cd", "bcd", "abcd"]`

Return `4`

The two words can be `"ab", "cd"` .

Example 3:

Given `["a", "aa", "aaa", "aaaa"]`

Return `0`

No such pair of words.

Solution


```
class Solution {
public:
    int set_bit(string& str){
        int mybits = 0;
        for (char c : str){
            mybits |= (1 << (c-'a'));
            if ((mybits == 0x3FFFFFF)) break;
        }
        return mybits;
    }

    int maxProduct(vector<string>& words) {
        int m_val = 0, w_size = words.size();
        int m[w_size], m_w_size[w_size];

        for(int i = 0; i < w_size; i++) {
            m[i] = set_bit(words[i]);
            m_w_size[i] = words[i].size();
        }

        for (int i = 0; i < w_size; i++)
            for (int j = i+1; j < w_size; j++)
                if ((m[i] & m[j])==0)
                    m_val = max((int)(m_w_size[i] * m_w_size[j]),
                                m_val);

        return m_val;
    }
};
```

Basic Calculator(224)

Question

Implement a basic calculator to evaluate a simple expression string.

The expression string may contain open `(` and closing parentheses `)`, the plus `+` or minus sign `-`, **non-negative** integers and empty spaces .

You may assume that the given expression is always valid.

Some examples:

```
"1 + 1" = 2
" 2-1 + 2 " = 3
"(1+(4+5+2)-3)+(6+8)" = 23
```

Note: Do not use the `eval` built-in library function.

Solution

```

class Solution {
public:
    int helper(string& s, int start, int end, int * return_pos){
        int sum = 0, op = 1; // 1 for +; -1 for -
        for (int i = start+1; i < end; i++)
            if (s[i] == ' ')
                continue;
            else if (s[i] == '+' || s[i] == '-')
                op = (s[i] == '+') ? 1:-1;
            else if(s[i] == '(')
                sum += op*helper(s, i, end, &i);
            else if(s[i] == ')'){
                *return_pos = i; return sum;
            }else{
                int j = 0, num = 0;
                while (isdigit(s[i+j]))
                    num = num*10 + (s[i+(j++)]-'0');
                i = i+j-1; sum+=op*num;
            }

        return sum;
    }

    int calculate(string s) {
        if(s.empty()) return 0;
        s = "("+s+")";
        return helper(s, 0, s.size()-1, nullptr);
    }
};

```

Palindrome Pairs(336)

Question

Given a list of unique words. Find all pairs of distinct indices (i, j) in the given list, so that the concatenation of the two words, i.e. `words[i] + words[j]` is a palindrome.

Example 1: Given `words = ["bat", "tab", "cat"]`

Return `[[0, 1], [1, 0]]`

The palindromes are `["battab", "tabbat"]`

Example 2: Given `words = ["abcd", "dcba", "lls", "s", "sssll"]`

Return `[[0, 1], [1, 0], [3, 2], [2, 4]]`

The palindromes are `["dcbabcd", "abcdcba", "slls", "llssssll"]`

Solution

```
bool isPalin(string& str, int start, int end){
    if (start == end) return true;
    while (start < end) {
        if (str[start] != str[end]) return false;
        start++; end--;
    }
    return true;
}

void add_to_rtn(vector<vector<int>>& rtn, unordered_map<string, int>
               string& word, int i, bool rev){
    auto it = map.find(word);
    if(it != map.end() && it->second != i){
        if(!rev){
            rtn.push_back({i, it->second});
        }
    }
}
```

```

        if (word.empty()) rtn.push_back({it->second, i});
    }else rtn.push_back({it->second, i});
    }
}

vector<vector<int>> palindromePairs(vector<string>& words) {
    vector<vector<int>> rtn = {};
    int words_size = words.size();
    if (words_size < 2) return rtn;
    unordered_map<string, int> map;
    string empty_word = "";
    for (int i = 0; i < words_size; ++i) map[words[i]] = i;

    for (int i = 0; i < words_size; ++i) {
        if (words[i].empty()) continue;
        string& word = words[i], rword = words[i];
        int word_size = word.size();
        reverse(rword.begin(), rword.end());
        add_to_rtn(rtn, map, isPalin(word, 0, word_size-1)? empty_v

        for (int j = 0; j < word_size; ++j) {
            if (j+1 < word_size && isPalin(word, 0, j)){
                string sub_string = rword.substr(0, word_size - j - 1);
                add_to_rtn(rtn, map, sub_string, i, true);
            }
            if (0 < j && j < word_size && isPalin(word, j, word_size-1)){
                string sub_string = rword.substr(word_size - j - 1, j);
                add_to_rtn(rtn, map, sub_string, i, false);
            }
        }
    }

    return rtn;
}

```

Array(Not Uploaded)

Two Sum(1)

Question

Given an array of integers, return indices of the two numbers such that they add up to a specific target.

You may assume that each input would have exactly one solution.

Example: Given nums = [2, 7, 11, 15], target = 9,

Because $\text{nums}[0] + \text{nums}[1] = 2 + 7 = 9$, return [0, 1].

UPDATE (2016/2/13): The return format had been changed to zero-based indices. Please read the above updated description carefully.

Link: <https://leetcode.com/problems/two-sum/>

Solution

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int, int> map;
        for(int i = 0; i < nums.size(); i++){
            int diff = target - nums[i];
            auto it = map.find(nums[i]);
            if(it != map.end()){
                return {it->second, i};
            }else{
                map[diff] = i;
            }
        }

        return {};
    }
};
```


3Sum(15)

Question

Given an array S of n integers, are there elements a, b, c in S such that $a + b + c = 0$? Find all unique triplets in the array which gives the sum of zero.

Note:

- Elements in a triplet (a,b,c) must be in non-descending order. (ie, $a \leq b \leq c$)
- The solution set must not contain duplicate triplets.

For example, given array $S = \{-1\ 0\ 1\ 2\ -1\ -4\}$,

A solution set is:

$(-1, 0, 1)$

$(-1, -1, 2)$

Solution

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums){
        if(nums.size() <=2) return {};
        sort(nums.begin(), nums.end());
        vector<vector<int>> rtn = {};
        int size = nums.size();
        for(int i = 0; i < size - 2; ){
            int l = i+1, r = size - 1;
            while(l < r){
                if(nums[i] + nums[l] + nums[r] == 0){
                    rtn.push_back({nums[i], nums[l], nums[r]});
                    do{r--;}while(l < r && nums[r] == nums[r+1]);
                    do{l++;}while(l < r && nums[l] == nums[l-1]);
                }else if(nums[i] + nums[l] + nums[r] > 0){
                    do{r--;}while(l < r && nums[r] == nums[r+1]);
                }else{
                    do{l++;}while(l < r && nums[l] == nums[l-1]);
                }
            }
            do {i++;} while(i != 0 &&
                        nums[i] == nums[i-1] &&
                        i < size);
        }
        return rtn;
    }
};
```

Remove Duplicates from Sorted Array(26)

Question

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example, Given input array nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively. It doesn't matter what you leave beyond the new length.

Solution

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if(nums.size() <= 1 ) return nums.size();
        int pos = 1;
        while(pos < nums.size()){
            if(nums[pos] != nums[pos-1])
                pos++;
            else{
                nums.erase(nums.begin()+pos);
            }
        }
        return nums.size();
    }
};
```

Remove Element(27)

Question

Given an array and a value, remove all instances of that value in place and return the new length.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

Solution

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        if(nums.size() == 0) return 0;
        int end = nums.size()-1;
        for(int i = 0; i <= end;){
            if(nums[i] == val) swap(nums[i], nums[end--]);
            else i++;
        }
        return end+1;
    }
};
```

Next Permutation(31)

Question

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

1, 2, 3 → 1, 3, 2

3, 2, 1 → 1, 2, 3

1, 1, 5 → 1, 5, 1

Solution

```
class Solution {
public:
    void nextPermutation(std::vector<int> &nums) {
        if(nums.size() <= 1) return;
        int left = nums.size()-2, right = nums.size()-1;

        while(true){
            if(nums[left] < nums[right])
                break;
            else{
                if(right != left) right--;
                else{
                    left--;
                    if(left == -1) break;
                    right = nums.size()-1;
                }
            }
        }
        if(left != -1) swap(nums[left], nums[right]);
        else left = -1;
        sort(nums.begin()+left+1, nums.end());
        return;
    }
};
```

Search for a Range(34)

Question

Given a sorted array of integers, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of $O(\log n)$.

If the target is not found in the array, return $[-1, -1]$.

For example,

```
Given [5, 7, 7, 8, 8, 10] and target value 8,  
return [3, 4].
```

Solution

```
class Solution {
public:
    vector<int> helper(vector<int>& nums, int target,
                      int start, int end){
        if(nums.size() == 0 || start > end) return {};
        if(start == end && nums[start] == target)
            return {start, end};
        if(start == end) return {};
        int mid = (start + end)/2;

        if(target > nums[mid])
            return helper(nums, target, mid+1, end);
        else if(target < nums[mid])
            return helper(nums, target, start, mid);
        else{
            vector<int> l = helper(nums, target, start, mid),
                       r = helper(nums, target, mid+1, end);
            int p1 = mid, p2 = mid;
            if(l.size() != 0) p1 = l[0];
            if(r.size() != 0) p2 = r[1];
            return {p1, p2};
        }
    }

    vector<int> searchRange(vector<int>& nums, int target) {
        vector<int> rtn = helper(nums, target, 0, nums.size()-1);
        if (rtn.size() != 0) return {rtn[0], rtn[1]};
        else return {-1, -1};
    }
};
```


Search Insert Position(35)

Question

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples.

```
[1,3,5,6], 5 → 2  
[1,3,5,6], 2 → 1  
[1,3,5,6], 7 → 4  
[1,3,5,6], 0 → 0
```

Solution

```
class Solution {  
public:  
    int searchInsert(vector<int> nums, int target) {  
        for(int i = 0; i < nums.size(); i++)  
            if(nums[i] >= target) return i;  
  
        return nums.size();  
    }  
};
```

Combination Sum(39)

Question

Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

Note:

- All numbers (including target) will be positive integers.
- Elements in a combination (a_1, a_2, \dots, a_k) must be in non-descending order. (ie, $a_1 \leq a_2 \leq \dots \leq a_k$).
- The solution set must not contain duplicate combinations. For example, given candidate set `2,3,6,7` and target `7`, A solution set is:

```
[7]
[2, 2, 3]
```

Solution

```
class Solution {
public:
    vector<vector<int>> helper(vector<int>& candidates,
                              int target, int start){
        if(candidates.size() == 0 || target <= 0) return {};
        vector<vector<int>> rtn = {};
        for(int i = start; i < candidates.size(); i++){
            while(i != 0 && candidates[i] == candidates[i-1]) i++;

            int newtarget = target - candidates[i];
            if(newtarget<0) continue;

            vector<vector<int>> temp = helper(candidates,
                                              newtarget, i);

            if(newtarget == 0) temp.push_back({candidates[i]});
            else
                for(int j = 0; j < temp.size(); j++) {
                    temp[j].push_back(candidates[i]);
                    sort(temp[j].begin(), temp[j].end());
                }
            rtn.insert(rtn.end(), temp.begin(), temp.end());
        }
        return rtn;
    }

    vector<vector<int>> combinationSum(vector<int>& candidates,
                                     int target) {
        sort(candidates.begin(), candidates.end());
        return helper(candidates, target, 0);
    }
};
```

Combination Sum II(40)

Question

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.

