Jaden's Solutions of All Typical Problems

1 Sort, mergesort	2 quicksort, partition
3 Binary search tree	4 Binary Tree
5 array	6 graph
7 bit operation	8 Linked list
9 heap, priority queue	10 Dynamic Programming
11 hash map, hash table	12 string
13 Permutations, recursion	14 Backpack or Ksum (DP)
15 stack	16 queue
17 OOD design	18 Container
19 system design	20 others
21 C# basics	22 Company

1 Sort, mergesort, merge

1.1 Bubble Sort

- O(n square) time complexity
- compare first one with all others, replace if larger, then keeping comparing until find the smallest one. Then start from next value.

```
protected virtual void BubbleSort()
{
    int[] arr = BuildArray();
    for(int i=0; i<arr.Length; i++)
    {
        for(int j = i + 1; j < arr.Length; j++)
```

```
        {
            if(arr[i] > arr[j])
            {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    PrintArray(arr);
}

protected virtual void BubbleSortFromUserInput()
{
    string str = Console.ReadLine();
    int num;
    int[] arr = str.Split(' ').Where(o => int.TryParse(o, out num))
                              .Select(o => int.Parse(o))
                              .ToArray();
    for(int i=0; i<arr.Length; i++)
    {
        for(int j=i+1; j<arr.Length; j++)
        {
            if(arr[i] < arr[j])
            {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    PrintArray(arr);
}
```

1.2 MergeSort

Write MergeSort algorithm implementation. https://www.geeksforgeeks.org/merge-sort/

- Just need to split array into each value and merge all together
- not stable as same value node position can be changed
- Just find halfLen, then divide based on two array with different length as it cannot always evenly be divided.
- T - O(nlogn) => becase split take logn and each merge take n, totally merge logn times. so logn+nlogn
- S - O(n) => at same time only one level exists, one level has all n elements

```
public int[] MergeSort(int[] arr)
{
    if (arr == null) return arr;
    if (arr.Length == 1) return arr;
    return MergeSort(arr, arr.Length);
```

```
}

protected int[] MergeSort(int[] arr, int len)
{
    if (len == 1) return arr;
    int halfLen = arr.Length / 2;
    int[] left = new int[halfLen];
    int[] right = new int[len-halfLen];
    for (int i = 0; i < halfLen; i++)
    {
        left[i] = arr[i];
    }
    for (int i = 0; i < len - halfLen; i++)
    {
        right[i] = arr[i + halfLen];
    }
    left = MergeSort(left, halfLen);
    right = MergeSort(right, len - halfLen);
    return Merge(left, right);
}

protected int[] Merge(int[] left, int[] right)
{
    int leftIndex = 0;
    int rightIndex = 0;
    int curIndex = 0;
    int[] res = new int[left.Length + right.Length];
    while (leftIndex < left.Length && rightIndex < right.Length)
    {
        if (left[leftIndex] > right[rightIndex])
        {
            res[curIndex] = right[rightIndex];
            rightIndex++;
        }
        else
        {
            res[curIndex] = left[leftIndex];
            leftIndex++;
        }
        curIndex++;
    }
    while (leftIndex < left.Length)
    {
        res[curIndex] = left[leftIndex];
        leftIndex++;
        curIndex++;
    }
    while (rightIndex < right.Length)
    {
        res[curIndex] = right[rightIndex];
        rightIndex++;
        curIndex++;
    }
```

```
        return res;
    }
```

1.3 Insertion Sort

need to write insertion sort here

- basically insert each value to a new array, shift position if needed
- T - (n^2) (each node need to shifted n times worst cases)

1.4 Merge Intervals

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].

- sort the vectors by start of interval
- then traverse each node and check if next node is needed or not

https://leetcode.com/problems/merge-intervals/description/

```
/**
 * Definition for an interval.
 * public class Interval {
 *     public int start;
 *     public int end;
 *     public Interval() { start = 0; end = 0; }
 *     public Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public IList<Interval> Merge(IList<Interval> intervals) {
        intervals = intervals.OrderBy(o => o.start).ToList();
        if(intervals.Count() <= 1) return intervals;
        IList<Interval> res = new List<Interval>();
        res.Add(intervals[0]);
        for(int i=1; i<intervals.Count(); i++){
            if(intervals[i].start <= res[res.Count() -1].end
                && intervals[i].end > res[res.Count() - 1].end)
            {
                res[res.Count() - 1].end = intervals[i].end;
            }
            else if(intervals[i].start > res[res.Count() -1].end){
                res.Add(intervals[i]);
            }
        }
        return res;
    }
}
```

1.5 Insert Interval

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].

https://leetcode.com/problems/insert-interval/description/

- it's already sorted, so first insert it into given list, so the problem become merge intervals like previous problem
- just traverse and merge if we need

```
/**
 * Definition for an interval.
 * public class Interval {
 *     public int start;
 *     public int end;
 *     public Interval() { start = 0; end = 0; }
 *     public Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public IList<Interval> Insert(IList<Interval> intervals, Interval newInterval)
{
        IList<Interval> res = new List<Interval>();
        if(intervals == null || (intervals!=null && intervals.Count() == 0)){
            res.Add(newInterval);
            return res;
        }

        IList<Interval> list = new List<Interval>(intervals);
        int i=0;
        for(i=0; i<list.Count(); i++){
            if(newInterval.start <= list[i].start){
                list.Insert(i, newInterval);
                break;
            }
        }
        if(i == list.Count()) list.Add(newInterval);

        res.Add(list[0]);
        for(int j=1; j<list.Count(); j++){
            if(list[j].start <= res[res.Count() -1].end && list[j].end >
res[res.Count()-1].end)
                res[res.Count()-1].end = list[j].end;
            else if(list[j].start > res[res.Count()-1].end)
                res.Add(list[j]);
        }
        return res;
    }
}
```

2 quicksort, partition

2.1 quicksort

Implment quicksort so that each time partition will return index of pivot vlaue.
https://www.geeksforgeeks.org/quick-sort/
https://www.cp.eng.chula.ac.th/~vishnu/datastructure/QuickSort.pdf

- Use slow and fast way will return partition index correctly (always the last one <= pivot)
- pick which one to be pivoit in partion does not matter. Just impact the speed
- if everytime pick the largest or smallest element, then T - O(n^2) (as no binary split anymore)
- on average T - O(nlogn)
- S - O(1)
- unstable sort (same value elements position may change)

```
public void QuickSort(int[] arr)
{
    if (arr == null) return;
    if (arr.Length == 0) return;
    QuickSort(arr, 0, arr.Length - 1);
}

protected void QuickSort(int[] arr, int start, int end)
{
    if (start < end)
    {
        int index = Partition(arr, start, end);
        QuickSort(arr, start, index - 1);
        QuickSort(arr, index + 1, end);
    }
}

protected int Partition(int[] arr, int start, int end)
{
    // last element as pivot value, so finally pivot value will stay at index slow
    // if choose not the last one as pivot, need to swap pivot element with the
last one to make sure slow index it good finally
    int pivot = arr[end];
    int slow = start - 1;
    int fast = start;
    while (fast <= end)
    {
        if (arr[fast] <= pivot)
        {  // bring smaller than pivot value to left
            slow++;
            int temp = arr[slow];
            arr[slow] = arr[fast];
            arr[fast] = temp;
        }
        fast++;
```

```
        }
        return slow;  // slow is the correct final index of pivot
    }
```

2.2 Rearrange positive and negative numbers in O(n) time and O(1) extra space

An array contains both positive and negative numbers in random order. Rearrange the array elements so that positive and negative numbers are placed alternatively. Number of positive and negative numbers need not be equal. If there are more positive numbers they appear at the end of the array. If there are more negative numbers, they too appear in the end of the array.

For example, if the input array is
[-1, 2, -3, 4, 5, 6, -7, 8, 9]
then the output should be
[9, -7, 8, -3, 5, -1, 2, 4, 6]

https://www.geeksforgeeks.org/rearrange-positive-and-negative-numbers-publish/

Solution:

not S - (1) solution:

- traverse and insert into two array, then combine two array
- T - O(n)
- S - O(n)

partition, pivot is 0

- The solution is to first separate positive and negative numbers using partition process of QuickSort. In the partition process, consider 0 as value of pivot element so that all negative numbers are placed before positive numbers. Once negative and positive numbers are separated, we start from the first negative number and first positive number, and swap every alternate negative number with next positive number.
- 0 is positive
- T - O(n), S - O(1)
- https://ideone.com/TkkkZo

```
public void Rearrange(int A[],int n){
    int index = partition(A,n);
    for(int i=1; i<index && index<n; i+=2){  // i<n is also ok, but not optimized
        swap(A,i,index);
        index++;
    }
}
protected int partition(int A[], int n) {
    int slow = 0;  // begin index
    int fast = 0;
    int pivot = 0;
    while(fast < n){     // pivot = 0
        if(A[fast] >= pivot){
```

```
                swap(A,slow, fast);
                slow++;
            }
            fast++;
        }
        return slow;  // first negative
    }
```

2.3 Partition Arrray

Given an array "a" of integers and an int "k", Partition the array (i.e move the elements in "a") such that
-- All elements < k are moved to the left
-- All elements >= k are moved to the right
Return the partitioning Index, i.e the first index "i" a[i] >= k.

- just use quick sort partition template, pivot is givin k

```
// will add code later
```

2.4 Sort Letters by Case

Given a string which contains only letters. Sort it by lower case first and upper case second.
Note
It's not necessary to keep the original order of lower-case letters and upper case letters.
Example
For "abAcD", a reasonable answer is "acbAD"
Challenge Expand
Do it in one-pass and in-place.

- same with partition array, pivot is if upper or lower case
- we do not need to really sort lettters by alphabetic order
- actually, we just go through the array, if face lower case, just replace it will the begin index. if upper case, keep going until the end of the array.

```
// will add code later
```

2.5 Sort Colors

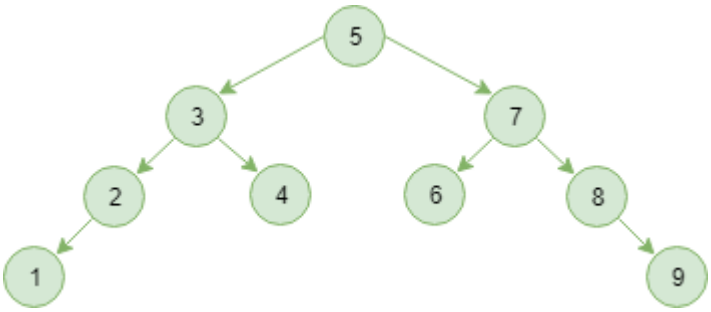
Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.
Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.
Note:
You are not suppose to use the library's sort function for this problem.

For this problem:

- partition left 0,1 and right 2.
- then partion left 0, and right 1
- now it is sorted
  For 4 numbers, 0,1,2,3.
  partition each one will be O(3n), but we can partion from middle, the O(2n)
  so For k numbers 0,1,2.....k
  partion from middle, we have O(nlogk)
- this solution is better than normal quick sort partition template, see below for details:

```
void sortColors(int A[], int n) {
    int slow = 0;
    int fast = 0;
    int pivot = 0;
    while(fast < n){     // pivot = 0, put all 0s to the beginning
        if(A[fast] == pivot){
            swap(A,slow, fast);
            slow++;
        }
        fast++;
    }
    fast = slow;   // pivot == 1, put all 1s to the beginning after all 0s
    pivot = 1;
    while(fast < n){
        if(A[fast] == pivot){
            swap(A,slow, fast);
            slow++;
        }
        fast++;
    }
}
```

3 Binary search tree



- each tree node has left and right node
- left is smaller than parent, right is larger than parent
- usually it has no duplicates (othersise same value may be in diffent levels, search will be slow, if needed, add a counter for each node)
- it is different with binary tree (parent is not smaller or bigger than childen)
- it is ordered data structure as inorder traversal is ordered
- inert, search, delete take T - O(logn) time

- deletion is hard (no child, has one child, has both children)
    - if no child, just delete
    - if one child, replace it with the child
    - if has both children, then replace it with its right node's left most node (it is the inorder successor node)

```
public class TreeNode
{
    public int val;
    public TreeNode left;
    public TreeNode right;
    public TreeNode(int value)
    {
        this.val = value;
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp
{
    // Implemented by using TreeNode
    // Supports Pre-order, in-order, post-order iteration,
    // Supports BFS & DFS iteration and search
    public class BinarySearchTree
    {
        private TreeNode root;
        public BinarySearchTree()
        {
            root = null;
        }

        public void Test()
        {
            BinarySearchTree bst = BuildBinarySearchTree();
            bst.PreOrderTraversal();
            bst.InOrderTraversal();
            bst.PostOrderTraversal();
            bst.BFSTraversal();
            bst.DFSTraversal();
            bst.BFSSearch(6);
            bst.BinarySearch(6);
            bst.Height();
        }

        //                5
```

```
        //             3       7
        //           2   4   6    8
        //        1                 9
        public BinarySearchTree BuildBinarySearchTree()
        {
            BinarySearchTree bst = new BinarySearchTree();
            int[] arr = new int[] { 5, 3, 7, 2, 4, 6, 8, 1, 9};
            foreach (int v in arr)
            {
                bst.AddNode(v);
            }
            return bst;
        }

        protected virtual void AddNode(int value)
        {
            if (root == null)
            {
                root = new TreeNode(value);
                return;
            }
            AddNode(root, value);
        }

        protected void AddNode(TreeNode node, int value)
        {
            if(node.val >= value)
            {
                if (node.left == null)
                    node.left = new TreeNode(value);
                else
                    AddNode(node.left, value);
            }
            else
            {
                if (node.right == null)
                    node.right = new TreeNode(value);
                else
                    AddNode(node.right, value);
            }
        }

        protected virtual void VisitNode(TreeNode node)
        {
            Console.Write(node.val.ToString() + ' ');
        }

        protected void PreOrderTraversal()
        {
            Console.WriteLine("PreOrderTraversal");
            PreOrderTraversal(root);
            Console.WriteLine("\n");
        }
```

```
        protected void PreOrderTraversal(TreeNode node)
        {
            if (node == null) return;
            VisitNode(node);
            PreOrderTraversal(node.left);
            PreOrderTraversal(node.right);
        }

        protected void InOrderTraversal()
        {
            Console.WriteLine("InOrderTraversal");
            InOrderTraversal(root);
            Console.WriteLine("\n");
        }

        protected void InOrderTraversal(TreeNode node)
        {
            if (node == null) return;
            InOrderTraversal(node.left);
            VisitNode(node);
            InOrderTraversal(node.right);
        }

        protected void PostOrderTraversal()
        {
            Console.WriteLine("PostOrderTraversal");
            PostOrderTraversal(root);
            Console.WriteLine("\n");
        }

        protected void PostOrderTraversal(TreeNode node)
        {
            if (node == null) return;
            PostOrderTraversal(node.left);
            PostOrderTraversal(node.right);
            VisitNode(node);
        }

        protected void BFSTraversal()
        {
            Console.WriteLine("BFSTraversal");
            BFSTraversal(root);
            Console.WriteLine();
        }

        // BFS or Level Order(need count of cur queue for define level) Traversal
        // Use Queue is eaiser and more straigtforward
        protected void BFSTraversal(TreeNode node)
        {
            if (node == null) return;
            Queue<TreeNode> queue = new Queue<TreeNode>();
            queue.Enqueue(node);
            while(queue.Count() > 0)
            {
```

```
                TreeNode curNode = queue.Dequeue();
                VisitNode(curNode);
                if (curNode.left != null)
                    queue.Enqueue(curNode.left);
                if (curNode.right != null)
                    queue.Enqueue(curNode.right);
            }
        }

        protected int Height(TreeNode node)
        {
            if (node == null) return 0;
            int leftHeight = Height(node.left);
            int rightHeight = Height(node.right);
            return Math.Max(leftHeight, rightHeight) + 1;
        }
    }
}
```

3.1 serialize Binary search tree and restore from preorder traversal

- use preorder traversal to serialize
- restore: build a range, initial INT_MIN, INT_MAX, then change, use reference pointer
- T - O(n) same with preorder traversal
- pre-order should just work, just need to have range check to avoid node empty issue
- compare with "build from sorted array" problem below
- normally int value should not be int.MaxValue and int.MinValue

https://www.geeksforgeeks.org/construct-bst-from-given-preorder-traversa/
https://articles.leetcode.com/determine-if-binary-tree-is-binary/ https://articles.leetcode.com/saving-binary-search-tree-to-file/ https://articles.leetcode.com/serializationdeserialization-of-binary/
https://www.programcreek.com/2014/05/leetcode-serialize-and-deserialize-binary-tree-java/
https://leetcode.com/problems/serialize-and-deserialize-bst/description/

```
public TreeNode BuildBST(int[] arr, ref int index, int min, int max){
    if(index > arr.Length -1) return null;
    if(arr[index] < min || arr[index] > max) return null;
    TreeNode node = new Node(arr[index]);
    index++;
    node.left = buildBST(arr, index, min, node.val);
    node.right = buildBST(arr, index, node.val, max);
    return node;
}

public TreeNode BuildBST(int[] arr)
{
    if(arr == null) return null;
    if (arr.Length == 0) return NULL;
        TreeNode root = BuildBST(arr, 0, int.MinValue, int.MaxValue);
```

```
        return root;
}
```

3.2 build BST from sorted array with minimum height

- most balanced == minimum height
- binary search algorithm (problem #5.8), each get mid value as root

```
pubic TreeNode CreateMinimalBinarySearchTree(int[] arr){
    if(arr == null) return null;
    return CreateMinimalBinarySearchTree(arr, 0 arr.Length-1);
}

protected TreeNode CreateMinimalBinarySearchTree(int[] arr, int start, int end){
    if(start > end) return null;
    int mid = end + (start - end) / 2;
    TreeNode node = new TreeNode(arr[mid]);
    node.left = CreateMinimalBinarySearchTree(arr, start, mid-1);
    mode.right = CreateMinimalBinarySearchTree(arr, mid+1, end);
}
```

3.4 convert sorted singly list to BST

- just traverse linked list and store it into array
- just pick mid and assign left and right
- T - O(n)

```
pubic TreeNode CreateMinimalBinarySearchTree(TreeNode root){
    if(root == null) return null;
    List<int> list = new List<int>();
    while(root != null){
        list.Add(root.val);
        root = root.next;
    }
    return CreateMinimalBinarySearchTree(list.ToArray(), 0 list.Count()-1);
}
protected TreeNode CreateMinimalBinarySearchTree(int[] arr, int start, int end){
    if(start > end) return null;
    int mid = end - (end - start) / 2;
    TreeNode node = new TreeNode(arr[mid]);
    node.left = CreateMinimalBinarySearchTree(arr, start, mid-1);
    mode.right = CreateMinimalBinarySearchTree(arr, mid+1, end);
}
```

3.5 Pow(x, n)

Implement pow(x, n).

https://leetcode.com/problems/powx-n/description/

- directly multiply n times x
- n can be negative => n>0 ? res : 1/res
- T - O(n)

```
public class Solution {
    public double MyPow(double x, int n) {
        if(n==0) return 1;
        if(x==1) return 1;
        if(x==0) return 0;
        int k = Math.Abs(n);
        double res = 1;
        while(k > 0){
            res = res * x;
            k--;
        }
        return n>0 ? res : 1/res;
    }
}
```

- INT_MIN is -2147483648, but 2147483647 is largest int, so special case here
- x^n = x^n/2 * x^n/2 * x (n divied half, n % 2 always be 1 or 0)
- 3^9 = 3^4 * 3^4 * 3, so like this, divide it and find 3^4, then give the result using recursion
- store the result of Pow(x, n/2)
- T - O(logn)

```
public class Solution {
    public double MyPow(double x, int n) {
        if(n==0) return 1;
        if(x==1) return 1;
        if(x==0) return 0;
        if(n == int.MinValue)
            return (1/HelperPow(x, Math.Abs(n+1))) * (1/x); // INT_MIN is
-2147483648, but 2147483647 is largest int
        double res = HelperPow(x, Math.Abs(n));
        return n>0 ? res : 1/res;
    }
    protected double HelperPow(double x, int n){
        if(n == 0) return 1;
        double temp = HelperPow(x, n/2);
        if(n%2 == 0)
            return temp*temp;
        else
            return temp*temp*x;
    }
}
```

3.6 Sqrt(x)

Implement int sqrt(int x).
Compute and return the square root of x.