Note:

- All numbers (including target) will be positive integers.
- Elements in a combination (a_1, a_2, \dots, a_k) must be in non-descending order. (ie, $a_1 \leq a_2 \leq \dots \leq a_k$).
- The solution set must not contain duplicate combinations. For example, given candidate set `10,1,2,7,6,1,5` and target `8`, A solution set is:

```
[[1, 7] [1, 2, 5] [2, 6] [1, 1, 6]]
```

```
## Solution
```cpp
class Solution {
public:
 vector<vector<int>> helper(vector<int>& candidates,
 int target, int start){
 if(candidates.size() == 0 || target <= 0 ||
 start >= candidates.size())
 return {};
 vector<vector<int>> rtn = {};
 for(int i = start; i < candidates.size(); i++){
 while(i != start &&
 candidates[i] == candidates[i-1] &&
 i < candidates.size()) i++;
 if(i >= candidates.size()) break;
 int newtarget = target - candidates[i];
 if(newtarget < 0) continue;
```

```
 vector<vector<int>> temp = helper(candidates,
 newtarget,
 i+1);

 if(newtarget == 0) temp.push_back({candidates[i]});
 else
 for(int j = 0; j < temp.size(); j++) {
 temp[j].push_back(candidates[i]);
 sort(temp[j].begin(), temp[j].end());
 }
 rtn.insert(rtn.end(), temp.begin(), temp.end());
 }
 return rtn;
}

vector<vector<int>> combinationSum2(vector<int>& candidates,
 int target) {
 sort(candidates.begin(), candidates.end());
 return helper(candidates, target, 0);
}

};
```

# Rotate Image(48)

## Question

You are given an  $n \times n$  2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Follow up: Could you do this in-place?

## Solution

```
class Solution {
public:
 void rotate(vector<vector<int>>& matrix) {
 reverse(matrix.begin(), matrix.end());
 for(int i = 0; i < matrix.size()-1; i++)
 for(int j = i + 1; j < matrix.size(); j++)
 swap(matrix[i][j], matrix[j][i]);
 }
};
```

# Maximum Subarray(53)

## Question

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array `[-2, 1, -3, 4, -1, 2, 1, -5, 4]`, the contiguous subarray `[4, -1, 2, 1]` has the largest sum = `6`.

More practice:

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

## Solution

```
class Solution {
public:
 int maxSubArray(vector<int>& nums) {
 if(nums.size() == 0) return 0;
 int max = nums[0], sum = nums[0];
 for(int i = 1; i < nums.size(); i++){
 if(sum < 0) sum = nums[i];
 else sum += nums[i];
 if(sum > max) max = sum;
 }
 return max;
 }
};
```

## Spiral Matrix(54)

### Question

Given a matrix of  $m \times n$  elements ( $m$  rows,  $n$  columns), return all elements of the matrix in spiral order.

For example, Given the following matrix:

```
[1, 2, 3],
[4, 5, 6],
[7, 8, 9]
]
```

You should return `[1, 2, 3, 6, 9, 8, 7, 4, 5]` .



## Jump Game(55)

### Question

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

For example:

```
A = [2,3,1,1,4], return true.
```

```
A = [3,2,1,0,4], return false.
```

### Solution

```
class Solution {
public:

 bool helper(vector<int>& nums, int right){
 int newright = -1;
 for(int i = right-1; i >=0; i--){
 if(nums[i] + i >= right){
 newright = i;
 break;
 }
 }
 if(newright == 0) return true;
 if(newright != -1) return helper(nums, newright);
 return false;
 }

 bool canJump(vector<int>& nums) {
 if(nums.size() <= 1) return true;
 return helper(nums, nums.size()-1);
 }
};
```

## Merge Intervals(56)

### Question

Given a collection of intervals, merge all overlapping intervals.

For example,

```
Given [1,3],[2,6],[8,10],[15,18],
return [1,6],[8,10],[15,18].
```

### Solution

```
/**
 * Definition for an interval.
 * struct Interval {
 * int start;
 * int end;
 * Interval() : start(0), end(0) {}
 * Interval(int s, int e) : start(s), end(e) {}
 * };
 */
class Solution {
public:
 bool static compare(Interval i1, Interval i2){
 return i1.start == i2.start ?
 i1.end < i2.end:i1.start < i2.start;
 }

 vector<Interval> merge(vector<Interval>& intervals) {
 if(intervals.size() == 0) return {};
 sort(intervals.begin(), intervals.end(), compare);
 vector<Interval> rtn = {intervals[0]};
 for(int i = 1; i < intervals.size(); i++){
 if(intervals[i].start > rtn.rbegin()->end)
 rtn.push_back(intervals[i]);
 else
 rtn.rbegin()->end = max(rtn.rbegin()->end,
 intervals[i].end);
 }

 return rtn;
 }
};
```

## Unique Paths(62)

### Question

A robot is located at the top-left corner of a  $m \times n$  grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?



Above is a  $3 \times 7$  grid. How many possible unique paths are there?

**Note:**  $m$  and  $n$  will be at most 100.

### Solution

```
class Solution {
public:
 int uniquePaths(int m, int n) {
 if(m == 0 || n == 0) return 0;
 vector<int> temp = {}; vector<vector<int>> table = {};
 for(int i = 0; i < n; i++) temp.push_back(1);
 for(int i = 0; i < m; i++) table.push_back(temp);

 for(int i = 1; i < m; i++)
 for(int j = 1; j < n; j++)
 table[i][j] = table[i-1][j] + table[i][j-1];

 return table[m-1][n-1];
 }
};
```

## Unique Paths II(63)

### Question

Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

For example, There is one obstacle in the middle of a 3x3 grid as illustrated below.

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

The total number of unique paths is 2.

**Note:** m and n will be at most 100.

### Solution

```

class Solution {
public:
 int uniquePathsWithObstacles(
 vector<vector<int>>& obstacleGrid) {
 if((obstacleGrid.size() == 0 ||
 obstacleGrid[0].size() == 0) ||
 (obstacleGrid[0][0] == 1) ||
 (obstacleGrid[obstacleGrid.size()-1]
 [obstacleGrid[0].size()] == 1))
 return 0;
 obstacleGrid[0][0] = 1;

 for(int v = 1; v < obstacleGrid[0].size(); v++)
 obstacleGrid[0][v] = (obstacleGrid[0][v] != 1 &&
 obstacleGrid[0][v-1] != 0)?
 1:0;

 for(int h = 1; h < obstacleGrid.size(); h++)
 obstacleGrid[h][0] = (obstacleGrid[h][0] != 1 &&
 obstacleGrid[h-1][0] != 0)?
 1:0;

 for(int v = 1; v < obstacleGrid.size(); v++)
 for(int h = 1; h < obstacleGrid[1].size(); h++)
 obstacleGrid[v][h] = (obstacleGrid[v][h] != 1)?
 obstacleGrid[v-1][h] +
 obstacleGrid[v][h-1]:
 0;

 return obstacleGrid[obstacleGrid.size()-1]
 [obstacleGrid[0].size()-1];
 }
};

```



# Minimum Path Sum(64)

## Question

Given a  $m \times n$  grid filled with non-negative numbers, find a path from top left to bottom right which minimizes the sum of all numbers along its path.

**Note:** You can only move either down or right at any point in time.

## Solution

```
class Solution {
public:
 int minPathSum(vector<vector<int>>& grid) {
 if(grid.size() == 0 || grid[0].size() == 0) return 0;

 for(int v = 1; v < grid.size(); v++)
 grid[v][0] += grid[v-1][0];
 for(int h = 1; h < grid[0].size(); h++)
 grid[0][h] += grid[0][h-1];

 for(int v = 1; v < grid.size(); v++)
 for(int h = 1; h < grid[0].size(); h++)
 grid[v][h] += min(grid[v-1][h], grid[v][h-1]);

 return grid.back().back();
 }
};
```

## Plus One(66)

### Question

Given a non-negative number represented as an array of digits, plus one to the number.

The digits are stored such that the most significant digit is at the head of the list.

### Solution

```
class Solution {
public:
 vector<int> plusOne(vector<int>& digits) {
 if(digits.size() == 0) return digits;
 int carry = 1;
 for(int i = digits.size()-1; i >= 0; i--){
 int sum = digits[i] + carry;
 carry = sum/10;
 digits[i] = sum%10;
 }
 if(carry) digits.insert(digits.begin(), carry);
 return digits;
 }
};
```

## Set Matrix Zeroes(73)

### Question

Given a  $m \times n$  matrix, if an element is 0, set its entire row and column to 0. Do it in place.

Follow up: Did you use extra space? A straight forward solution using  $O(mn)$  space is probably a bad idea. A simple improvement uses  $O(m + n)$  space, but still not the best solution. Could you devise a constant space solution?

### Solution

```
class Solution {
public:
 void setZeroes(vector<vector<int>>& matrix) {
 if(matrix.size() == 0 || matrix[0].size() == 0) return;
 vector<int> v, h;
 v.resize(matrix.size(), 0);
 h.resize(matrix[0].size(), 0);

 for(int i = 0; i < matrix.size(); i++)
 for(int j = 0; j < matrix[0].size(); j++)
 if(matrix[i][j] == 0){ v[i] = 1; h[j] = 1;}

 for(int i = 0; i < v.size(); i++)
 if(v[i] == 1)
 for(int j = 0; j < matrix[0].size(); j++)
 matrix[i][j] = 0;

 for(int i = 0; i < h.size(); i++)
 if(h[i] == 1)
 for(int j = 0; j < matrix.size(); j++)
 matrix[j][i] = 0;

 return;
 }
};
```

## Search a 2D Matrix(74)

### Question

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

Integers in each row are sorted from left to right. The first integer of each row is greater than the last integer of the previous row. For example,

Consider the following matrix:

```
[
 [1, 3, 5, 7],
 [10, 11, 16, 20],
 [23, 30, 34, 50]
]
```

Given target = 3 , return true .

### Solution

```
class Solution {
public:
 bool helper(vector<vector<int>>& matrix, int target, int rmin,
 int rmax, int cmin, int cmax){
 if(rmin > rmax || cmin > cmax) return false;
 if(rmin == rmax && cmin == cmax)
 return matrix[rmin][cmin] == target;
 int rmid = (rmin + rmax)/2, cmid = (cmin + cmax)/2;

 if(rmax - rmin == 1){
 return helper(matrix, target, rmin, rmin, cmin, cmax)||
 helper(matrix, target, rmax, rmax, cmin, cmax);
 }
 }
}
```

```
 if(matrix[rmid][cmid] > target) {
 if(rmin == rmax)
 return helper(matrix, target, rmin, rmid,
 cmin, cmid-1);
 else
 return helper(matrix, target, rmin, rmid,
 cmin, cmax);
 }else if(matrix[rmid][cmid] == target){
 return true;
 }else{
 if(rmin == rmax)
 return helper(matrix, target, rmid, rmax,
 cmid+1, cmax);
 else
 return helper(matrix, target, rmid, rmax,
 cmin, cmax);
 }

 return false;
 }

 bool searchMatrix(vector<vector<int>>& matrix, int target) {
 if(matrix.size() == 0 || matrix[0].size() == 0)
 return false;
 return
 helper(matrix, target, 0, matrix.size()-1, 0,
 matrix[0].size()-1);
 }
};
```

## Sort Colors(75)

### Question

Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

**Note:** You are not suppose to use the library's sort function for this problem.

Follow up: A rather straight forward solution is a two-pass algorithm using counting sort. First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.

Could you come up with an one-pass algorithm using only constant space?

link: <https://leetcode.com/problems/sort-colors/>

# Subsets(78)

## Question

Given a set of distinct integers, `nums`, return all possible subsets.

**Note:**

- Elements in a subset must be in non-descending order.
- The solution set must not contain duplicate subsets.

For example, If `nums = [1, 2, 3]` , a solution is:

```
[
 [3],
 [1],
 [2],
 [1, 2, 3],
 [1, 3],
 [2, 3],
 [1, 2],
 []
]
```

## Solution



```
class Solution {
public:
 vector<int> helper(vector<int>& nums, int magic){
 vector<int> rtn = {};
 for(int i = 0; i < nums.size(); i++)
 if(((1<<i) & magic) != 0) rtn.push_back(nums[i]);
 return rtn;
 }

 vector<vector<int> > subsets(vector<int>& nums) {
 sort(nums.begin(), nums.end());

 vector<vector<int> > rtn = {};
 for(int i = 0; i < (1 << nums.size()); i++)
 rtn.push_back(helper(nums, i));

 return rtn;
 }
};
```

# Word Search(79)

## Question

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example, Given **board** =

```
[
 ['A','B','C','E'],
 ['S','F','C','S'],
 ['A','D','E','E']
]
word = "ABCCED", -> returns true,
word = "SEE", -> returns true,
word = "ABCB", -> returns false.
```

## Solution

```
class Solution {
public:

 bool exist(vector<vector<char>>& board, string word, int row,
 int col, int pos){
 if(pos >= word.size()) return true;
 bool exists = false;

 if(row - 1 >= 0 &&
 board[row-1][col] == word[pos]){//up
 char k = board[row-1][col];
 board[row-1][col] = '-';
```

```

 exists = exist(board, word, row-1, col, pos+1);
 board[row-1][col] = k;
 }

 if(!exists && col-1 >= 0 &&
 board[row][col-1] == word[pos]){ //left
 char k = board[row][col-1];
 board[row][col-1] = '-';
 exists = exist(board, word, row, col-1, pos+1);
 board[row][col-1] = k;
 }

 if(!exists && (row+1) < board.size() &&
 board[row+1][col] == word[pos]){ //down
 char k = board[row+1][col];
 board[row+1][col] = '-';
 exists = exist(board, word, row+1, col, pos+1);
 board[row+1][col] = k;
 }

 if(!exists && col+1 < board[0].size() &&
 board[row][col+1] == word[pos]){ //right
 char k = board[row][col+1];
 board[row][col+1] = '-';
 exists = exist(board, word, row, col+1, pos+1);
 board[row][col+1] = k;
 }
 return exists;
}

bool exist(vector<vector<char>>& board, string word) {
 bool exists = false;
 for(int i = 0; i < board.size() && exists == false; i++){
 for(int j = 0; j < board[0].size() &&
 exists == false; j++){
 if(board[i][j] == word[0]){
 char k = board[i][j];
 board[i][j] = '-';
 exists = exist(board, word, i, j, 1);
 if(exists) return exists;
 }
 }
 }
 return false;
}

```

```
 board[i][j] = k;
 }
}

return exists;
}

};
```

# Remove Duplicates from Sorted Array II(80)

## Question

Follow up for "Remove Duplicates": What if duplicates are allowed at most twice?

For example, Given sorted array `nums = [1, 1, 1, 2, 2, 3]`,

Your function should return `length = 5`, with the first five elements of `nums` being `1, 1, 2, 2` and `3`. It doesn't matter what you leave beyond the new length.

## Solution