- binary search, nice design
- T- O(lgn)
- Magic Number 0x5f3759df, not used here,

```
    int sqrt(int x){
        if(x<1) return 0;
        if(x==1) return 1;
        return helper(1,x,x);
    }
    int helper(int start, int end, int x){
        if(start>end){
            return start-1;
        }
        long mid = end-(end-start)/2;  // mid*mid will overflow
        if(mid*mid == x){
            return mid;
        }else if(mid*mid < x){
            helper(mid+1,end,x);
        }else{
            helper(start,mid-1,x);
        }
    }
```

3.7 Trailing Number of zeros in n!

Given an integer n, write a function that returns count of trailing zeroes in n!.

http://www.geeksforgeeks.org/count-trailing-zeroes-factorial-number/

- this is too math, ignore
- any number can be the result of prime numbers mutiplication 10 = 2* 5
- trailing 0 is coming from result*10, 10 is coming from 2*5, so finding count of 2 and 5
- number of 2 is more than 5, so count 5
- 28!, 28/5 is all number with last digit 5, but 25,75,125 have more 5s, so we need to divide 5,25,125...
- return when result<1

```
int findTrailingZeros(int  n)
{
    int count = 0;
    // Keep dividing n by powers of 5 and update count
    for (int i=5; n/i>=1; i *= 5)
          count += n/i;
    return count;
}
```

3.8 Recover Binary Search Tree

Two elements of a binary search tree (BST) are swapped by mistake.
Recover the tree without changing its structure.
Note:
A solution using O(n) space is pretty straight forward. Could you devise a constant space solution?
https://leetcode.com/problems/recover-binary-search-tree/description/

- use inorde dfs to find two nodes
- how to find? 1,3,4,8,10,12,14 ------- 1,3,12,8,10,4,14
    - dfs until cur < prev. so cur is 8, prev is 12, so first = prev
    - then cur is 4, prev is 10, so second = cur

```
class Solution {
    TreeNode first = NULL;
    TreeNode second = NULL;
    TreeNode last = new TreeNode(int.MinValue);

    void RecoverTree(TreeNode *root) {
        InorderDfs(root);
        // swap
        int temp = first.val;
        first.val = second.val;
        second.val = temp;
    }
    void InorderDfs(TreeNode* root) {
        if (root == null) return;
        inorderDfs(root->left);
        if (first == NULL && root.val < last.val) {
            first = last;
        }
        if (first != NULL && root.val < last.val) {
            second = root;
        }
        last = root;
        inorderDfs(root.right);
    }
}
```

3.9 Determine if a Binary Tree is a Binary Search Tree (BST)

https://articles.leetcode.com/determine-if-binary-tree-is-binary/

totallay 3 solutions:

- brute force
    - for each node, check all of its left children and children of chilren are less than it, same for right
    - T - O(n^2)
- range check

    - pre-order traversal (parent always before children)
    - pass min and max value for each node
    - T - O(n)
- In-Order traversal
    - as long as each node is less than next node during in-order traversal, it is a valid BST

```
// range check
public bool IsBST(TreeNode root){
    long min = int.MinValue - 1;
    long max = int.MaxValue + 1;
    IsBST(root, min, max);
}
protected bool IsBST(TreeNode node, long min, long max){
    if(node.val > min && node.val < max)
        return IsBST(node.left, min, node.val)
               && IsBST(node.right, node.val, max);
}
```

4 Binary Tree

4.1 serialize binary tree and deserialize it

```
// serialize:  preorder traversal, if NULL, print "#"

public string SerializeBT(Node root){
        IList<int> list = new List<int>();
    SerializeBT(root, list);
    string res = null;
    foreach(int v in list)
        res = res + " " + v.ToString();
    return res;
}

protected void SerializeBT(Node node, IList<int> list){
    if (node == null){
        list.Add('#');
                return;
        }
    list.Add(node.value);
        SerializeBT(node.left, list);
        serializeBT(root.right, list);
}
```

```
public Node ReadBinaryTree(string serialBT){
    int num;
    List<string> list = serialBT.Split(' ').ToList();
    if(list.Count() == 0) return null;
```

```
    return ReadBinaryTree(list, 0);
}

protected Node ReadBinaryTree(List<string> list, int pos){
    if(list[pos] == "#" || pos >= list.Count()){
        return null;
    }
    Node node = new Node(int.Parse(list[pos]));
    node.left = ReadBinaryTree(list, pos+1);
    node.right = ReadBinaryTree(list, pos+1);
    return node;
}
```

4.2 construct binary tree from preorder and inorder traversal

// ignore

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

    TreeNode* helper(vector<int>& inorder, int in_start, int in_end, vector<int>&
preorder, int pre_start, int pre_end ){
        if(in_start > in_end) return NULL;
        int value = preorder[pre_start];
        int position = mymap[value];
        TreeNode* root = new TreeNode(value);
        root->left = helper(inorder, in_start, position-1, preorder, pre_start+1,
pre_start+position-in_start);
        root->right = helper(inorder, position+1, in_end, preorder, pre_end-
(in_end-position)+1, pre_end);
        return root;
    }

    void storeIndex(vector<int>& inorder){
        for(int i=0;i<inorder.size();i++){
            mymap[inorder[i]] = i;              // because no duplicates
        }
    }
```

4.3 construct binary tree from postorder and inorder traversal

// ignore

```
    TreeNode* helper(vector<int>& inorder, int in_start, int in_end, vector<int>&
postorder, int post_start, int post_end ){
        if(in_start > in_end) return NULL;
        int value = postorder[post_end];    // last is root
        int position = mymap[value];
        TreeNode* root = new TreeNode(value);
        root->left = helper(inorder, in_start, position - 1, postorder,
post_start, post_start + position - in_start - 1);//diff
        root->right = helper(inorder, position + 1, in_end, postorder, post_start
+ position - in_start, post_end - 1); //diff
        return root;
    }
```

4.4 Binary Tree Maximum Path Sum

Given a binary tree, find the maximum path sum.
The path may start and end at any node in the tree.
For example:
Given the below binary tree,
1
/ \
2 3
Return 6.

- notice that it starts from any node in the tree, so 2,1,3 is valid, result is 6
- use postorder dfs, for each node, check its left and right, then return the max sum include this current node. at same time and store real max_s as the result.

```
    int maxPathSum(TreeNode *root) {
        int max_s = INT_MIN;
        dfs(root, max_s);
        return max_s;
    }
    int dfs(TreeNode *root, int& max_s){
        if (!root) {return 0;}
        int l = dfs(root->left, max_s);
        int r = dfs(root->right, max_s);
        int m = root->val;
        if (l>0) {m+=l;}
        if (r>0) {m+=r;}
        max_s = max(max_s,m); // store largest sum (cur->val+left+right) each time

        // for recursion here, we want return max sum when root is included, so we
can only
        // choose max(left,right) and max>0
        if (max(l,r)>0) { return (max(l,r)+root->val); }
        else {return root->val;}
    }
```

4.5 Balanced Binary Tree

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.

- just use height method to check left side and right side

```
    bool isBalanced(TreeNode *root) {
       if(!root) return true;
       return std::abs(height(root->left) - height(root->right)) <= 1
                && isBalanced(root->left)
                && isBalanced(root->right);
    }
    int height(TreeNode* root){
        if(!root) return 0;
        return std::max(height(root->left), height(root->right)) + 1;
    }
```

4.6 Binary Tree Preorder Traversal

```
    vector<int> preorderTraversal(TreeNode *root) {
        std::vector<int> vec;
        std::stack<TreeNode*> st;
        if(!root) return vec;
        st.push(root);
        while(!st.empty()){
            TreeNode* node = st.top();
            st.pop();
            vec.push_back(node->val);
            if(node->right) st.push(node->right);  // notice here right before
left
            if(node->left) st.push(node->left);
        }
        return vec;
    }
```

4.7 Binary Tree Inorder Traversal

- recusive version
- iterative version

```
  // recursive straightforward to implement
    vector<int> inorderTraversal(TreeNode* root){
        vector<int> vec;
        inorderTraversal(root,vec);
        return vec;
```

```
    }
    void inorderTraversal(TreeNode* root, vector<int>& vec){
        if(!root) return;
        inorderTraversal(root->left, vec);
        vec.push_back(root->val);
        inorderTraversal(root->right, vec);
    }
    vector<int> inorderTraversal(TreeNode *root) {
        vector<int> vec;
        if(!root) return vec;
        std::stack<TreeNode*> st;
        TreeNode* cur = root;
        while(true){
            if(cur){
                st.push(cur);
                cur = cur->left;
            }else{
                if(!st.empty()){
                    TreeNode* node = st.top();
                    st.pop();
                    vec.push_back(node->val);
                    cur = node->right;
                }else{
                    break;
                }
            }
        }
        return vec;
    }
```

4.8 Binary Tree Postorder Traversal