```
class Solution {
public:
 int removeDuplicates(vector<int>& nums) {
 if(nums.size() <= 2) return nums.size();
 vector<int> rtn = {nums[0], nums[1]};
 int pos = 2;
 while(pos < nums.size()){
 if(!((nums[pos] == nums[pos-1]) &&
 nums[pos] == nums[pos-2]))
 rtn.push_back(nums[pos]);
 pos++;
 }
 nums = rtn;
 return rtn.size();
 }
};
```

# Merge Sorted Array(88)

## Question

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

**Note:** You may assume that nums1 has enough space (size that is greater or equal to  $m + n$ ) to hold additional elements from nums2. The number of elements initialized in nums1 and nums2 are m and n respectively.

## Solution

```
class Solution {
public:
 void merge(vector<int>& nums1, int m,
 vector<int>& nums2, int n) {
 if(nums2.size() == 0) return;
 int pos = m+n-1;
 m--;n--;
 while(n >= 0)
 nums1[pos--] = (m >= 0 && nums1[m] > nums2[n]) ?
 nums1[m--]:nums2[n--];
 }
};
```

## Subsets II(90)

### Question

Given a collection of integers that might contain duplicates, `nums`, return all possible subsets.

**Note:**

- Elements in a subset must be in non-descending order.
- The solution set must not contain duplicate subsets.

For example, If **nums** = `[1, 2, 2]` , a solution is:

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

### Solution

```
class Solution {
private:
 void helper(vector<int>& nums, vector<vector<int> > &rtn,
 vector<int>& adds_on, int start){
 rtn.push_back(adds_on);
 for(int i = start; i < nums.size(); i++){
 adds_on.push_back(nums[i]);
 if(i == start || nums[i] != nums[i-1])
 helper(nums, rtn, adds_on, i+1);
 adds_on.pop_back();
 }
 }

public:
 vector<vector<int> > subsetsWithDup(vector<int>& nums) {
 sort(nums.begin(), nums.end());

 vector<vector<int> > rtn = {{}};
 vector<int> adds_on = {};

 for(int i = 0; i < nums.size(); i++){
 adds_on.push_back(nums[i]);
 if(i == 0 || nums[i] != nums[i-1])
 helper(nums, rtn, adds_on, i+1);
 adds_on.pop_back();
 }
 return rtn;
 }
};
```



# Construct Binary Tree from Preorder and Inorder Traversal(105)

## Question

Given preorder and inorder traversal of a tree, construct the binary tree.

**Note:** You may assume that duplicates do not exist in the tree.

## Solution

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:

 TreeNode* helper(vector<int>& preorder, vector<int>& inorder,
 int pstart, int pend, int istart, int iend){
 if(preorder.size() == 0 || pend < pstart || iend < istart)
 return NULL;

 int data = preorder[pstart];
 TreeNode* t = new TreeNode(data);
 if(pstart == pend) return t;

 int inorderhead_pos = istart;
 while(inorder[inorderhead_pos] != data) inorderhead_pos++;

 t->left = helper(preorder, inorder, pstart+1,
```

```
 pstart+1 + (inorderhead_pos-1 - istart),
 istart, inorderhead_pos-1);
 t->right = helper(preorder, inorder,
 pstart+1+(inorderhead_pos-1-istart)+1,
 pend, inorderhead_pos+1, iend);

 return t;
}

TreeNode* buildTree(vector<int>& preorder,
 vector<int>& inorder) {
 return helper(preorder, inorder, 0, preorder.size()-1,
 0, inorder.size()-1);
}

};
```

# Construct Binary Tree from Inorder and Postorder Traversal(106)

## Question

Given inorder and postorder traversal of a tree, construct the binary tree.

**Note:** You may assume that duplicates do not exist in the tree.

## Solution

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:

 TreeNode* helper(vector<int>& inorder, vector<int>& postorder,
 int pstart, int pend, int istart, int iend) {

 if(iend < istart || pend < pstart) return NULL;

 TreeNode* t = new TreeNode(postorder[pend]);
 if(iend - istart == 0) return t;

 int i_pos = istart;
 while(inorder[i_pos] != postorder[pend]) i_pos++;
 int dis = i_pos - istart;

 t->left = helper(inorder, postorder, pstart,
```

```
 pstart + dis-1, istart,
 i_pos - 1);
 t->right = helper(inorder, postorder, pstart + dis,
 pend -1, istart + dis + 1, iend);

 return t;
}

TreeNode* buildTree(vector<int>& inorder,
 vector<int>& postorder) {
 return helper(inorder, postorder, 0, inorder.size()-1, 0,
 postorder.size()-1);
}

};
```

## Pascal's Triangle(118)

### Question

Given numRows, generate the first numRows of Pascal's triangle.

For example, given numRows = 5 , Return

```
[
 [1],
 [1,1],
 [1,2,1],
 [1,3,3,1],
 [1,4,6,4,1]
]
```

### Solution

```
class Solution {
public:
 vector<vector<int>> generate(int numRows) {
 if(numRows == 0) return {};
 vector<vector<int>> rtn = {{1}};

 for(int i = 1; i < numRows; i++){
 vector<int> temp = {1};
 for(int j = 1; j < rtn.back().size(); j++){
 temp.push_back(rtn.back()[j] + rtn.back()[j-1]);
 }
 temp.push_back(1);
 rtn.push_back(temp);
 }

 return rtn;
 }
};
```

# Pascal's Triangle II(119)

## Question

Given an index k, return the kth row of the Pascal's triangle.

For example, given k = 3 , Return [1, 3, 3, 1] .

**Note:** Could you optimize your algorithm to use only O(k) extra space?

## Solution

```
class Solution {
public:
 vector<int> getRow(int rowIndex) {
 queue<vector<int> > rtn;
 rtn.push({1});
 for(int i = 1; i <= rowIndex; i++){
 vector<int> temp = {1};
 vector<int> prev = rtn.front();
 rtn.pop();
 for(int j = 1; j < prev.size(); j++){
 temp.push_back(prev[j] + prev[j-1]);
 }
 temp.push_back(1);
 rtn.push(temp);
 }

 return rtn.front();
 }
};
```

# Maximum Product Subarray(152)

## Question

Find the contiguous subarray within an array (containing at least one number) which has the largest product.

For example, given the array `[2, 3, -2, 4]`, the contiguous subarray `[2, 3]` has the largest product = `6`.

## Solution

```
class Solution {
public:
 int maxProduct(vector<int>& nums) {
 if(nums.empty()) return 0;
 int global_max = nums[0], local_max = nums[0],
 local_min = nums[0];

 for(int i = 1; i < nums.size(); i++) {
 int temp_min = local_min, temp_max = local_max;
 local_min = min(nums[i], min(nums[i] * temp_min,
 nums[i] * temp_max));
 local_max = max(nums[i], max(nums[i] * temp_min,
 nums[i] * temp_max));
 global_max = max(global_max, local_max);
 }

 return global_max;
 }
};
```



# Find Minimum in Rotated Sorted Array(153)

## Question

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2` ).

Find the minimum element.

You may assume no duplicate exists in the array.

## Solution

```
class Solution {
public:
 int helper(vector<int>& nums, int start, int end){
 if(nums.empty()) return 0;
 if(start == end) return nums[start];

 int mid = (start + end)/2;
 if(nums[mid] > nums[end]) return helper(nums, mid+1, end);
 return helper(nums, start, mid);
 }

 int findMin(vector<int>& nums) {
 return helper(nums, 0, nums.size()-1);
 }
};
```

## Find Minimum in Rotated Sorted Array II(154)

Follow up for "Find Minimum in Rotated Sorted Array":  
What if duplicates are allowed?  
Would this affect the run-time complexity? How and why?

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2` ).

Find the minimum element.

The array may contain duplicates.

### Solution

```
class Solution {
public:
 int helper(vector<int>& nums, int start, int end){
 if(nums.empty()) return 0;
 if(start == end) return nums[start];

 int mid = (start + end)/2;
 if(nums[mid] == nums[end])
 return min(helper(nums, start, mid),
 helper(nums, mid+1, end));
 if(nums[mid] > nums[end])
 return helper(nums, mid+1, end);
 return helper(nums, start, mid);
 }

 int findMin(vector<int>& nums) {
 return helper(nums, 0, nums.size()-1);
 }
};
```

# Majority Element(169)

## Question

Given an array of size  $n$ , find the majority element. The majority element is the element that appears more than  $\lfloor n/2 \rfloor$  times.

You may assume that the array is non-empty and the majority element always exist in the array.

## Solution

```
class Solution {
public:
 int majorityElement(vector<int>& nums) {
 int num_size = nums.size();
 if(num_size == 1) return nums[0];
 int counter = 0, candidate = nums[0];

 for(int n : nums){
 if(counter == 0){
 counter++;
 candidate = n;
 }else if(n == candidate) counter++;
 else counter--;
 }
 return candidate;
 }
};
```

## Combination Sum III(216)

### Question

Find all possible combinations of k numbers that add up to a number n, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

Ensure that numbers within the set are sorted in ascending order.

#### Example 1:

Input: k = 3, n = 7

Output:

```
[[1, 2, 4]]
```

#### Example 2:

Input: k = 3, n = 9

Output:

```
[[1, 2, 6], [1, 3, 5], [2, 3, 4]]
```

### Solution

```
class Solution {
public:
 vector<vector<int>> helper(int k, int n, vector<int> nums){
 vector<vector<int>> rtn = {};
 if(k == 0 || nums.size() < k) return rtn;
 if(n > nums.back() * k || n < k * nums[0]) return rtn;
 for(int i = 0; i < nums.size();){
 int num = nums[i];
 if(k == 1 && num == n) return {{num}};
 // if(k == 1) i++;
 nums.erase(nums.begin());
 vector<vector<int>> temp = helper(k-1, n-num, nums);
 for(int j = 0; j < temp.size(); j++)
 temp[j].insert(temp[j].begin(), num);
 rtn.insert(rtn.end(), temp.begin(), temp.end());
 }

 return rtn;
 }

 vector<vector<int>> combinationSum3(int k, int n) {
 vector<int> nums = {1, 2, 3, 4, 5, 6, 7, 8, 9};
 return helper(k, n, nums);
 }
};
```

# Contains Duplicate(217)

## Question

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

## Solution

```
class Solution {
public:
 bool containsDuplicate(vector<int>& nums) {
 if(nums.size()<=1) return false;
 unordered_set<int> m;

 for(int i = 0; i < nums.size(); i++){
 int old_size = m.size();
 if(m.insert(nums[i]).second == false) return true;
 }
 return false;
 }
};
```

# Contains Duplicate II(219)

## Question

Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that `nums[i] = nums[j]` and the difference between i and j is at most k.

## Solution

```
class Solution {
public:
 bool containsNearbyDuplicate(vector<int>& nums, int k) {
 if(nums.size() == 0 || k == 0) return false;

 unordered_map<int, int> map;

 for(int i = 0; i < nums.size(); i++){
 auto range = map.equal_range(nums[i]);
 for(auto it = range.first; it != range.second; ++it){
 if(abs(it->second - i) <= k) return true;
 }
 map[nums[i]] = i;
 }

 return false;
 }
};
```



## Median of Two Sorted Arrays(4)

### Question

There are two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively. Find the median of the two sorted arrays. The overall run time complexity should be

`$O(\log(m+n))$`  .

# Summary Ranges(228)

## Question

Given a sorted integer array without duplicates, return the summary of its ranges.

For example, given `[0,1,2,4,5,7]` , return `["0->2", "4->5", "7"]` .

## Solution

```
class Solution {
public:
 vector<string> summaryRanges(vector<int>& nums) {
 if(nums.empty()) return {};
 vector<string> rtn;
 int start = 0, last = 0, n_size = nums.size();

 for(int i = 1; i < n_size; i++){
 if(abs(nums[i] - nums[last]) <= 1){
 last++;
 }else{
 rtn.push_back(to_string(nums[start]) +
 (last == start ?
 "" :
 ("->" + to_string(nums[last]))));
 start = i; last = i;
 }
 }

 rtn.push_back(to_string(nums[start]) +
 (last == start ?
 "" :
 ("->" + to_string(nums[last]))));

 return rtn;
 }
};
```



# Find Peak Element(162)

## Question

A peak element is an element that is greater than its neighbors.

Given an input array where  $\text{num}[i] \neq \text{num}[i+1]$  , find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that  $\text{num}[-1] = \text{num}[n] = -\infty$  .

For example, in array  $[1, 2, 3, 1]$  , 3 is a peak element and your function should return the index number 2.

**Note:** Your solution should be in logarithmic complexity.

## Solution

```
class Solution {
public:
 int findPeakElement(vector<int>& nums) {
 int n_size = nums.size(), start = 0, end = n_size-1;
 if(n_size <= 1 || (n_size>=2 && nums[0] > nums[1]))
 return 0;
 while (start < end) {
 if(start == end) return start;
 int mid = (start + end)/2;
 if (nums[mid] > nums[mid+1] &&
 nums[mid] > nums[mid-1])
 return mid;
 else nums[mid+1] > nums[mid-1] ?
 start = mid+1: end = mid;
 }
 return nums[n_size - 1] > nums[n_size - 2]?
 n_size-1:0;
 }
};
```

# Longest Increasing Path in a Matrix(329)

## Question

Given an integer matrix, find the length of the longest increasing path.

From each cell, you can either move to four directions: left, right, up or down. You may NOT move diagonally or move outside of the boundary (i.e. wrap-around is not allowed).

Example 1:

```
nums = [
 [9,9,4],
 [6,6,8],
 [2,1,1]
]
```

Return **4**

The longest increasing path is `[1, 2, 6, 9]` .

Example 2:

```
nums = [
 [3,4,5],
 [3,2,6],
 [2,2,1]
]
```

Return **4**

The longest increasing path is `[3, 4, 5, 6]` . Moving diagonally is not allowed.

## Solution

```

class Solution {
public:
 vector<int>* m;

 int helper(vector<vector<int>>& matrix, int row, int col){
 if ((*m)[row*matrix[0].size() + col] != -1)
 return (*m)[row*matrix[0].size() + col];
 int rtn = 0;

 int up = -1, down = -1, left = -1, right = -1;

 if (row > 0 && matrix[row][col] < matrix[row-1][col]) //up
 rtn = max(rtn, helper(matrix, row-1, col));

 if(row < matrix.size() - 1 &&
 matrix[row][col] < matrix[row+1][col]) // down
 rtn = max(rtn, helper(matrix, row+1, col));

 if(col > 0 &&
 matrix[row][col] < matrix[row][col - 1]) // left
 rtn = max(rtn, helper(matrix, row, col-1));

 if(col < matrix[0].size() - 1 &&
 matrix[row][col] < matrix[row][col+1]) //right
 rtn = max(rtn, helper(matrix, row, col+1));

 return (*m)[row * matrix[0].size() + col] = 1 + rtn;
 }

 int longestIncreasingPath(vector<vector<int>>& matrix) {
 if(matrix.empty() || matrix.back().empty()) return 0;
 m = new vector<int>(matrix.size() * matrix[0].size(), -1);
 int m_val = -1;
 for(int i = 0; i < matrix.size(); i++)
 for(int j = 0; j < matrix.back().size(); j++)
 m_val = max(m_val,
 (*m)[i*matrix[0].size() + j] == -1 ?
 helper(matrix, i, j) :
 (*m)[i*matrix[0].size() + j]);
 }
}

```

```
 return m_val;
 }
};
```



# Game of Life(289)

## Question

According to the Wikipedia's article: "The `Game of Life` , also known simply as `Life` , is a cellular automaton devised by the British mathematician John Horton Conway in 1970."

Given a board with  $m$  by  $n$  cells, each cell has an initial state live (1) or dead (0). Each cell interacts with its eight neighbors (horizontal, vertical, diagonal) using the following four rules (taken from the above Wikipedia article):

- Any live cell with fewer than two live neighbors dies, as if caused by under-population.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by over-population..
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.
- Write a function to compute the next state (after one update) of the board given its current state.

### Follow up:

- Could you solve it in-place? Remember that the board needs to be updated at the same time: You cannot update some cells first and then use their updated values to update other cells.
- In this question, we represent the board using a 2D array. In principle, the board is infinite, which would cause problems when the active area encroaches the border of the array. How would you address these problems?

## Solution

```

class Solution {
public:
 int compute_live(vector<vector<int>>& board, int row, int col) {
 int sum = 0;
 int row_max = board.size()-1;
 int col_max = board[0].size()-1;
 if(row > 0) sum += board[row-1][col]; //up
 if(row < row_max) sum += board[row+1][col]; //down
 if(col > 0) sum += board[row][col-1]; //left
 if(col < col_max) sum += board[row][col+1]; //right
 if(row > 0 && col > 0)
 sum += board[row-1][col-1]; //up-left
 if(row > 0 && col < col_max)
 sum += board[row-1][col+1]; //up-right
 if(row < row_max && col > 0)
 sum += board[row+1][col-1]; //down-left
 if(row < row_max && col < col_max)
 sum += board[row+1][col+1]; //down-right
 return sum;
 }

 void gameOfLife(vector<vector<int>>& board) {
 vector<int> line(board[0].size(), 0);
 int row_max = board.size()-1;
 int col_max = board[0].size()-1;
 vector<vector<int>> b_copy(board.size(), line);
 for (int i = 0; i <= row_max; i++) {
 for (int j = 0; j <= col_max; j++) {
 int sum = compute_live(board, i, j);
 if(board[i][j])
 b_copy[i][j] = (sum < 2 || sum > 3) ? 0 : 1;
 else b_copy[i][j] = (sum == 3) ? 1 : 0;
 }
 }
 board = b_copy;
 }
};

```



## H-Index(274)

### Question

Given an array of citations (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia: "A scientist has index  $h$  if  $h$  of his/her  $N$  papers have **at least**  $h$  citations each, and the other  $N - h$  papers have **no more than**  $h$  citations each."

For example, given `citations = [3, 0, 6, 1, 5]`, which means the researcher has `5` papers in total and each of them had received `3, 0, 6, 1, 5` citations respectively. Since the researcher has `3` papers with at least `3` citations each and the remaining two with no more than `3` citations each, his h-index is `3`.

**Note:** If there are several possible values for  $h$ , the maximum one is taken as the h-index.

#### Hint:

- An easy approach is to sort the array first.
- What are the possible values of h-index?
- A faster approach is to use extra space.

### Solution

```
class Solution {
public:
 int hIndex(vector<int>& citations) {
 if (citations.empty()) return 0;
 int c_size = citations.size(),
 rtn = 0, map[c_size+1];
 for (int i = 0; i <= c_size; i++) map[i] = 0;

 for (int i = 0; i < c_size; i++)
 (citations[i] >= c_size) ?
 map[c_size] += 1 : map[citations[i]] += 1;

 for (int i = c_size; i >= 0; i--)
 if ((rtn += map[i]) >= i) return min(i, rtn);

 return rtn;
 }
};
```

## Search a 2D Matrix II(240)

### Question

Write an efficient algorithm that searches for a value in an `m x n` 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. For example,

Consider the following 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]
]
```

Given `target = 5` , return `true` .

Given `target = 20` , return `false` .

### Solution

```
class Solution {
public:
 bool searchMatrix(vector<vector<int>>& matrix, int target) {
 if (matrix.empty() || matrix.back().empty() ||
 target < matrix[0][0] ||
 target > matrix.back().back())
 return false;

 int row_size = matrix.size(),
 col_size = matrix.back().size(),
 i = 0, j = col_size - 1;

 while (i < row_size && j >= 0) {
 if (target == matrix[i][j]) return true;
 else if (target > matrix[i][j]) i++;
 else j--;
 }
 return false;
 }
};
```

# Reconstruct Itinerary(332)

## Question

Given a list of airline tickets represented by pairs of departure and arrival airports `[from, to]` , reconstruct the itinerary in order. All of the tickets belong to a man who departs from `JFK` . Thus, the itinerary must begin with `JFK` .

### Note:

- If there are multiple valid itineraries, you should return the itinerary that has the smallest lexical order when read as a single string. For example, the itinerary `["JFK", "LGA"]` has a smaller lexical order than `["JFK", "LGB"]` .
- All airports are represented by three capital letters (IATA code).
- You may assume all tickets form at least one valid itinerary.

**Example 1:** tickets = `[["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]`

Return `["JFK", "MUC", "LHR", "SFO", "SJC"]` .

**Example 2:** tickets = `[["JFK", "SFO"], ["JFK", "ATL"], ["SFO", "ATL"], ["ATL", "JFK"], ["ATL", "SFO"]]` Return

`["JFK", "ATL", "JFK", "SFO", "ATL", "SFO"]` . Another possible reconstruction is `["JFK", "SFO", "ATL", "JFK", "ATL", "SFO"]` . But it is larger in lexical order.

## Solution

```
class Solution {
public:
 vector<string> recur(unordered_map<string,
 vector<string>>& map,
 string& start, int exp_size){
 vector<string> rtn = {};
 if(exp_size == 0) return {start};
```



```

 if (map.find(start) == map.end()) return {};

 // so, at this point, map.size() != 0 && we can find it
 for (string& record: map.find(start)->second) {
 if (record.size() != 3) continue;
 string rec = record;
 record += '?';
 rtn = recur(map, rec, exp_size-1);
 if (rtn.size() != 0 || exp_size == 0){
 rtn.insert(rtn.begin(), start);
 return rtn;
 }
 record.pop_back();
 }
 return rtn;
 }

 vector<string> findItinerary(vector<pair<string,
 string>> tickets) {
 unordered_map<string, vector<string>> map;

 vector<string> rtn = {};
 for (pair<string, string> p : tickets) {
 auto it = map.find(p.first);
 if (it != map.end()) it->second.push_back(p.second);
 else map[p.first] = {p.second};
 }

 string it = "JFK";

 for (auto& it_sort : map)
 sort(it_sort.second.begin(), it_sort.second.end());

 rtn = recur(map, it, tickets.size());

 return rtn;
 }
};

```

# Count of Range Sum(327)

## Question

Given an integer array `nums`, return the number of range sums that lie in `[lower, upper]` inclusive. Range sum `S(i, j)` is defined as the sum of the elements in `nums` between indices `i` and `j` ( $i \leq j$ ), inclusive.

Note: A naive algorithm of  $O(n^2)$  is trivial. You MUST do better than that.

Example: Given `nums = [-2, 5, -1]`, `lower = -2`, `upper = 2`, Return `3`.  
The three ranges are : `[0, 0]`, `[2, 2]`, `[0, 2]` and their respective sums are: `-2`, `-1`, `2`.

## Solution

```

class Solution {
public:
 int mergesort(long long* sum, int left, int right, int lower, int upper) {
 int rtn = 0, mid = (left + right)/2, i = left, k = mid + 1,
 long long temp[right-left+1];

 if (left >= right)
 return (left == right && sum[left] >= lower && sum[left] <= upper);

 rtn = mergesort(sum, left, mid, lower, upper) +
 mergesort(sum, mid+1, right, lower, upper);

 for (int i = left; i <= mid; ++i){
 while(j <= right && sum[j]-sum[i] < lower) ++j;
 while(k <= right && sum[k]-sum[i] <= upper) ++k;
 rtn +=k-j;
 }

 for(i=k=left, j=mid+1; k<=right; ++k)
 temp[k-left] = (i<=mid) && (j>right || sum[i]<sum[j]) ? sum[i++] : sum[j++];
 copy(temp, temp + (right-left+1), sum+left);
 return rtn;
 }

 int countRangeSum(vector<int>& nums, int lower, int upper) {
 int nums_size = nums.size(); long long sum[nums_size+1]; sum[0] = 0;
 for (int i = 1; i <= nums_size; ++i) sum[i] = sum[i-1] + nums[i-1];
 return mergesort(sum, 1, nums_size, lower, upper);
 }
};

```

# Insert Interval(57)

## Question

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

Example 1: Given intervals `[1, 3], [6, 9]` , insert and merge `[2, 5]` in as `[1, 5], [6, 9]` .

Example 2: Given `[1, 2], [3, 5], [6, 7], [8, 10], [12, 16]` , insert and merge `[4, 9]` in as `[1, 2], [3, 10], [12, 16]` .

This is because the new interval `[4, 9]` overlaps with `[3, 5], [6, 7], [8, 10]` .

## Solution

```

class Solution {
public:
 vector<Interval> insert(vector<Interval>& intervals, Interval r
 if(intervals.size() == 0) return {newInterval};
 int int_size = intervals.size(), i = 0;
 vector<Interval> rtn = {};
 bool inserted = false;
 for (i = 0; i < int_size; ++i) {
 if (!inserted) {
 if (intervals[i].end < newInterval.start) rtn.push_
 else if((intervals[i].start <= newInterval.start &
 (intervals[i].start >= newInterval.start &
 intervals[i].end >= newInterval.end)){
 newInterval.start = min(intervals[i].start, new
 newInterval.end = max(intervals[i].end, new
 rtn.push_back(newInterval);
 inserted = true;
 }else if(newInterval.end < intervals[i].start){
 rtn.push_back(newInterval);
 inserted = true;
 break;
 }
 }
 else{
 if (intervals[i].start > rtn.back().end) break;
 else if(intervals[i].start > rtn.back().start && in
 else{
 rtn.back().start = min(intervals[i].start, rtn
 rtn.back().end = max(intervals[i].end, rtn
 }
 }
 }
 if(i != int_size) rtn.insert(rtn.end(), intervals.begin()+i
 if(!inserted) rtn.push_back(newInterval);

 return rtn;
 }
};

```

# Best Time to Buy and Sell Stock with Cooldown(309)

## Question

Say you have an array for which the  $i$ th element is the price of a given stock on day  $i$ .