```
    // two stack way is easy to understand but use more space
    // one stack is better and compatible with preorder traversal
    vector<int> postorderTraversal(TreeNode *root) {
        vector<int> vec;
        if(!root) return vec;
        std::stack<TreeNode*> st;
        st.push(root);
        TreeNode* temp = root;  // temp can be any other than NULL
        while(!st.empty()){
            TreeNode* node = st.top();
            if( (!node->left && !node->right) || node->left == temp || node-
>right==temp){  // leaf or its parent of prev one
                vec.push_back(node->val);
                temp = node;
                st.pop();
            }else{
                if(node->right) st.push(node->right);
                if(node->left) st.push(node->left);
            }
```

```
        }
        return vec;
    }
```

5 array

5.1 Challenge 3: Hill --- hired.com

Given an array of integer elements
Your task is to
write a function that finds the minimum value X that makes possible the following: generate a new array that is sorted in strictly ascending order by increasing or decreasing each of the elements of the initial array with integer values in the [0, X] range.
Example: Having the initial array [5, 4, 3, 2, 8], get [2, 3, 4, 5, 8] which is sorted in strictly ascending order.
***(e.g. [1, 2, 2, 3] is NOT strictly ascending order)

- this is a logically complex one, but solution is clear and easy
- -x to +x means find 2x that can sort the array with addition only (result / 2 will be final result)
- 2x starts from 0, if cur > prev with 2x, then 2x is good
- if cur < prev even with 2x, then cur need to be prev + 1
- notice that each node just need to be larger than previous node by 1
- some hired.com problems: https://github.com/cirosantilli/notes/blob/master/motivation/hired.com.md

```
protected int FindMinXForArray()
{
    int[] arr = new int[] { 5, 4, 3, 2, 8};
    if (arr.Length <= 1) return 0;
    int x2 = 0;
    int prev = arr[0];
    for(int i=1; i<arr.Length; i++)
    {
        if(arr[i] + x2 > prev)
        {
            prev = arr[i];
        }
        else
        {
            prev++;
            x2 = prev - arr[i];
        }
    }
    double res = Math.Ceiling((double)x2 / 2);
    Console.WriteLine(res);
    return (int)res;
}
```

5.2 First Missing Positive

Given an unsorted integer array, find the first missing positive integer.
For example,
Given [1,2,0] return 3,
and [3,4,-1,1] return 2.
Your algorithm should run in O(n) time and uses constant space.

- actually, it starts from 1,2,3....n; find first one missing in this array. so {7,8,9} -> 1
- so we can just use a map to mark all elements, then traverse from 1 to array.length, this is O(n) memory solution
- for constant solution, need override original array's not useful elements

```
    int firstMissingPositive(int A[], int n) {
        if(n==0) return 1;
        std::map<int,bool> mp;
        for(int i=0; i<n; i++){
            mp[A[i]] = true;
        }
        for(int i=1; i<=n; i++){
            if(mp.find(i)==mp.end()){
                return i;
            }
        }
        return n+1;
    }
```

For O(1) extra memory solution

- swap all valid element to their final position, then traverse and find which position is not valid (A[i] != i+1)
- do this loop for each one, because if its on its final position, we break, so T-O(n)

```
    int firstMissingPositive(int A[], int n) {
        for (int i = 0; i<n;i++){
            if ( (A[i]>0)&&(A[i]<=n)){
                while (A[i]!=(i+1)){
                    if ((A[i]>n)||(A[i]<=0)||(A[A[i]-1]==A[i])) {break;}// break
if invalid or in final pos
                    int tmp;
                    tmp = A[A[i]-1];
                    A[A[i]-1]=A[i];
                    A[i]=tmp;
                }
            }
        }

        int i=0;
        for (; i<n;i++){
            if (A[i]!=(i+1)){
                return i+1;
```

```
            }
        }
        return i+1;
    }
```

5.3 Candy

There are N children standing in a line. Each child is assigned a rating value.
You are giving candies to these children subjected to the following requirements:
Each child must have at least one candy.
Children with a higher rating get more candies than their neighbors.
What is the minimum candies you must give?

- neighbors mean left and right side two neighbor children, not all other children
- if two have same ratings, no need to have same candy, give as less as possible
- like raings 1,2,2 , we give 1,2,1 candy, because last 2' neighbor is 2, so we give it 1
- solution:
    - just left traverse and right traverse to find final sum of candies
    - when right traverse, choose the max(left[i],right[i])

```
    int candy(vector<int> &ratings) {
        if( ratings.size() == 0 ) return 0;
        vector<int> lc(ratings.size(),1);  // default can be 1
        vector<int> rc(ratings.size(),1);
        int res=0;
        for (int i=1;i<lc.size();i++){
            if (ratings[i]>ratings[i-1]){
                lc[i]=lc[i-1]+1;
            }
        }
        rc[rc.size()-1] = lc[lc.size()-1];
        res = rc[rc.size()-1];
        for (int i=rc.size()-2;i>=0;i--){
            if (ratings[i]>ratings[i+1]){
                rc[i] = rc[i+1] + 1;
            }
            res += std::max(lc[i],rc[i]);
        }
        return res;
    }
```

5.4 Max Points on a Line

Given n points on a 2D plane, find the maximum number of points that lie on the same straight line.

- use two loop, and point[i] is start point, then calculate each slop start from this point. store slopes in map with number of points with same slop
- use double because slop is a result of division, care precision

- second loop, we do not need start from 0, because if any point before point[j] is in a same line, previous loops have cover them.
- take care if all points are the same, map will be empty

```
    int maxPoints(vector<Point> &points) {
        if (points.size() == 0) {
            return 0;
        }
        std::map<double,int> mp;
        int max = 1;
        for(int i = 0 ; i < points.size(); i++) {
            mp.clear();   // clear each time
            int dup = 0;  // duplicates
            for(int j = i + 1; j < points.size(); j++) {   // start from i+1
                if (points[j].x == points[i].x && points[j].y == points[i].y) {
                    dup++;
                    continue;
                }
                // if the line through two points are parallel to y coordinator,
then K(slop) is
                // Integer.MAX_VALUE
                double key = points[j].x - points[i].x == 0 ?
                    INT_MAX :
                    (double)(points[j].y - points[i].y) / (double)(points[j].x -
points[i].x);
                if (mp.find(key) != mp.end()) {
                    mp[key]++;
                } else {
                    mp[key] = 2;
                }
            }
            if( mp.size()==0 ) max = std::max(1+dup,max);  // if all points are
the same
            for (auto item : mp) {
                // duplicate may exist
                max = std::max(item.second+dup,max);
            }
        }
        return max;
    }
```

5.5 Rotated Digits

X is a good number if after rotating each digit individually by 180 degrees, we get a valid number that is different from X. Each digit must be rotated - we cannot choose to leave it alone.

A number is valid if each digit remains a digit after rotation. 0, 1, and 8 rotate to themselves; 2 and 5 rotate to each other; 6 and 9 rotate to each other, and the rest of the numbers do not rotate to any other number and become invalid.

Now given a positive number N, how many numbers X from 1 to N are good?

Example: Input: 10 Output: 4 Explanation: There are four good numbers in the range [1, 10] : 2, 5, 6, 9. Note that 1 and 10 are not good numbers, since they remain unchanged after rotating. Note:

N will be in range [1, 10000]
[https://leetcode.com/problems/rotated-digits/description/]

- 2, 5, 6, 9 make a number valid and different with original one
- 3, 4, 7 exists then invaid
- 0, 1, 8 is valid but cannot make number different

```
public class Solution {
    public int RotatedDigits(int N) {
        List<int> validRotateNums = new List<int>() {2, 5, 6, 9};
        List<int> invalidRotateNums = new List<int>() {3, 4, 7};
        int num = 1;
        int count = 0;
        while(num <= N){
            int curNum = num;
            bool hasValidDigit = false;
            while(curNum > 0){
                int curDigit = curNum % 10;
                curNum = curNum / 10;
                if(invalidRotateNums.Contains(curDigit)){
                   hasValidDigit = false;
                   break;
                }
                else if(validRotateNums.Contains(curDigit)){
                   hasValidDigit = true;
                }
            }
            if(hasValidDigit) count++;
            num++;
        }
        return count;
    }
}
```

5.6 Valid Tic-Tac-Toe State

A Tic-Tac-Toe board is given as a string array board. Return True if and only if it is possible to reach this board position during the course of a valid tic-tac-toe game.

The board is a 3 x 3 array, and consists of characters " ", "X", and "O". The " " character represents an empty square.

Here are the rules of Tic-Tac-Toe:

Players take turns placing characters into empty squares (" "). The first player always places "X" characters, while the second player always places "O" characters. "X" and "O" characters are always placed into empty squares, never filled ones. The game ends when there are 3 of the same (non-empty) character filling any row,

column, or diagonal. The game also ends if all squares are non-empty. No more moves can be played if the game is over.

```
Example 1:
Input: board = ["O  ", "   ", "   "]
Output: false
Explanation: The first player always plays "X".

Example 2:
Input: board = ["XOX", " X ", "   "]
Output: false
Explanation: Players take turns making moves.