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times) with the following restrictions:

- You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).
- After you sell your stock, you cannot buy stock on next day. (ie, cooldown 1 day)

**Example:**

```
prices = [1, 2, 3, 0, 2]
maxProfit = 3
transactions = [buy, sell, cooldown, buy, sell]
```

## Solution

```
class Solution {
public:
 int maxProfit(vector<int>& prices) {
 int price_size = prices.size();
 if (price_size <= 1) return 0;
 vector<int> buy(price_size, 0), sell(price_size, 0), cool(price_size, 0);
 buy[0] = -prices[0], sell[0] = INT_MIN, cool[0] = 0;
 for (int i = 1; i < price_size; ++i) {
 buy[i] = max(cool[i-1] - prices[i], buy[i-1]);
 sell[i] = buy[i-1] + prices[i];
 cool[i] = max(sell[i-1], cool[i-1]);
 }

 return max(buy[price_size-1], max(sell[price_size-1], cool[price_size-1]));
 }
};
```

# Tree

## Question

You are given an integer array `nums` and you have to return a new counts array. The counts array has the property where `counts[i]` is the number of smaller elements to the right of `nums[i]`.

**Example:**

```
Given nums = [5, 2, 6, 1]
```

```
To the right of 5 there are 2 smaller elements (2 and 1).
```

```
To the right of 2 there is only 1 smaller element (1).
```

```
To the right of 6 there is 1 smaller element (1).
```

```
To the right of 1 there is 0 smaller element.
```

Return the array `[2, 1, 1, 0]`.

## Solution



```

class Solution {
public:
 struct STree {
 int val, lt, dup;
 // lt is the # of inserted numbers that Less Than cur node
 // dup is the same val number
 STree *left, *right;
 STree(int x, int lv = 0, int dv = 0, STree* l = NULL, STree* r = NULL) : val(x), lt(lv), dup(dv), left(l), right(r){}
 };

 int insert(STree &T, int num){
 int rtn = 0;
 if (T.val == num){
 T.dup++; rtn = T.lt;
 }else if (num < T.val){
 T.lt++;
 if (T.left == NULL) T.left = new STree(num, 0, 1);
 else rtn = insert(*T.left, num);
 }else{
 rtn = (T.lt + T.dup);
 if (T.right == NULL) T.right = new STree(num, 0, 1);
 else rtn += insert(*T.right, num);
 }
 return rtn;
 }

 vector<int> countSmaller(vector<int>& nums) {
 if (nums.empty()) return {};
 STree *T = new STree(nums.back(), 0, 1);
 int num_size = nums.size();
 vector<int> rtn(num_size); rtn.back() = 0;
 for (int i = num_size-2; i >=0; --i)
 rtn[i] = insert(*T, nums[i]);
 return rtn;
 }
};

```

# Binary Tree Inorder Traversal(94)

## Question

Given a binary tree, return the inorder traversal of its nodes' values.

For example: Given binary tree `{1, #, 2, 3},`

```
1
 \
 2
 /
3
```

return `[1, 3, 2]` .

## Solution

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 vector<int> inorderTraversal(TreeNode* root) {
 if(root == NULL) return {};
 vector<int> rtn;
 if(root->left != NULL) rtn = inorderTraversal(root->left);

 rtn.push_back(root->val);

 vector<int> right;
 if(root->right != NULL) {
 right = inorderTraversal(root->right);
 rtn.insert(rtn.end(), right.begin(), right.end());
 }

 return rtn;
 }
};
```

# Validate Binary Search Tree(98)

## Question

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left and right subtrees must also be binary search trees.

## Solution

```
class Solution {
public:
 bool myfunc(TreeNode* root, int min, int max,
 bool min_inc, bool max_inc){
 if(root == NULL) return true;

 if((root->val > min || (root->val >= min && min_inc))
 &&(root->val < max || (root->val <= max && max_inc)))
 if(myfunc(root->left, min, root->val,
 !(root->val == min) && min_inc,
 false))
 return myfunc(root->right, root->val,
 max, false,
 !(root->val == max) && max_inc);

 return false;
 }

 bool isValidBST(TreeNode* root) {
 return myfunc(root, INT_MIN, INT_MAX, true, true);
 }
};
```

# Same Tree(100)

## Question

Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

## Solution

```
class Solution {
public:
 bool isSameTree(TreeNode* p, TreeNode* q) {
 if(!p && !q) return true;
 if((!p || !q)) return false;

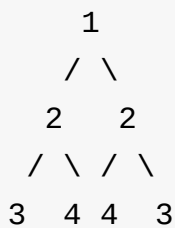
 return (p->val == q->val) &&
 isSameTree(p->left, q->left) &&
 isSameTree(p->right, q->right);
 }
};
```

# Symmetric Tree(101)

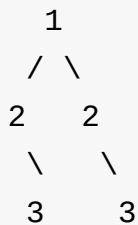
## Question

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree is symmetric:



But the following is not:



## Solution

```
class Solution {
public:
 bool match(TreeNode* l, TreeNode* r){
 if(!l && !r) return true;
 if(!l || !r) return false;
 return (l->val == r->val) && match(l->right, r->left) &&
 match(l->left, r->right);
 }

 bool isSymmetric(TreeNode* root) {
 return root == NULL? true:match(root->left, root->right);
 }
};
```



# Binary Tree Level Order Traversal(102)

## Question

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example: Given binary tree `{3, 9, 20, #, #, 15, 7}` ,

```
 3
 / \
 9 20
 / \
 15 7
```

return its level order traversal as:

```
[
 [3],
 [9, 20],
 [15, 7]
]
```

## Solution

```
class Solution {
public:
 vector<vector<int>> levelOrder(TreeNode* root) {
 if(!root) return {};

 vector<vector<int> > rtn;

 queue<pair<TreeNode*, int> > q;
 q.push({root, 1});

 while(q.size() != 0){
 pair<TreeNode*, int> p = q.front();
 TreeNode* n = p.first;
 q.pop();

 if(n->left != NULL) q.push({n->left, 1 + p.second});
 if(n->right != NULL) q.push({n->right, 1 + p.second});

 while(p.second > rtn.size()) rtn.push_back({});

 rtn[p.second-1].push_back(n->val);

 }

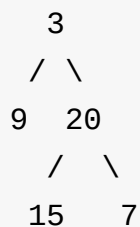
 return rtn;
 }
};
```

# Binary Tree Zigzag Level Order Traversal(103)

## Question

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example: Given binary tree `{3, 9, 20, #, #, 15, 7}` ,



return its zigzag level order traversal as:

```
[
 [3],
 [20, 9],
 [15, 7]
]
```

## Solution

```
class Solution {
public:
 vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
 if(!root) return {};

 vector<vector<int> > rtn;

 queue<pair<TreeNode*, int> > q;
 q.push({root, 1});

 while(q.size() != 0){
 pair<TreeNode*, int> p = q.front();
 TreeNode* n = p.first;
 q.pop();

 if(n->left != NULL) q.push({n->left, 1 + p.second});
 if(n->right != NULL) q.push({n->right, 1 + p.second});

 while(p.second > rtn.size()) rtn.push_back({});
 if(p.second%2 == 0){
 rtn[p.second-1].insert(rtn[p.second-1].begin(),
 n->val);
 }else
 rtn[p.second-1].push_back(n->val);

 }

 return rtn;
 }
};
```

# Maximum Depth of Binary Tree(104)

## Question

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

## Solution

```
class Solution {
public:
 int maxDepth(TreeNode* root) {
 if(root == NULL) return 0;
 return 1 + max(maxDepth(root->left),
 maxDepth(root->right));
 }
};
```

# Construct Binary Tree from Preorder and Inorder Traversal(105)

## Question

Given preorder and inorder traversal of a tree, construct the binary tree.

Note: You may assume that duplicates do not exist in the tree.

## Solution

```
class Solution {
public:
 TreeNode* helper(vector<int>& preorder, vector<int>& inorder,
 int pstart, int pend, int istart, int iend){
 if(preorder.size() == 0 || pend < pstart || iend < istart)
 return NULL;

 int data = preorder[pstart];
 TreeNode* t = new TreeNode(data);
 if(pstart == pend) return t;

 int inorderhead_pos = istart;
 while(inorder[inorderhead_pos] != data)
 inorderhead_pos++;

 t->left = helper(preorder, inorder, pstart+1,
 pstart+1 +
 (inorderhead_pos-1 - istart),
 istart, inorderhead_pos-1);
 t->right = helper(preorder, inorder, pstart + 1 +
 (inorderhead_pos-1 - istart) + 1,
 pend, inorderhead_pos+1, iend);

 return t;
 }

 TreeNode* buildTree(vector<int>& preorder,
 vector<int>& inorder) {
 return helper(preorder, inorder, 0,
 preorder.size()-1, 0, inorder.size()-1);
 }
};
```

## Construct Binary Tree from Inorder and Postorder Traversal ( 106 )

### Question

Given inorder and postorder traversal of a tree, construct the binary tree.

**Note:** You may assume that duplicates do not exist in the tree.

### Solution

```
class Solution { public: TreeNode* helper(vector& inorder, vector& postorder, int
pstart, int pend, int istart, int iend) {
```



```

 if(iend < istart || pend < pstart) return NULL;

 TreeNode* t = new TreeNode(postorder[pend]);
 if(iend - istart == 0) return t;

 int i_pos = istart;
 while(inorder[i_pos] != postorder[pend]) i_pos++;
 int dis = i_pos - istart;

 t->left = helper(inorder, postorder, pstart,
 pstart + dis-1, istart, i_pos - 1);

 t->right = helper(inorder, postorder, pstart + dis,
 pend -1, istart + dis + 1, iend);

 return t;
 }

 TreeNode* buildTree(vector<int>& inorder,
 vector<int>& postorder) {
 return helper(inorder, postorder, 0, inorder.size()-1,
 0, postorder.size()-1);
 }

```

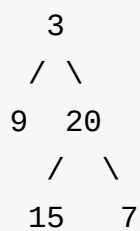
```
};
```

## Binary Tree Level Order Traversal II(107)

### Question

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

For example: Given binary tree {3, 9, 20, #, #, 15, 7} ,



return its bottom-up level order traversal as:

```
[
 [15, 7],
 [9, 20],
 [3]
]
```

### Solution

```
class Solution {
public:
 vector<vector<int>> levelOrderBottom(TreeNode* root) {
 if(root==0) return {};
 queue<pair<int,TreeNode*> > q;
 q.push(make_pair(0,root));
 int lastlevel =0;
 vector<vector<int> > rtn;
 rtn.push_back({});
 while(q.size()!=0){
 pair<int,TreeNode*> p = q.front();q.pop();
 if(lastlevel == p.first)
 rtn[0].push_back(p.second->val);
 else{
 rtn.insert(rtn.begin(),{p.second->val});
 lastlevel = p.first;
 }

 if(p.second->left!=NULL)
 q.push(make_pair(p.first+1,p.second->left));
 if(p.second->right!=NULL)
 q.push(make_pair(p.first+1,p.second->right));

 }
 return rtn;
 }
};
```

# Convert Sorted Array to Binary Search Tree(108)

## Question

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

## Solution

```
class Solution {
public:
 TreeNode* helper(vector<int>& nums, int start, int end){
 if(start > end) return NULL;
 if(start == end) return (new TreeNode(nums[start]));

 int mid = (start + end)/2;

 TreeNode* rtn = new TreeNode(nums[mid]);

 rtn->left = helper(nums, start, mid-1);
 rtn->right = helper(nums, mid+1, end);
 return rtn;
 }

 TreeNode* sortedArrayToBST(vector<int>& nums) {
 if(nums.size() == 0) return NULL;
 if(nums.size() == 1) return new TreeNode(nums[0]);
 return helper(nums, 0, nums.size()-1);
 }
};
```

# Balanced Binary Tree(110)

## Question

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

## Solution

```
class Solution {
public:
 int helper(TreeNode* root){
 if(root == NULL) return 0;
 int left_level = helper(root->left);
 int right_level = helper(root->right);

 return ((left_level >= 0) && (right_level >= 0) &&
 abs(left_level - right_level) <= 1)?
 1+max(left_level, right_level):INT_MIN;
 }

 bool isBalanced(TreeNode* root) {
 if(root == NULL) return true;
 int left_level = helper(root->left);
 int right_level = helper(root->right);
 return (left_level >= 0) && (right_level >= 0) &&
 abs(left_level - right_level) <= 1;
 }
};
```

# Minimum Depth of Binary Tree(111)

## Question

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

## Solution

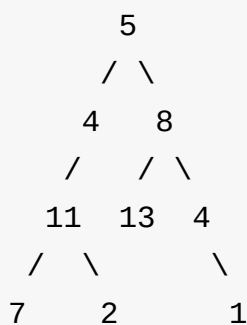
```
class Solution {
public:
 int minDepth(TreeNode* root) {
 if(root == NULL) return 0;
 int l = minDepth(root->left);
 int r = minDepth(root->right);
 if(l + r == 1 || l + r == r) return 1 + l + r;
 return 1+min(l,r);
 }
};
```

# Path Sum(112)

## Question

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

For example: Given the below binary tree and `sum = 22`,



return true, as there exist a root-to-leaf path `5->4->11->2` which sum is 22.

## Solution

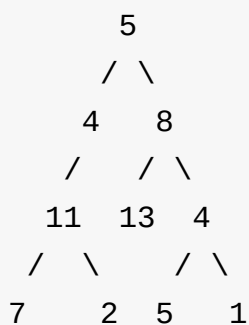
```
class Solution {
public:
 bool hasPathSum(TreeNode* root, int sum) {
 if(root == NULL) return false;
 if(root->left == NULL && root->right == NULL &&
 sum - root->val == 0) return true;
 int l = hasPathSum(root->left, sum - root->val);
 int r = hasPathSum(root->right, sum - root->val);
 return l || r;
 }
};
```

## Path Sum II(113)

### Question

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

For example: Given the below binary tree and `sum = 22`,



return

```
[
 [5, 4, 11, 2],
 [5, 8, 4, 5]
]
```

### Solution



```
class Solution {
public:
 vector<vector<int>> pathSum(TreeNode* root, int sum) {
 if(root == NULL) return {};
 vector<vector<int>> rtn, l, r;

 if(root->left == NULL && root->right == NULL &&
 sum - root->val == 0) return {{root->val}};

 rtn = pathSum(root->left, sum - root->val);
 r = pathSum(root->right, sum - root->val);

 rtn.insert(rtn.end(), r.begin(), r.end());

 for(int i = 0; i < rtn.size(); i++)
 rtn[i].insert(rtn[i].begin(), root->val);

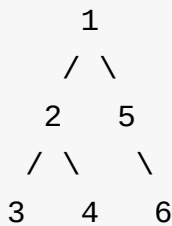
 return rtn;
 }
};
```

# Flatten Binary Tree to Linked List(114)

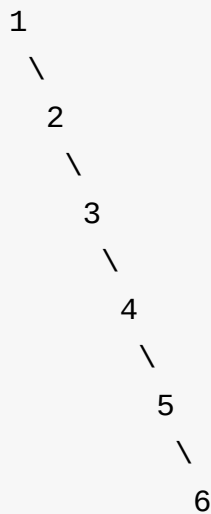
## Question

Given a binary tree, flatten it to a linked list in-place.

For example, Given



The flattened tree should look like:



[click to show hints.](#)

Hints: If you notice carefully in the flattened tree, each node's right child points to the next node of a pre-order traversal.

## Solution

```
class Solution {
public:
 void flatten(TreeNode* root) {
 if(root == NULL) return;

 flatten(root->left);
 flatten(root->right);

 TreeNode *cur = root, *r = root->right;

 root->right = root->left;
 root->left = NULL;

 while(cur->right!= NULL) cur = cur->right;
 cur->right = r;
 }
};
```

# Populating Next Right Pointers in Each Node(116)

## Question

Given a binary tree

```
struct TreeLinkNode {
 TreeLinkNode *left;
 TreeLinkNode *right;
 TreeLinkNode *next;
}
```

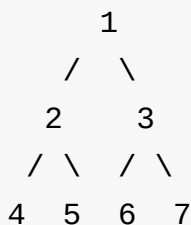
Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to `NULL`.

Initially, all next pointers are set to `NULL`.

### Note:

- You may only use constant extra space.
- You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and every parent has two children).

For example, Given the following perfect binary tree,



After calling your function, the tree should look like:

```
 1 -> NULL
 / \
 2 -> 3 -> NULL
 / \ / \
4->5->6->7 -> NULL
```

## Solution

```
/**
 * Definition for binary tree with next pointer.
 * struct TreeLinkNode {
 * int val;
 * TreeLinkNode *left, *right, *next;
 * TreeLinkNode(int x) : val(x), left(NULL),
 * * right(NULL), next(NULL) {}
 * };
 */
class Solution {
public:
 void helper(TreeLinkNode *l, TreeLinkNode *r){
 if(l != NULL) l->next = r;
 if(r != NULL) r->next = NULL;
 l = l->right;
 r = r->left;
 while(l!= NULL){
 l->next = r;
 l = l->right;
 r = r->left;
 }
 }

 void connect(TreeLinkNode *root) {
 if(root == NULL || (root->left == NULL &&
 root->right == NULL))
 return;

 if(root->left != NULL) connect(root->left);
 if(root->right != NULL) connect(root->right);

 root->next = NULL;

 helper(root->left, root->right);
 }
};
```

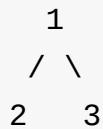
# Binary Tree Maximum Path Sum(124)

## Question

Given a binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path does not need to go through the root.

For example: Given the below binary tree,



Return 6 .

## Solution

```
class Solution {

public:
 int gmax = INT_MIN;

 int helper(TreeNode* root){
 if(root == NULL) return 0;
 int l = max(0, helper(root->left)),
 r = max(0, helper(root->right));

 int part = max(root->val,
 max(root->val + l, root->val + r));

 int sum = max(part, root->val + l + r);
 if (sum > gmax) gmax = sum;

 return part;
 }

 int maxPathSum(TreeNode* root) {
 return max(gmax, helper(root));
 }
};
```



# Sum Root to Leaf Numbers(129)

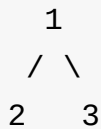
## Question

Given a binary tree containing digits from `0-9` only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path `1->2->3` which represents the number `123`.

Find the total sum of all root-to-leaf numbers.

For example,



The root-to-leaf path `1->2` represents the number `12`. The root-to-leaf path `1->3` represents the number `13`.

Return the `sum = 12 + 13 = 25`.

## Solution

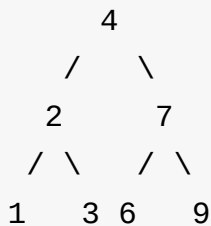
```
class Solution {
public:
 vector<string> helper(TreeNode* root){
 if(root == NULL) return {};
 if(root->left == NULL && root->right == NULL)
 return {to_string((root->val))};
 vector<string> l = helper(root->left),
 r = helper(root->right);
 l.insert(l.begin(), r.begin(), r.end());
 int size = l.size();
 for(int i = 0; i < size; i++){
 l[i] = to_string((root->val)) + l[i];
 }
 return l;
 }

 int sumNumbers(TreeNode* root) {
 if(root == NULL) return 0;
 vector<string> vec = helper(root);
 int s = vec.size();
 int sum = 0;
 for(int i = 0; i < s; i++){
 sum += stoi(vec[i].c_str());
 }
 return sum;
 }
};
```

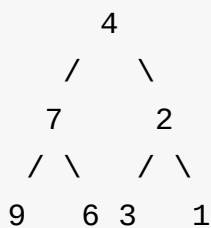
# Invert Binary Tree(226)

## Question

Invert a binary tree.



to



## Solution

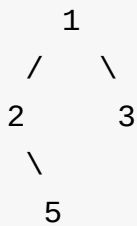
```
class Solution {
public:
 TreeNode* invertTree(TreeNode* root) {
 if(root == NULL) return NULL;
 TreeNode *l = invertTree(root->left),
 *r = invertTree(root->right);
 root->right = l;
 root->left = r;
 return root;
 }
};
```

# Binary Tree Paths(257)

## Question

Given a binary tree, return all root-to-leaf paths.

For example, given the following binary tree:



All root-to-leaf paths are:

```
["1->2->5", "1->3"]
```

## Solution

```
class Solution {
public:
 vector<string> binaryTreePaths(TreeNode* root) {
 if(root == NULL) return {};
 if(root->left == NULL && root->right == NULL)
 return {to_string(root->val)};
 vector<string> rtn = binaryTreePaths(root->left),
 r = binaryTreePaths(root->right);
 rtn.insert(rtn.end(), r.begin(), r.end());
 for(int i = 0; i < rtn.size(); i++)
 rtn[i] = to_string(root->val) + "->" + rtn[i];
 return rtn;
 }
};
```

# Kth Smallest Element in a BST(230)

## Question

Given a binary search tree, write a function **kthSmallest** to find the **kth** smallest element in it.

**Note:** You may assume **k** is always valid,  $1 \leq k \leq \text{BST's total elements}$ .

**Follow up:** What if the BST is modified (insert/delete operations) often and you need to find the **kth** smallest frequently? How would you optimize the **kthSmallest** routine?

**Hint:**

- Try to utilize the property of a BST.
- What if you could modify the BST node's structure?
- The optimal runtime complexity is  $O(\text{height of BST})$ .

## Solution

```
class Solution {
public:
 bool helper(TreeNode* root, int k, int* val){
 if (root == nullptr){
 val = 0;
 return false;
 }
 int left_size = 0, right_size = 0;

 bool rtn = helper(root->left, k, &left_size);
 if (rtn){
 *val = left_size;
 return rtn;
 }

 if (k - left_size == 1) {
 *val = root->val;
 return true;
 }

 rtn = helper(root->right, k - 1 - left_size, &right_size);
 if (rtn){
 *val = right_size;
 return rtn;
 }

 *val = left_size + 1 + right_size;
 return false;
 }

 int kthSmallest(TreeNode* root, int k) {
 int val = 0;
 if(helper(root, k, &val)) return val;
 else return -1;
 }
};
```

# Minimum Height Trees(310)

## Question

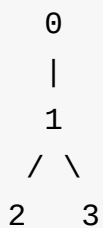
For a undirected graph with tree characteristics, we can choose any node as the root. The result graph is then a rooted tree. Among all possible rooted trees, those with minimum height are called minimum height trees (MHTs). Given such a graph, write a function to find all the MHTs and return a list of their root labels.

**Format** The graph contains `n` nodes which are labeled from `0` to `n - 1`. You will be given the number `n` and a list of undirected `edges` (each edge is a pair of labels).

You can assume that no duplicate `edges` will appear in `edges`. Since all edges are undirected, `[0, 1]` is the same as `[1, 0]` and thus will not appear together in `edges`.

### Example 1:

Given `n = 4, edges = [[1, 0], [1, 2], [1, 3]]`



return `[1]`

### Example 2:

Given `n = 6, edges = [[0, 3], [1, 3], [2, 3], [4, 3], [5, 4]]`

```
0 1 2
 \ | /
 3
 |
 4
 |
 5
```

```
return [3, 4]
```

Note:

(1) According to the definition of tree on Wikipedia: “a tree is an undirected graph in which any two vertices are connected by exactly one path. In other words, any connected graph without simple cycles is a tree.”

(2) The height of a rooted tree is the number of edges on the longest downward path between the root and a leaf.

## Solution



```

class Solution {
public:
 vector<int> findMinHeightTrees(int n, vector<pair<int, int>>& edges) {
 if(edges.empty()) return {0};
 vector<int> rtn, adj_counter(n);
 vector<vector<int>> mtx(n, vector<int>());
 queue<int> leaves;
 int counter = n;

 for (pair<int, int>& p : edges) {
 mtx[p.first].push_back(p.second);
 mtx[p.second].push_back(p.first);
 }
 for (int i = 0; i < n; ++i)
 if ((adj_counter[i] = mtx[i].size()) == 1) leaves.push(i);

 while (counter > 2) {
 int leaves_size = leaves.size();
 for (int i = 0; i < leaves_size; ++i) {
 int this_leaf = leaves.front(); leaves.pop();
 counter--;
 for (int& k : mtx[this_leaf])
 if(--adj_counter[k] == 1) leaves.push(k);
 }
 rtn.push_back(leaves.front());
 leaves.pop();
 if (!leaves.empty()) rtn.push_back(leaves.front());
 }

 return rtn;
 }
};

```

# Count of Smaller Numbers After Self(315)

## Question

You are given an integer array `nums` and you have to return a new counts array. The counts array has the property where `counts[i]` is the number of smaller elements to the right of `nums[i]`.

### Example:

```
Given nums = [5, 2, 6, 1]
```

```
To the right of 5 there are 2 smaller elements (2 and 1).
```

```
To the right of 2 there is only 1 smaller element (1).
```

```
To the right of 6 there is 1 smaller element (1).
```

```
To the right of 1 there is 0 smaller element.
```

Return the array `[2, 1, 1, 0]`.

## Solution

```

class Solution {
public:
 struct STree {
 int val, lt, dup;
 // lt is the # of inserted numbers that Less Than cur node
 // dup is the same val number
 STree *left, *right;
 STree(int x, int lv = 0, int dv = 0, STree* l = NULL, STree* r = NULL) : val(x), lt(lv), dup(dv), left(l), right(r){}
 };

 int insert(STree &T, int num){
 int rtn = 0;
 if (T.val == num){
 T.dup++; rtn = T.lt;
 }else if (num < T.val){
 T.lt++;
 if (T.left == NULL) T.left = new STree(num, 0, 1);
 else rtn = insert(*T.left, num);
 }else{
 rtn = (T.lt + T.dup);
 if (T.right == NULL) T.right = new STree(num, 0, 1);
 else rtn += insert(*T.right, num);
 }
 return rtn;
 }