Example 3:
Input: board = ["XXX", "   ", "OOO"]
Output: false

Example 4:
Input: board = ["XOX", "O O", "XOX"]
Output: true
Note:
```

board is a length-3 array of strings, where each string board[i] has length 3. Each board[i][j] is a character in the set {" ", "X", "O"}. [https://leetcode.com/problems/valid-tic-tac-toe-state/description/]

- just need to cover all win, lose, valid, invalid cases

```
public class Solution {
    public bool ValidTicTacToe(string[] board) {
        // can only have X and O
        // starts from X
        // X always more than O by 1
        // If already win, no more
        // if all full 3*3, then no more
        if(board.Length != 3) return false;
        int countX = 0;
        int countO = 0;
        bool win = false;
        bool xWin = false;
        foreach(string s in board){
            if(s.Length != 3) return false;
            foreach(char c in s){
                if(c != 'X' && c != 'O' && c != ' ')
                    return false;
                else if (c == 'X') countX++;
                else if( c== 'O') countO++;
            }
            if(s == "XXX" || s == "OOO"){
              if(win) return false;
              win = true;
```

```
                if(s == "XXX") xWin = true;
            }
        }
        if(countX != countO && countX != (countO + 1)) return false;
        for(int i=0; i<3; i++){
            string row = board[0][i].ToString() + board[1][i].ToString() +
board[2][i].ToString();
            if(row == "XXX" || row == "OOO"){
              if(win) return false;
              win = true;
              if(row == "XXX") xWin = true;
            }
        }
        string row1 = board[0][0].ToString() + board[1][1].ToString() + board[2]
[2].ToString();
        string row2 = board[0][2].ToString() + board[1][1].ToString() + board[2]
[0].ToString();
        if(row1 == "XXX" || row2 == "OOO" || row1 == "OOO" || row2 == "XXX"){
              if(win) return false;
              win = true;
              if(row1 == "XXX" || row2 == "XXX") xWin = true;
        }
        if(xWin && countX != (countO + 1)) return false;
        if(win && !xWin && countX != countO) return false;
        return true;
    }
}
```

5.7 Number of Subarrays with Bounded Maximum

We are given an array A of positive integers, and two positive integers L and R (L <= R).

Return the number of (contiguous, non-empty) subarrays such that the value of the maximum array element in that subarray is at least L and at most R.

Example : Input: A = [2, 1, 4, 3] L = 2 R = 3 Output: 3 Explanation: There are three subarrays that meet the requirements: [2], [2, 1], [3]. Note:

L, R and A[i] will be an integer in the range [0, 10^9]. The length of A will be in the range of [1, 50000].

[https://leetcode.com/problems/number-of-subarrays-with-bounded-maximum/description/]

- Just need to starts from one begin index, keep the current max, if cur max is between L and R, then count++.
- if cur<L, then go to next element
- if cur>R, then begin index ++

```
public class Solution {
    public int NumSubarrayBoundedMax(int[] A, int L, int R) {
        if(L > R) return -1;
        if(A == null || A.Length == 0) return 0;
```

```
        int i = 0;
        int count = 0;
        int max = 0;
        int begin = 0;
        while(begin < A.Length){
            i = begin;
            while(i < A.Length){
                 max = Math.Max(max, A[i]);
                 if(max >= L && max <= R){
                    count++;
                    i++;
                 }
                 else if (max < L) i++;
                 else if(max > R) break;
            }
            max = 0;
            begin++;
        }
        return count;
    }
}
```

5.8 binary search in ordered array

- regular binary search algorithm
- array is alreayd sorted, so pick mid, split in two half in recursive
- T - O(logn)

```
public int SearchInArray(int[] arr, int value){
    if(arr == null) return -1;
    return SearchInArray(arr, 0, arr.Length, value);
}
protected int SearchInArray(int[] arr, int start, int end, int value){
    if(start > end) return -1;
    int mid = end - (end-start) / 2; // same as (start + end) / 2, avoid integer
overflows
    if(arr[mid] == value) return mid;
    else if(arr[mid] > value) return SearchInArray(arr, start, mid-1);
    else if(arr[mid] < value) return SearchInArray(arr, mid+1, end);
}
```

If the value is not in the array, return the correct index

```
public int SearchInArray(int[] arr, int value){
    if(arr == null) return -1;
    return SearchInArray(arr, 0, arr.Length, value);
}
protected int SearchInArray(int[] arr, int start, int end, int value){
    // it will always point to same element, then do some special case handling.
```

```
    if(start == end){
        if(start == 0) return 0;
        else if(arr[start] > value) return start -1;
        else return start;
    }
    int mid = end - (end-start) / 2; // same as (start + end) / 2, avoid integer
overflows
    if(arr[mid] == value) return mid;
    else if(arr[mid] > value) return SearchInArray(arr, start, mid-1);
    else if(arr[mid] < value) return SearchInArray(arr, mid+1, end);
}
```

6 graph

```
class Vertex{
private:
        int val;
        vector<Edge*> edges;
public:
        Vertex(int val):val(val){}
        void addEdge(Vertex* target){
                Edge* edge = new Edge(this, target);
                edges.push_back(edge);
        }
};

class Edge{
private:
        Vertex* source;
        Vertex* target;
public:
        Edge(Vertex* source, Vertex* target):source(source),target(target){}
        Vertex* getTarget(){
                return target;
        }
};

class Graph{
private:
        vector<Vertex*> vertices;
public:
        vector<Vertex*> getVertices(){
                return vertices;
        }
        void DFS();
        void BFS();
};
```

6.1 social network, try to suggest new friend based on your existed friend to yourself --- Youtube

- your friends' friends contains yourself, you will always return yourself
- you may return a new friend that was already my friend
- use one set and one map, set store me and my friends, another one count friends and map can be used to avoid suggest my existed friends or myself

```
struct Person{
  int ID;
  string name;
  vector<Person*> friends;
};

Person* SuggestNewFriend(Person* person){
    map<Person*,int> mymap;
    set<Person*> myset;
    int max = 0;
    Person* res=NULL;
    myset.insert(person);
    for(int i=0;i<person->friends.size();i++)
        myset.insert(person->friends[i]);

    for(int i=0;i<person->friends.size();i++){
Person *p = person.friends[i];
for(auto per : p->friends){
   if(myset.find(per) != myset.end()) continue;
        if(mymap.find(per)== mymap.end()){
                mymap[per] = 1;
                if(max <= 1){
                        max = 1;
                        res = per;
                }
        }else{
                mymap[per]++;
                if(max < mymap[per]){
                        max = mymap[per];
                        res = per;
                }
        }
       }

     }

    map<int,Person*> mymap2;      // sorted by counts, can return all sugested
friend
    for(auto* per : mymap){
        mymap2[per->second] = per->first;
    }
    return person_1;
}
```

6.2 buid a graph, then dfs and bfs it

- build graph

```
struct Node{
        int val;
        vector<Node*> children;
        Node(int val): val(val){}
};  // remember to add semicolon here always after struct and class
int main(int argc, char *argv[]) {
        Node* node0 = new Node(0);
        Node* node1 = new Node(1);
        Node* node2 = new Node(2);
        Node* node3 = new Node(3);
        Node* node4 = new Node(4);
        Node* node5 = new Node(5);
        Node* node6 = new Node(6);
        Node* node7 = new Node(7);
        node0->children.push_back(node1);
        node0->children.push_back(node2);
        node0->children.push_back(node3);
        node7->children.push_back(node4);
        node7->children.push_back(node5);
        node7->children.push_back(node6);
        node0->children.push_back(node7);
        Solution sl;
        sl.BFS(node0);
        cout << endl;
        sl.DFS(node0);
        cout << endl;
}
```

- DFS standard – recursion

```
void DFS(Node* node){
        if(!node) return;
        std::map<Node*, bool> visited;
        DFSHelper(node,visited);
}
void DFSHelper(Node* node, map<Node*,bool>& visited){
        if(!node) return;
        visit(node);
        visited[node] = true;
        for(int i=0;i<node->children.size(); i++){
                if(visited.find(node->children[i]) == visited.end() ){
                        DFSHelper(node->children[i],visited);
                }
        }
}
```

- BFS standard -- use queue

```
//note: need to BFS each node of graph to gurantee traverse all nodes, use the
same map
void BFS(Node* node){
        if(!node) return;
        std::queue<Node*> q;
        std::map<Node*,bool> visited;
        q.push(node);
        visited[node] = true;
        visit(node);
        while(!q.empty()){
                Node* cur = q.front();
                q.pop();
                for(int i=0; i<cur->children.size();i++){
                        vector<Node*>& children = cur->children;
                        if(visited.find(children[i]) == visited.end()){
                                q.push(children[i]);
                                visited[children[i]] = true;
                                visit(children[i]);
                        }
                }
        }
}
```

6.3 Clone Graph

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

- similar with copy random pointer of list problem
- first add all original nodes into vec, and build map<original, copy> -- BFS, DFS is also ok
- then traverse vec and use map to build new graph one node by one node
- do not need check cycle, visited map will avoid all cycles

```
    UndirectedGraphNode *cloneGraph(UndirectedGraphNode *node) {
        if(!node) return NULL;
        std::map<UndirectedGraphNode*, UndirectedGraphNode*> mymap;
        vector<UndirectedGraphNode*> vec;
        std::queue<UndirectedGraphNode*> q;
        // BFS traverse graph and add all new nodes to vec
        q.push(node);
        mymap[node] = new UndirectedGraphNode(node->label);
        vec.push_back(node);
        while(!q.empty()){
            UndirectedGraphNode* cur = q.front();
            q.pop();
            for(int i=0; i<cur->neighbors.size(); i++){
                if( mymap.find(cur->neighbors[i]) == mymap.end() ){
                    vec.push_back(cur->neighbors[i]);
                    UndirectedGraphNode* new_node = new
UndirectedGraphNode(cur->neighbors[i]->label);
                    mymap[cur->neighbors[i]] = new_node;
```