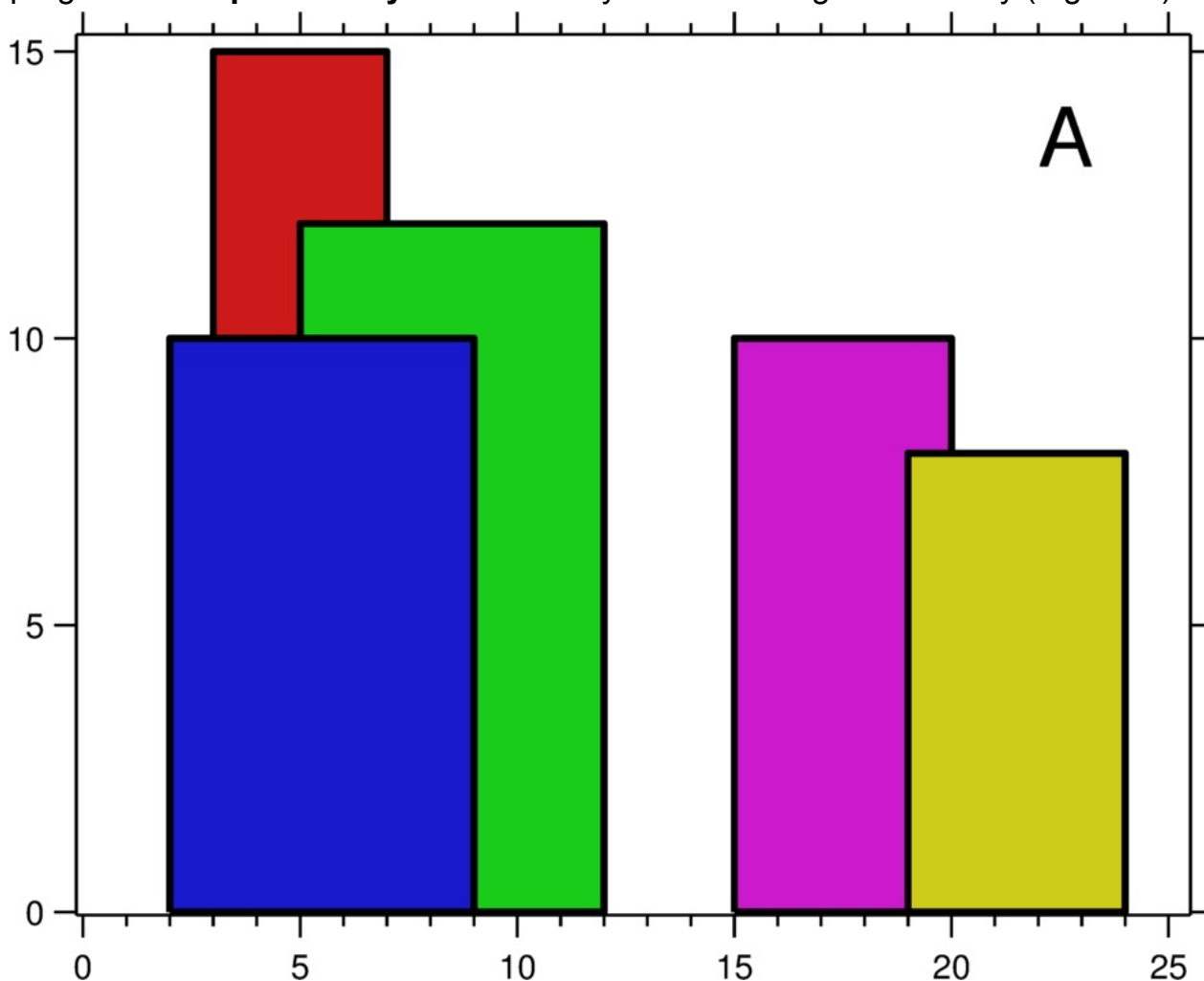
 vector<int> countSmaller(vector<int>& nums) {
 if (nums.empty()) return {};
 STree *T = new STree(nums.back(), 0, 1);
 int num_size = nums.size();
 vector<int> rtn(num_size); rtn.back() = 0;
 for (int i = num_size-2; i >=0; --i)
 rtn[i] = insert(*T, nums[i]);
 return rtn;
 }
};

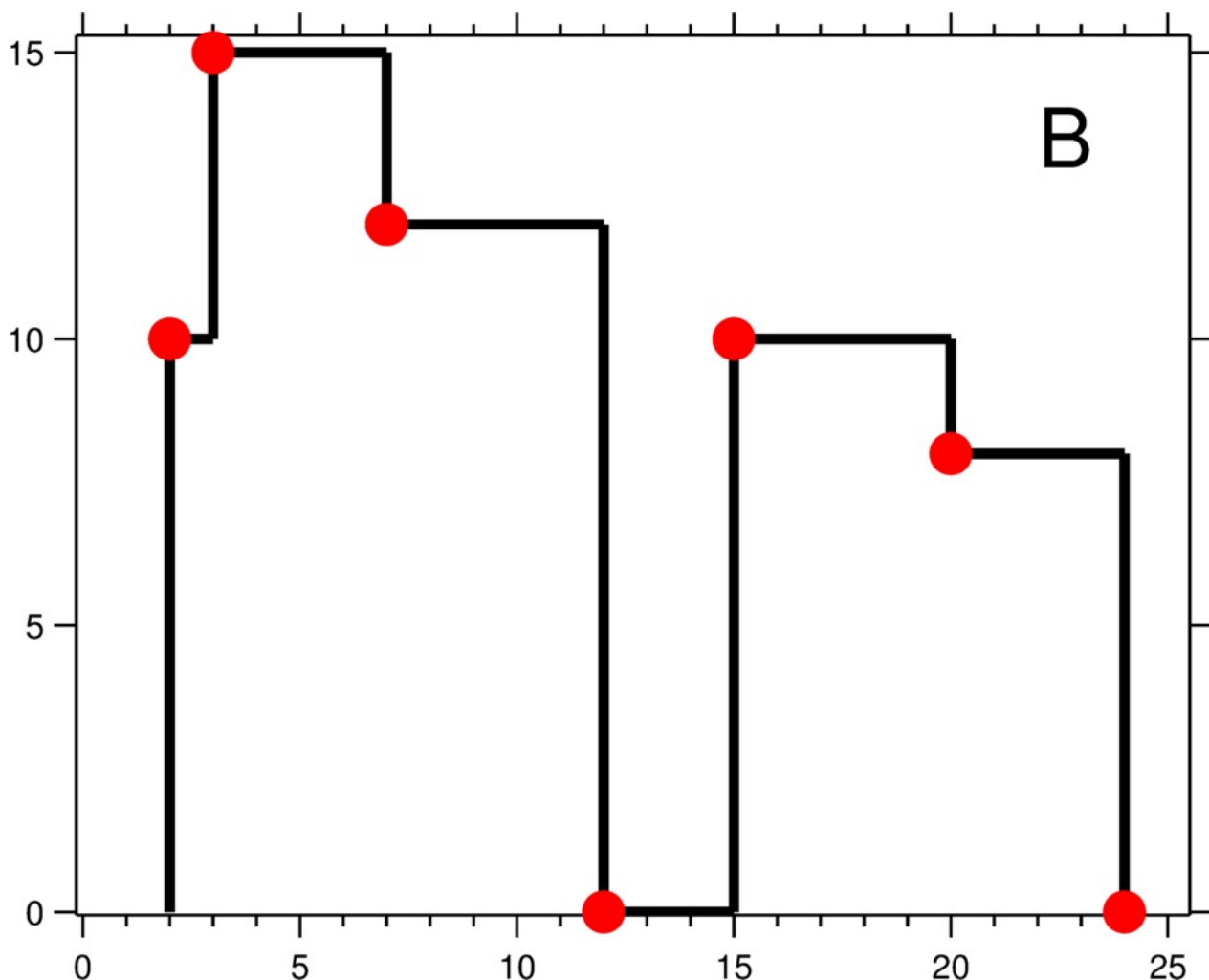
```

## The Skyline Problem(218)

### Question

A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Now suppose you are **given the locations and height of all the buildings** as shown on a cityscape photo (Figure A), write a program to **output the skyline** formed by these buildings collectively (Figure B).





Buildings Skyline Contour The geometric information of each building is

represented by a triplet of integers  $[L_i, R_i, H_i]$ , where  $L_i$  and  $R_i$  are the x coordinates of the left and right edge of the  $i$ th building, respectively, and  $H_i$  is its height. It is guaranteed that  $0 \leq L_i, R_i \leq \text{INT\_MAX}$ ,  $0 < H_i \leq \text{INT\_MAX}$ , and  $R_i - L_i > 0$ . You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height 0.

For instance, the dimensions of all buildings in Figure A are recorded as:  $[[2, 9, 10], [3, 7, 15], [5, 12, 12], [15, 20, 10], [19, 24, 8]]$ .

The output is a list of "key points" (red dots in Figure B) in the format of  $[[x_1, y_1], [x_2, y_2], [x_3, y_3], \dots]$  that uniquely defines a skyline. A key point is the left endpoint of a horizontal line segment. Note that the last key point, where the rightmost building ends, is merely used to mark the termination of the skyline, and always has zero height. Also, the ground in between any two adjacent buildings should be considered part of the skyline contour.

For instance, the skyline in Figure B should be represented as: `[ [2 10], [3 15], [7 12], [12 0], [15 10], [20 8], [24, 0] ]` .

### Notes:

- The number of buildings in any input list is guaranteed to be in the range `[0, 10000]` .
- The input list is already sorted in ascending order by the left x position `Li` .
- The output list must be sorted by the x position.
- There must be no consecutive horizontal lines of equal height in the output skyline. For instance, `[... [2 3], [4 5], [7 5], [11 5], [12 7] ...]` is not acceptable; the three lines of height 5 should be merged into one in the final output as such: `[... [2 3], [4 5], [12 7], ...]`

## Solution

```
class Solution {
public:
 vector<pair<int, int>> getSkyline(vector<vector<int>>& buildings) {
 vector<pair<int, int> > edges, rtn;
 multiset<int> mset = {0};
 int pre = 0, cur = 0;
 for (int i = 0; i < buildings.size(); ++i) {
 edges.push_back({buildings[i][0], buildings[i][2] * -1});
 edges.push_back({buildings[i][1], buildings[i][2]});
 }

 sort(edges.begin(), edges.end());

 for (pair<int, int>& p: edges) {
 pair<int, int> temp = p;
 p.second < 0 ? (void)mset.insert(p.second*-1): (void)mset.erase(p.second);
 if (pre != (cur = *mset.rbegin())) rtn.push_back({temp.first, cur});
 }
 return rtn;
 }
};
```



## **Design(Not Started 19)**



# LRU Cache(146)

## Question

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and set.

- `get(key)` - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.
- `set(key, value)` - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

## Solution

```
class LRUCache{
 const int cap;
 unordered_map<int, list<pair<int, int> >::iterator> map;
 list<pair<int, int>> lst;
public:
 LRUCache(int capacity):cap(capacity) {}

 int get(int key) {
 auto it = map.find(key);
 if(it == map.end()) return -1;

 lst.push_front(*it->second);

 lst.erase(it->second);
 it->second = lst.begin();
 return it->second->second;
 }

 void set(int key, int value) {
 auto it = map.find(key);
 if(it == map.end()){ // insert new
 while (map.size() >= cap) {
 map.erase(lst.crbegin()->first);
 lst.pop_back();
 }
 }else // reset existed
 lst.erase(it->second);

 lst.push_front({key, value});
 map[key] = lst.begin();
 }
};
```

## Min Stack(155)

### Question

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- getMin() -- Retrieve the minimum element in the stack.

### Solution

```
class MinStack {
 stack<int> s;
 stack<int> m;
public:
 void push(int x) {
 if (s.empty() || m.top() >= x) m.push(x);
 s.push(x);
 }

 void pop() {
 if (m.top() == s.top()) m.pop();
 s.pop();
 }

 int top() {
 return s.empty()? 0:s.top();
 }

 int getMin() {
 return m.empty()? 0:m.top();
 }
};
```

# Binary Search Tree Iterator(173)

## Question

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

Calling `next()` will return the next smallest number in the BST.

**Note:** `next()` and `hasNext()` should run in average  $O(1)$  time and uses  $O(h)$  memory, where  $h$  is the height of the tree.

## Solution

```
class BSTIterator {
 stack<TreeNode *> t;
public:
 BSTIterator(TreeNode *root) {
 TreeNode * cur = root;
 while(cur){t.push(cur); cur = cur->left; }
 }

 /** @return whether we have a next smallest number */
 bool hasNext() {
 return !t.empty();
 }

 /** @return the next smallest number */
 int next() {
 if(!hasNext()) return 0;
 TreeNode * top = t.top(), *r = top->right;
 t.pop();
 if(r!= nullptr)
 while (r) {t.push(r); r = r->left; }

 return top->val;
 }
};

/**
 * Your BSTIterator will be called like this:
 * BSTIterator i = BSTIterator(root);
 * while (i.hasNext()) cout << i.next();
 */
```

# Peeking Iterator(284)

## Question

Given an Iterator class interface with methods: `next()` and `hasNext()` , design and implement a PeekingIterator that support the `peek()` operation -- it essentially `peek()` at the element that will be returned by the next call to `next()` .

Here is an example. Assume that the iterator is initialized to the beginning of the list: `[1, 2, 3]` .

Call `next()` gets you 1, the first element in the list.

Now you call `peek()` and it returns 2, the next element. Calling `next()` after that still return 2.

You call `next()` the final time and it returns 3, the last element. Calling `hasNext()` after that should return false.

### Hint:

- Think of "looking ahead". You want to cache the next element.
- Is one variable sufficient? Why or why not?
- Test your design with call order of `peek()` before `next()` vs `next()` before `peek()`.
- For a clean implementation, check out Google's guava library source code.

**Follow up:** How would you extend your design to be generic and work with all types, not just integer?

## Solution