```
                    q.push(cur->neighbors[i]);
                }
            }
        }
        // traverse vec and build the new graph
        for(int i=0; i<vec.size(); i++){
            UndirectedGraphNode* new_node = mymap[vec[i]];
            for(int j=0; j<vec[i]->neighbors.size(); j++){
                new_node->neighbors.push_back(mymap[vec[i]->neighbors[j]]);
            }
        }
        return mymap[vec[0]];
    }
```

6.4 topological sorting

- order may differ, think all nodes are .h files, order them based on compile order
- steps:
  - use BFS to store map<Node*, int> Node with number of in-degrees
  - use BFS again to print all node with 0 in-degree
  - so, after two BFS, result will be: 5 2 3 1 4 0
- we must have a vectoe<Node*> to contain all nodes in graph first, then we start with any node in the vector.
- we need visited<Node*, bool> to indicate that node is visited or not

```
// will write after awhile
//http://www.geeksforgeeks.org/topological-sorting/
```

6.5 letter sort

alphabet dictionary: a-z
we have following words (inputs):
abe
ec
db
try to find there inner order.
output: adbec

- same with topological sort, just create a acyclic directed graph
- then BFS two times to give output order based on input order

```
// will add code after awhile
```

6.6 detect cycle in undirected graph

http://www.geeksforgeeks.org/detect-cycle-in-a-graph/
Tarjan algorithm for finding all strongly connected component (cycles):
http://stackoverflow.com/questions/546655/finding-all-cycles-in-graph

```
// A C++ Program to detect cycle in a graph using DFS
#include<iostream>
#include <list>
#include <limits.h>
using namespace std;

class Graph
{
    int V;    // No. of vertices
    list<int> *adj;    // Pointer to an array containing adjacency lists
    bool isCyclicUtil(int v, bool visited[], bool *rs);  // used by isCyclic()
public:
    Graph(int V);   // Constructor
    void addEdge(int v, int w);   // to add an edge to graph
    bool isCyclic();    // returns true if there is a cycle in this graph
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

// This function is a variation of DFSUytil() in
http://www.geeksforgeeks.org/archives/18212
bool Graph::isCyclicUtil(int v, bool visited[], bool *recStack)
{
    if(visited[v] == false)
    {
        // Mark the current node as visited and part of recursion stack
        visited[v] = true;
        recStack[v] = true;

        // Recur for all the vertices adjacent to this vertex
        list<int>::iterator i;
        for(i = adj[v].begin(); i != adj[v].end(); ++i)
        {
            if ( !visited[*i] && isCyclicUtil(*i, visited, recStack) )
                return true;
            else if (recStack[*i])   // back edge here
                return true;            // now recStack contains cycle vertices
        }
```

```
    }
    recStack[v] = false;  // remove the vertex from recursion stack
    return false;
}

// Returns true if the graph contains a cycle, else false.
// This function is a variation of DFS() in
http://www.geeksforgeeks.org/archives/18212
bool Graph::isCyclic()
{
    // Mark all the vertices as not visited and not part of recursion
    // stack
    bool *visited = new bool[V];
    bool *recStack = new bool[V];
    for(int i = 0; i < V; i++)
    {
        visited[i] = false;
        recStack[i] = false;
    }

    // Call the recursive helper function to detect cycle in different
    // DFS trees
    for(int i = 0; i < V; i++)
        if (isCyclicUtil(i, visited, recStack))
            return true;

    return false;
}

int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    if(g.isCyclic())
        cout << "Graph contains cycle";
    else
        cout << "Graph doesn't contain cycle";
    return 0;
}
```

6.7 find wheter there is a route between two nodes in a directed graph --- CTCI

6.8 Surrounded Regions

Given a 2D board containing 'X' and 'O', capture all regions surrounded by 'X'.
A region is captured by flipping all 'O's into 'X's in that surrounded region.
For example,
X X X X
X O O X
X X O X
X O X X
After running your function, the board should be:
X X X X
X X X X
X X X X
X O X X

- brute force: try go 4 diection from each element, flip if facing edge of matrix
- better solution: use BFS
    - create a new bool table to set all not captured O to true, so finally all falses are captured
    - from outer to inner side, so set top, left, right, bottom line Os to true in bool table, and use queue to store each true element.
    - pop each element in queue to check its top, bottom, left and right, set true if it is O, push it into queue
    - traverse bool table to flip Os when element in bool table is false

```
    void solve(vector<vector<char>> &board) {
        int row = board.size();  //get row number
        if (row==0){return;}
        int col = board[0].size(); // get column number
        vector<vector<bool> > bb(row, vector<bool>(col)); //result vector
        queue<pair<int,int> > q; // queue for BFS
        //search "O" from 1st row
        for (int i=0;i<col;i++){
            if (board[0][i]=='O'){
                q.push(make_pair(0,i));
                bb[0][i]=true;
            }
        }
        //search "O" from 1st column
        for (int i=0;i<row;i++){
            if (board[i][0]=='O'){
                q.push(make_pair(i,0));
                bb[i][0]=true;
            }
        }
        //search "O" from last row
        for (int i=0;i<col;i++){
            if (board[row-1][i]=='O'){
                q.push(make_pair(row-1,i));
                bb[row-1][i]=true;
            }
        }
        //search "O" from last column
```

```
        for (int i=0;i<row;i++){
            if (board[i][col-1]=='O'){
                q.push(make_pair(i,col-1));
                bb[i][col-1]=true;
            }
        }
        //BFS
        int i,j; // current position
        while (!q.empty()){
            //get current live "O"
            i = q.front().first;
            j = q.front().second;
            //pop up queue
            q.pop();
            //search four directions
            if (i-1>0 && board[i-1][j]=='O' && bb[i-1][j]==false){
                bb[i-1][j]=true; q.push(make_pair(i-1,j));
            } //top
            if (i+1<row-1 && board[i+1][j]=='O'&& bb[i+1][j]==false){
                bb[i+1][j]=true; q.push(make_pair(i+1,j));

            } // bottom
            if (j-1>0 && board[i][j-1]=='O'&& bb[i][j-1]==false){
                bb[i][j-1]=true; q.push(make_pair(i,j-1));
            } // left
            if (j+1<col-1 && board[i][j+1]=='O'&& bb[i][j+1]==false){
                bb[i][j+1]=true; q.push(make_pair(i,j+1));
            } // right
        }
        //Get result
        for (int i=0;i<row;i++){
            for (int j=0;j<col;j++){
                if (board[i][j]=='O'&&bb[i][j]==false){
                    board[i][j]='X';
                }
            }
        }
        return;
    }
```

7 bit operation

7.1 Sum of two integer without arithmetic operations

```
+ ++ - --...
```

note: divide in two parts

a^b is carry result; a&b is result with carry.

plus two result is final result; (no + operation, so while loop or recursion)

```
int func(int a, int b){
        while(b!=0){
                int carry a^b;
                a = a&b;
                b = carry<<1;
        }
        return a;
}       int func(int a, int b){
        if(b==0) return a;
        return func(a&b, (a^b)<<1);
}
```

7.2 multiple of two integer without arithmetic operation

```
+ ++ - --...
```

7.3 swap two integer without extra memory

- a = a^b; b=a^b; a=a^b;

7.4 fastest way to reverse all bits of an integer

create a lookup table, contain 256 possible byte, the second is 0x80, which is reverse of 0x01. then b[0] = table[a[0]] b[1] = table[a[1]]

```
unsigned int v; // reverse 32-bit value, 8 bits at time
unsigned int c; // c will get v reversed

unsigned char * p = (unsigned char *) &v;
unsigned char * q = (unsigned char *) &c;
q[3] = BitReverseTable256[p[0]];
q[2] = BitReverseTable256[p[1]];
q[1] = BitReverseTable256[p[2]];
q[0] = BitReverseTable256[p[3]];
```

7.5 Single Number

Given an array of integers, every element appears twice except for one. Find that single one.
Note:
Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

- use map, with O(n) extra memory
- best solution, using A XOR A = 0, A XOR A XOR B = B
- for xor, we know (a^b)^c == a^(b^c)
- if c=a^b; so b=a^c, a=b^c; which can be used when swaping two values

```
 int singleNumber(int A[], int n){
    int result = 0;
    for(int i=0;i<n;i++){
        result = result^A[i];
    }
    return result;
}
```

7.6 Single Number II

Given an array of integers, every element appears three times except for one. Find that single one.
Note:
Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

- sum of 1s or 0s of whole array of specific bit will be 3k+1(one is 1) or 3k(one is 0)
- T-O(32n) so O(n)

```
int singleNumber(int A[], int n) {
    int result=0;
    for(int i=0;i<32;i++){
        int temp = 0;
        for(int j=0;j<n;j++){
            temp += A[j] >> i & 1;
        }
        temp %= 3;
        result |= temp<<i;
    }
    return result;
}
```

7.7 Single Number III

Given an array of integers, every element appears twice except for two. Find the two singles.

- xor all nunmbers, get x
- find last 1 in x as follows
  lowbit = c - c & (c - 1)
  or = c & ~(c-1)
- so x&lowbit == 0 or 1 will divided numbers into two groups
- xor each group and two results of two group are two numbers we need

7.8 Majority Number

Given an array of integers, the majority number is the number that occurs more than half of the size of the array. Find it.
Example
For [1, 1, 1, 1, 2, 2, 2], return 1

Challenge Expand
O(n) time and O(1) space

- because the number of this integer is larger than half of lenght, so use a counter, if two if same, count++, if not same count--. finally max is the integer.

```
int main(){
    int cnt, max, cur, n;
        int arr[8] = {0,1,2,2,2,2,4,5};
        cnt = 0;
        for(int i=0; i<8; i++){
                cur = arr[i];
                if(!cnt)
                {
                        max = cur;
                        cnt++;
                }
                else if(cur != max) cnt--;    // different
                else if(cur == max) cnt++;  // same
        }
        cout << max << endl;
        return 0;
}
```

7.9 O(1) Check Power of 2

- (x-1) & x == 0 -> x is power of 2
- 000100000 only one 1 in binary representation, then x-1 will be all 1s.
- so & between (x-1)&x will be 0 if x is power of 2

8 Linked list

8.1 insert a node

a -> b -> NULL, insert c

- insert to head
  - c->next = a;
  - head = c;
- insert to before b, after a
  - use while(head->next) find a, then,
  - c->next = b; a->next = c;
- insert c to the end of list
  - while(head->next) find b, then
  - b->next = c;
  - c->next = NULL;

8.2 remove a node

a -> b -> c -> d

- remove c
  - b->next = b->next->next
- if already at node c, how to remove c?
  - c->val = c->next->val;
  - c->next = c->next->next;

8.3 dummy node as head

```
// use dummy node as head when the head of the list will possibly changed

Node* dummy = new Node(-1);
dummy->next = head;
head = dummy;
….  // some operations

head = dummy->next;
delete dummy;
dummy = NULL;
return head;
```

8.4 remove duplicates in sorted singly list (leave one of duplicated nodes)

```
  // if node->val == node->next->val, just node points to next next one is enough
   ListNode* deleteDuplicates(ListNode* head){
        if(!head) return NULL;
        ListNode* node = head;
        while(node && node->next){
            if(node->val == node->next->val){
                node->next = node->next->next;
            }else{
                node = node->next;
            }
        }
        return head;
    }
```

8.5 remove duplicates in sorted singly list (delete all dupliated nodes)

```
   // use dummy node as head, and compare head->next and head->next->next
  //  if two is equal, try delelte one by one to delete all duplicates
   ListNode* deleteDuplicates(ListNode* head){
          if(!head) return NULL;
          ListNode* dummy = new ListNode(-1);     // dummy node as head
          dummy->next = head;
          head = dummy;
```

```
        while(head && head->next && head->next->next){
            if(head->next->val == head->next->next->val){
                int temp = head->next->val;
                while(head->next && head->next->val == temp){   // delete one by
one, good!
                    head->next = head->next->next;
                }
            }else{
                head = head->next;
            }
        }
        head = dummy->next;
        delete dummy;       // always remember delete object
        dummy = NULL;
        return head;
    }
```

8.6 reverse a list

orignal: a -> b -> c ->null
| | |
prev cur next

output: null<-a <-b <- c

- reverse list is just make all pointers reverse. then the end is our new head
- prev, head, next. head points to prev, then move prev and head one step
- notice return prev, prev is the end of list and head now is NULL

```
Node* reverseList(Node* head){
        if(!head) return NULL;
        Node* prev = NULL;
        Node* next = NULL;
        while(head){
            next = head->next;   // store next
            head->next = prev;   // head points to prev

            prev = head;           // move one step forward
            head = next;
        }
        return prev;         // return prev
    }
```

8.7 Reverse Linked List II

Reverse a linked list from position m to n. Do it in-place and in one-pass.
For example:
Given 1->2->3->4->5->NULL, m = 2 and n = 4,

return 1->4->3->2->5->NULL.
Note:
Given m, n satisfy the following condition:

- ≤ m ≤ n ≤ length of list.

```
 // slow and fast pointer, interval is m-n, then go same step until slow pointer
at first position
 // use dummy node as head, and cut the sublist and then use common reverse skill
to reverse it, then with the help of dummy node, connect sublist with original
list

 // always remember test cases, which is so important that you cannot forget!
    ListNode *reverseBetween(ListNode *head, int m, int n) {
        if(!head) return NULL;
        if (m>=n) return head;
        ListNode* slow = head;
        ListNode* fast = head;

        ListNode* dummy = new ListNode(-1);
        dummy->next = head;
        head = dummy;

        //fast go forward
        int count = n-m;
        while(count != 0){
            if(!fast) return dummy->next;
            else fast = fast->next;
            count--;
        }

        // slow and fast go together
        while(m > 1){
            if(!slow || !fast) return dummy->next;
            head = head->next;
            slow = slow->next;
            fast = fast->next;
            m--;
        }
        // reverse
        ListNode* prev = NULL;
        ListNode* next = NULL;
        ListNode* end = fast->next;     // need to store end, otherwise it will
        while(slow != end){                  // be changed
            next = slow->next;
            slow->next = prev;

            prev = slow;
            slow = next;
        }

        // connect
```

```
        head->next->next = slow;
        head->next = prev;

        head = dummy->next;
        delete dummy;
        dummy = NULL;
        return head;        // caution return head here
    }
```

8.8 merge two list

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.

- create a new dummy node, as the head of new list, then normal merge two list

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     public int val;
 *     public ListNode next;
 *     public ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode MergeTwoLists(ListNode l1, ListNode l2) {
        ListNode dummyNode = new ListNode(-1);
        ListNode node = dummyNode;
        while (l1 != null && l2 != null)
        {
            if (l1.val <= l2.val)
            {
                node.next = l1;
                l1 = l1.next;
                node = node.next;
            }
            else
            {
                node.next = l2;
                l2 = l2.next;
                node = node.next;
            }
        }
        while (l1 != null)
        {
            node.next = l1;
            node = node.next;
            l1 = l1.next;
        }
        while (l2 != null)
        {
```

```
            node.next = l2;
            node = node.next;
            l2 = l2.next;
        }
        return dummyNode.next;
    }
}
```

8.9 find middle of list

- double pointer: slow and fast pointer, fast goes two step each time, slow goes one step each time, when fast reach the end, slow is at the middle of list
- can be used in mergesort of list to find the mid node of sublist
- T - O(n/2)

```
   ListNode FindMiddle(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while(fast != null && fast.next != null && fast.next.next != null){
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
```

8.10 find or remove the nth node from the end of list

- slow and fast pointer, fast goes n node at first, then slow and fast goes together, when fast is at the end, slow is at the nth node from the end of list

8.11 detect is there cycle in a linked list

- two pointer, slow goes one step, fast goes two step each time, return when they meet
- we can prove it: if slow->fast, go back they will meet. if fast->slow, go forward will meet
- if fast goes three steps each time, plus check slow->next is enough
- use hash map is another using extra space way
- this hasCycle will be used in next problem

```
void hasCycle(ListNode *head, ListNode*& dup) {
        if (!head) return;
        ListNode* first = head;
        ListNode* second = head;
        while (first && second){
                if (first->next) first = first->next;
                else return ;
                if (second->next && second->next->next) second = second->next-
>next;
                else return ;
```

```
                if (first == second){
                        dup = first;
                        return;
                }
        }
}
```

8.12 if there is cycle, return the begin of cycle

- hasCycle return meet point, then start from head and this meet point, return node when these two nodes meet.
- prove: slow go k step to begin of loop, so fast goes 2k steps, and is k step before slow, and also k steps inside loop. then fast want cach up slow, need loopsize-k times, so now slow is k steps to the begin of loop. and k is module of loopsize. so head to begin is also k steps. they go together will meet at the begin of loop

```
        ListNode *detectCycle(ListNode *head) {
                ListNode* dup = NULL;    // meet point return from hasCycle
                hasCycle(head,dup);
                if (dup){
                        while (head != dup){
                                head = head->next;
                                dup = dup->next;
                        }
                        return head;
                }
                else{
                        return NULL;
                }
        }
```

8.13 Reorder List

Given a singly linked list L: L0→L1→…→Ln-1→Ln,
reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…
You must do this in-place without altering the nodes' values.
For example,
Given {1,2,3,4}, reorder it to {1,4,2,3}.

- find mid of list
- 2 reverse second half of list
- 3 merge two sublist

```
    public void reorderList(ListNode head) {
        if (head == null || head.next == null) {
            return;
        }
```

```
        ListNode mid = findMiddle(head);     // same with find middle code snippit
        ListNode tail = reverse(mid.next);
        mid.next = null;
        merge(head, tail);
    }

    private ListNode reverse(ListNode head) {
        ListNode newHead = null;
        while (head != null) {
            ListNode temp = head.next;
            head.next = newHead;
            newHead = head;
            head = temp;
        }
        return newHead;
    }

    private void merge(ListNode head1, ListNode head2) {
        int index = 0;
        ListNode dummy = new ListNode(0);
        while (head1 != null && head2 != null) {
            if (index % 2 == 0) {
                dummy.next = head1;
                head1 = head1.next;
            } else {
                dummy.next = head2;
                head2 = head2.next;
            }
            dummy = dummy.next;
            index ++;
        }
        if (head1 != null) {
            dummy.next = head1;
        } else {
            dummy.next = head2;
        }
    }
```

8.14 sort singly linked list

- use merge sort alogrithm nlog(n) time complexity
- slow and fast pointer to find the middle of sublist each time
- merge two sublist does not need extra space comared with mergesort of array

```
    // mergesort with constant memory, and nlog(n) time complexity
    // return node* as a sublist
    ListNode *sortList(ListNode *head) {
        if(!head) return head;
        if(!head->next) return head;    // act as left>right in array
        ListNode* mid = findMiddle(head);
        ListNode* right = sortList(mid->next);
```

```
        mid->next = NULL;               // break the list ahead
        ListNode* left = sortList(head);
        return merge(left, right);
    }
    ListNode* findMiddle(ListNode* head){
        ListNode* slow =head;
        ListNode* fast = head;
        while(fast && fast->next && fast->next->next){
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
    ListNode* merge(ListNode* left, ListNode* right){
        ListNode* dummy = new ListNode(-1);
        ListNode* cur = dummy;
        while(left && right){
            if(left->val < right->val){
                cur->next = left;
                left = left->next;
            }else{
                cur->next = right;
                right = right->next;
            }
            cur = cur->next;
        }
        if(left){
            cur->next = left;
        }
        if(right){
            cur->next = right;
        }
        cur = dummy->next;
        delete dummy;
        dummy = NULL;
        return cur;
    }
```