```
// Below is the interface for Iterator, which is already defined for you.
// **DO NOT** modify the interface for Iterator.
class Iterator {
 struct Data;
 Data;
```

```

 Data* data;
public:
 Iterator(const vector<int>& nums);
 Iterator(const Iterator& iter);
 virtual ~Iterator();
 // Returns the next element in the iteration.
 int next();
 // Returns true if the iteration has more elements.
 bool hasNext() const;
};

class PeekingIterator : public Iterator {
 int copy = 0;
 bool peeked, exists;
public:
 PeekingIterator(const vector<int>& nums) : Iterator(nums) {
 // Initialize any member here.
 // **DO NOT** save a copy of nums and manipulate it directly.
 // You should only use the Iterator interface methods.
 exists = Iterator::hasNext();
 peeked = false;
 }

 // Returns the next element in the iteration without advancing
 int peek() {
 if (peeked) return copy;
 peeked = true;
 if (exists){
 copy = Iterator::next();
 return copy;
 }
 }


 // hasNext() and next() should behave the same as in the Iterator class.
 // Override them if needed.
 int next() {
 if (!peeked) peek();
 peeked = false;
 exists = Iterator::hasNext();
 }
};

```



```
 return copy;
 }

 bool hasNext() const {
 return exists;
 }
};
```



## Find Median from Data Stream(295)

### Question

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

Examples: `[2, 3, 4]` , the median is `3`

`[2, 3]` , the median is  $(2 + 3) / 2 = 2.5$

Design a data structure that supports the following two operations:

- `void addNum(int num)` - Add a integer number from the data stream to the data structure.
- `double findMedian()` - Return the median of all elements so far.

For example:

```
add(1)
add(2)
findMedian() -> 1.5
add(3)
findMedian() -> 2
```

### Solution

```

class MedianFinder {
public:
 vector<int> vec = {};
 int insert_pos(int start, int end, int num){
 if(start == end)
 return num >= vec[(start + end) / 2] ?
 start + 1 : start;

 return (num <= vec[(start + end) / 2]) ?
 insert_pos(start, (start + end) / 2, num) :
 insert_pos(1 + (start + end) / 2, end, num);
 }

 // Adds a number into the data structure.
 void addNum(int num) {
 (vec.empty() || num >= vec.back()) ?
 (void)vec.push_back(num):
 (void)vec.insert(vec.begin()+insert_pos(0,
 vec.size()-1, num), num);
 }

 // Returns the median of current data stream
 double findMedian() {
 if(vec.size() == 0) return 0;
 int s = vec.size();
 return s%2 == 0 ?
 (vec[(s/2) - 1] + vec[(s/2)])/2.0 : vec[s/2];
 }
};

// Your MedianFinder object will be instantiated and called as such
// MedianFinder mf;
// mf.addNum(1);
// mf.findMedian();

```

# Implement Trie (Prefix Tree)(208)

## Question

Implement a trie with `insert`, `search`, and `startsWith` methods.

**Note:** You may assume that all inputs are consist of lowercase letters `a-z`.

## Solution

```
class TrieNode {
public:
 // Initialize your data structure here.
 TrieNode() {

 }
 unordered_map<char, TrieNode*> map;
 bool exists = false;
};

class Trie {
public:
 Trie() {
 root = new TrieNode();
 }

 // Inserts a word into the trie.
 void insert(string word) {
 TrieNode* cur = root;
 for(int i = 0; i < word.size(); i++){
 auto it = cur->map.find(word[i]);
 if (it == cur->map.end())
 cur->map[word[i]] = new TrieNode();
 cur = cur->map[word[i]];
 }
 cur->exists = true;
 }
};
```

```
 }

 // Returns if the word is in the trie.
 bool search(string word) {
 TrieNode* cur = root;
 for(int i = 0; i < word.size(); i++){
 auto it = cur->map.find(word[i]);
 if (it == cur->map.end()) return false;
 else cur = cur->map[word[i]];
 }
 return cur->exists;
 }

 // Returns if there is any word in the trie
 // that starts with the given prefix.
 bool startsWith(string prefix) {
 TrieNode* cur = root;
 for(int i = 0; i < prefix.size(); i++){
 auto it = cur->map.find(prefix[i]);
 if (it == cur->map.end()) return false;
 else cur = cur->map[prefix[i]];
 }
 return cur->map.size() > 0 || cur->exists;
 }

private:
 TrieNode* root;
};
```

## Implement Stack using Queues(225)

Implement the following operations of a stack using queues.

- `push(x)` -- Push element `x` onto stack.
- `pop()` -- Removes the element on top of the stack.
- `top()` -- Get the top element.
- `empty()` -- Return whether the stack is empty.

### Notes:

- You must use only standard operations of a queue -- which means only `push` to back, `peek/pop` from front, `size`, and `is empty` operations are valid.
- Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue.
- You may assume that all operations are valid (for example, no `pop` or `top` operations will be called on an empty stack).

## Solution

```
class Stack {
 // only put the first in q1, all the rest in q2;
 queue<int> q1, q2;
public:
 // Push element x onto stack.
 void push(int x) {
 if(q1.empty()){
 q1.push(x);
 }else{
 q2.push(x);
 while(q2.size() > 1) {
 q1.push(q2.front());
 q2.pop();
 }
 swap(q1, q2);
 }
 }

 // Removes the element on top of the stack.
 void pop() {
 q1.pop();
 while(q2.size() >= 1) {
 q1.push(q2.front());
 q2.pop();
 }
 }

 // Get the top element.
 int top() {
 return q1.front();
 }

 // Return whether the stack is empty.
 bool empty() {
 return q1.empty();
 }
};
```

## Depth-first Search (Not Started)



# Number of Islands(200)

## Question

Given a 2d grid map of '1' s (land) and '0' s (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

### Example 1:

```
11110
11010
11000
00000
```

Answer: 1

### Example 2:

```
11000
11000
00100
00011
```

Answer: 3

## Solution

```
class Solution {
public:
 int helper(vector<vector<char>>& grid, int i, int j){
 if(i > grid.size() -1 ||
 j > grid[0].size() -1 ||
 i < 0 || j < 0)
 return 0;
 if (grid[i][j] == '1') {
 grid[i][j] = '0';
 helper(grid, i+1, j);
 helper(grid, i, j+1);
 helper(grid, i-1, j);
 helper(grid, i, j-1);
 }
 return 1;
 }

 int numIslands(vector<vector<char>>& grid) {
 if(grid.empty() || grid[0].empty()) return 0;
 int rtn = 0;
 for(int i = 0; i < grid.size(); i++)
 for(int j = 0; j < grid[0].size(); j++)
 if(grid[i][j] == '1') rtn += helper(grid, i, j);
 return rtn;
 }
};
```

## Math

# Power of Three(326)

## Question

Given an integer, write a function to determine if it is a power of three.

Follow up: Could you do it without using any loop / recursion?

## Solution

```
class Solution {
public:
 bool isPowerOfThree(int n) {
 if(n == 3 || n == 1) return true;
 if(n == 0 || n % 3 != 0) return false;
 return isPowerOfThree(n/3);
 }
};
```

# Perfect Squares(279)

## Question

Given a positive integer  $n$ , find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to  $n$ .

For example, given  $n = 12$ , return 3 because  $12 = 4 + 4 + 4$ ; given  $n = 13$ , return 2 because  $13 = 4 + 9$ .

## Solution

```
class Solution {
public:
 int numSquares(int n) {
 int squares_size = ceil(sqrt(n)), map[n+1],
 squares[squares_size];
 map[1] = 1, map[0] = 0;

 for(int i = 0; i <= squares_size; i++) squares[i] = i*i;

 for (int i = 2; i <= n; i++) {
 map[i] = INT_MAX;
 for (int j = 1; j <= squares_size &&
 squares[j] <= i; j++)
 if (map[i - squares[j]] + 1 <= map[i])
 map[i] = map[i - squares[j]] + 1;
 }

 return map[n] ;
 }
};
```

# Power of Two(231)

## Question

Given an integer, write a function to determine if it is a power of two.

## Solution

```
class Solution {
public:
 bool isPowerOfTwo(int n) {
 for(int i = 0; i < 32 && n > 0; i++)
 if(n == (1 << i)) return true;

 return false;
 }
};
```

# Ugly Number

## Question

Write a program to check whether a given number is an ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5 . For example, 6, 8 are ugly while 14 is not ugly since it includes another prime factor 7 .

Note that 1 is typically treated as an ugly number.

## Solution

```
class Solution {
public:
 bool isUgly(int num) {
 if (num == 0) return false;
 if (num == 1) return true;
 if (num%2 == 0) {
 return isUgly(num/2);
 } else if (num%3 == 0) {
 return isUgly(num/3);
 } else if (num%5 == 0) {
 return isUgly(num/5);
 }

 return false;
 }
};
```

# Ugly Number II(264)

## Question

Write a program to find the  $n$ -th ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5. For example, 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 is the sequence of the first 10 ugly numbers.

Note that 1 is typically treated as an ugly number.

Hint:

The naive approach is to call `isUgly` for every number until you reach the  $n$ -th one. Most numbers are not ugly. Try to focus your effort on generating only the ugly ones. An ugly number must be multiplied by either 2, 3, or 5 from a smaller ugly number. The key is how to maintain the order of the ugly numbers. Try a similar approach of merging from three sorted lists: L1, L2, and L3. Assume you have  $U_k$ , the  $k$ th ugly number. Then  $U_{k+1}$  must be  $\text{Min}(L1 \cdot 2, L2 \cdot 3, L3 \cdot 5)$ .

## Solution

```
class Solution {
public:
 int nthUglyNumber(int n) {
 int num[n], p2 = 0, p3 = 0, p5 = 0; num[0] = 1;
 for (int i = 1; i < n; i++) {
 num[i] = min(2*num[p2], min(3*num[p3], 5*num[p5]));
 if (num[i] == 2*num[p2]) p2++;
 if (num[i] == 3*num[p3]) p3++;
 if (num[i] == 5*num[p5]) p5++;
 }
 return num[n-1];
 }
};
```





# Super Ugly Number(313)

## Question

Write a program to find the n-th super ugly number.

Super ugly numbers are positive numbers whose all prime factors are in the given prime list `primes` of size `k` . For example, `[1, 2, 4, 7, 8, 13, 14, 16, 19, 26, 28, 32]` is the sequence of the first 12 super ugly numbers given `primes = [2, 7, 13, 19]` of size 4.

### Note:

- (1) `1` is a super ugly number for any given `primes` .
- (2) The given numbers in `primes` are in ascending order.
- (3)  $0 < k \leq 100, 0 < n \leq 10^6, 0 < primes[i] < 1000$ .

## Solution

```
class Solution {
public:
 int nthSuperUglyNumber(int n, vector<int>& primes) {
 int primesL = primes.size(), nextInx;
 vector<int> index(primesL, 0), ugly(n, INT_MAX), indexVal(primesL, 1);
 ugly[0]=1;

 for(int i=1; i<n; ++i){
 ugly[i] = indexVal[0];
 nextInx = 0;
 for(int j=1; j<primesL; ++j){
 if(indexVal[j] < ugly[i]){
 nextInx = j;
 ugly[i]=indexVal[nextInx];
 }else if(indexVal[j] == ugly[i])
 indexVal[j] = ugly[++index[j]]*primes[j];
 }
 indexVal[nextInx] = ugly[++index[nextInx]]*primes[nextInx];
 }
 return ugly[n-1];
 }
};
```

# Patching Array(330)

## Question

Given a sorted positive integer array `nums` and an integer `n`, add/patch elements to the array such that any number in range `[1, n]` inclusive can be formed by the sum of some elements in the array. Return the minimum number of patches required.

**Example 1:** `nums = [1, 3]` , `n = 6` Return `1` .

Combinations of `nums` are `[1]`, `[3]`, `[1, 3]` , which form possible sums of: `1`, `3`, `4` .

Now if we add/patch `2` to `nums`, the combinations are: `[1]`, `[2]`, `[3]`, `[1, 3]`, `[2, 3]`, `[1, 2, 3]` .

Possible sums are `1`, `2`, `3`, `4`, `5`, `6` , which now covers the range `[1, 6]` .

So we only need `1` patch.

**Example 2:**

`nums = [1, 5, 10]` , `n = 20`

Return `2` .

The two patches can be `[2, 4]` .

**Example 3:**

`nums = [1, 2, 2]` , `n = 5`

Return `0` .

## Solution

```
class Solution {
public:
 int minPatches(vector<int>& nums, int n) {
 int len = nums.size(), sum = 0, patch = 0, count = 0;
 while (sum < n) {
 if (count < len && nums[count] <= sum + 1)
 sum += nums[count++];
 else {
 ++patch;
 if (sum > (INT_MAX - 1) / 2) sum = INT_MAX;
 else sum += (sum + 1);
 }
 }
 return patch;
 }
};
```

## Hash Table(Not started)

## Two Pointers(Not Started)

## **Trie(Not started 3)**



## Head(Not Started)

## **Greedy(Not Started)**

## **Devide And Conquer(Not Started)**

## Sort(Not Started 12)

## **Breadth-first Search (Not started 18)**

## **Stack(Not Started 19)**

## Binary Search(Not Started 24)

## **Backtracking(Not Started 29)**