8.15 partion list

```
    // sublist will only use constant memory, just do it
    // create two list, left < x, right >= x
ListNode* partition(ListNode* head, int x){
        if(!head) return NULL;
        ListNode* left = new ListNode(-1);
        ListNode* right = new ListNode(-1);
        ListNode* l_cur = left;
        ListNode* r_cur = right;
        while(head){
            if(head->val < x){
                l_cur->next = head;
```

```
                l_cur = l_cur->next;
            }else{
                r_cur->next = head;
                r_cur = r_cur->next;
            }
            head = head->next;
        }
        r_cur->next = NULL;         // break the original list
        l_cur->next = right->next;
        head = left->next;
        delete left;
        delete right;
        left = NULL;
        right = NULL;
        return head;
    }
```

8.16 Merge k Sorted Lists

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

- T- O(nk*log(k)), where k is number of list and n is the length of each list
- each heapify need at most log(k), and there are n*k nodes need to be heapify
- S - O(k) --- priority_queue size is k
- we can also merge by pairs and the T is also O(nklog(k)) -- log(k) time whole merge, and each whole merge will merge all nodes which is n*k, so its O(nklog(k))

```
    struct Compare{
        bool operator()( ListNode* lhs, ListNode* rhs){
            return lhs->val > rhs->val;
        }
    };

   ListNode *mergeKLists(vector<ListNode *> &lists) {
        if(lists.size() == 0) return NULL;
        std::priority_queue<ListNode*,vector<ListNode*>, Compare> pq;   // create
heap, see how to declare
        for(int i=0; i<lists.size(); i++){
            if(lists[i]){           // check NULL
                pq.push(lists[i]);
            }
        }
        // merge
        ListNode* dummy = new ListNode(-1);
        ListNode* cur = dummy;
        while(!pq.empty()){
            ListNode* top = pq.top();
            pq.pop();
            cur->next = top;
            cur = cur->next;
            if(top->next){              // push next, and heapify the p_q
```

```
                pq.push(top->next);
            }
        }
        cur = dummy->next;
        delete dummy;
        dummy = NULL;
        return cur;
    }
```

8.17 deep copy of singly list

- each time do tail->next = node where node is the newly copied node, just one pass
- doubly linked list is the same, one pass, connect prev and next

```
ListNode* copyList(Node* head){
    if(!head) return NULL;
    Node* dummy = new Node(-1);
    Node* tail = dummy;
    while(head){
        Node* node = new Node(head->label);
        tail->next = node;
        tail = tail->next;
        head = head->next;
    }
    return dummy->next;
}
```

8.18 Copy List with Random Pointer

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.

- do like single linked list first --- copy next, and store each node in map pair (original, copy)
- use map to find copied node and assign to random pointer
- T-O(n), S- O(n) using map

another T-O(n), S-O(1) approach:
a->a'->b->b'->c->c'->NULL

- insert copied node after each original node, so next pointer will be ok after split
- because copied node is after each original node, we have:
- head->next->random = head->random->next (finish random pointer)
  we can assign a->random->next (c') to a->next->random
- split copied list from mix list

```
    // T-O(n), S- O(n) using map
    RandomListNode *copyRandomList(RandomListNode *head) {
```

```
        if(!head) return NULL;
        std::map<RandomListNode*, RandomListNode*> mymap;
        vector<RandomListNode*> vec;
        RandomListNode* cur = head;
        RandomListNode* dummy = new RandomListNode(-1);
        RandomListNode* tail = dummy;
        while(cur){              // conect all next pointers in this loop
            RandomListNode* node = new RandomListNode(cur->label);
            tail->next = node;
            tail = tail->next;
            mymap[cur] = node;
            cur = cur->next;
        }
        tail = dummy->next;
        while(tail){             // connect all random pointers in this loop
            tail->random = mymap[head->random];// if head->next cannot find,
return NULL
            head = head->next;
            tail = tail->next;
        }
        head = dummy->next;
        return head;
    }
```

8.19 find intersect point of two singly list

- use map, T-(n) S-(n)
- better solution:
  - delta(list1, list2) = list1.size() - list2.size()
  - longer list go delta steps, then two list have same length
  - go forward together, and return the meet point

8.20 add two list -- youtube second round interview

- add two vector, same if add two linked list

```
vector<int> addition(vector<int>& vec1, vector<int>& vec2){
        vector<int> res;
        if(vec1.size() == 0) return vec2;
        if(vec2.size() == 0) return vec1;
        int cur=0;
        int carry = 0;
        int index1 = vec1.size()-1;
        int index2 = vec2.size()-1;
        while(index1 >= 0 && index2 >=0){
                cur = vec1[index1] + vec2[index2] + carry;
                carry = cur / 10;
                cur = cur % 10;
                res.insert(res.begin(),cur);
                index1--;
                index2--;
```

```
        }
        while(index1 >= 0){
                cur = vec1[index1] + carry;
                carry = cur / 10;
                cur = cur % 10;
                res.insert(res.begin(),cur);
                index1--;
        }
        while(index2 >= 0){
                cur = vec2[index2] + carry;
                carry = cur / 10;
                cur = cur % 10;
                res.insert(res.begin(),cur);
                index2--;
        }
        if(carry == 1) res.insert(res.begin(),1);
        return res;
}
/* add 1 to a vector*/
vector<int> addOne(vector<int>& vec){
        vector<int> res(vec.size());
        if(vec.size() == 0) return res;
        int cur = (vec[vec.size()-1] + 1) % 10;
        int carry = (vec[vec.size()-1] + 1) / 10;
        res[vec.size()-1] = cur;
        for(int i= (int)(vec.size())-2; i>=0; i--){
                cur = vec[i] + carry;
                cur = cur%10;
                carry = cur / 10;
                res[i] = cur;
        }
        if(carry == 1) res.insert(res.begin(),1);
        return res;
}
/* add two linked list, and list is from most significant to least*/
/*  1->2->3 + 9->3->8 == 3->6->1   (123+938 = 1061) */
use recursion, list->next until list==NULL, return head, then insert new head
Node* addLists(Node* list1, Node* list2){
        if(!list1) return list2;
        if(!list2) return list1;
        int list1_size = getSize(list1);
        int list2_size = getSize(list2);
        if(list1_size > list2_size){
                padList(list2,list1_size-list2_size);
        }else{
                padList(list1,list2_size-list1_size);
        }
        int carry = 0;     // reference, will check == 1 at last
        Node* head =  addListHelper(list1,list2, carry);
        if(carry == 1){     // check again if one last carry
                Node* node = new Node(1);
                node->next = head;
                head = node;
        }
```

```
        return head;
}
Node* addListHelper(Node* list1, Node* list2, int& carry){
        if(!list1 && !list2){
                return NULL;
        }
        Node* head = addListHelper(list1->next,list2->next,carry);
        int cur = list1->val + list2->val + carry;
        carry = cur / 10;
        cur = cur % 10;
        Node* node = new Node(cur);
        node->next = head;
        returnde;       }
}
```

8.21 find if a singly linked list is palindrom

There will be three solutions:

- reverse lis and compare two list is equal or not (no extra memory)
- push first half of list into stack, then pop and compare with second half (O(n) extra memory)
- use recursion, pass Node*& head each time

```
    bool recurse(Node*& head, Node* next ){
                recurse(head, next->next);
                head = head->next; // reference will change head here
    }
```

8.22 Insertion Sort List

Sort a linked list using insertion sort.

- just like insertion sort for array, insert cur node into previous list
- just traverse from head of the list, insert the cur node inside
- continue cur = cur->next

```
ListNode *insertionSortList(ListNode *head) {
    if(head == NULL)
    return NULL;
    ListNode* dummy = new ListNode(-1);
    ListNode* pre = dummy;
    ListNode* cur = head;
    while(cur != NULL)
    {
        ListNode* next = cur->next;  // save it
        pre = dummy;
        while(pre->next!=NULL && pre->next->val<=cur->val)
        {
            pre = pre->next;
```

```
            }
            cur->next = pre->next;
            pre->next = cur;
            cur = next; // start from the saved next
        }
        return dummy->next;
    }
```

8.23 Rotate List

Given a list, rotate the list to the right by k places, where k is non-negative.
For example:
Given 1->2->3->4->5->NULL and k = 2,
return 4->5->1->2->3->NULL.

- do not need 3 time reverse, because node can O(1) insertion
- just find the new head Node (k%n=index), then link old end to old head

```
    ListNode *rotateRight(ListNode *head, int k) {
        if(!head) return NULL;
        if(k==0) return head;  // k%0 will error
        int n = getLength(head);  // get length
        int p = k % n;           // real rotate number
        if(p == 0) return head;
        ListNode* cur = head;
        for(int i=0; i<n-p-1; i++){ // find new end
            cur = cur->next;
        }
        ListNode* temp = cur->next;
        ListNode* new_head = temp;  // new head
        cur->next = NULL;
        while(temp->next){
            temp = temp->next;
        }
        temp->next = head;   // link original end to original head
        return new_head;
    }
    int getLength(ListNode* head){
        int count = 0;
        while(head){
            count++;
            head = head->next;
        }
        return count;
    }
```

8.24 Reverse Nodes in k-Group

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.
If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.
You may not alter the values in the nodes, only nodes itself may be changed.
Only constant memory is allowed.
For example,
Given this linked list: 1->2->3->4->5
For k = 2, you should return: 2->1->4->3->5
For k = 3, you should return: 3->2->1->4->5

- we use reverse list template to reverse each sublist which has k nodes
- we need to return the first reversed list head as new head
- we need to store the previous reversed list's tail, and this tail will points to the new reversed list's head.
- we will count how many k goups we have, and then just traverse these groups
- at last link the left <k nodes

```
    ListNode *reverseKGroup(ListNode *head, int k) {
        if(!head) return NULL;
        int len = getLength(head);
        if(len < k || k==0 || k==1) return head;
        int group_num = len / k;   // count how many k goups
        ListNode* cur = head;
        ListNode* last_cur = head;
        ListNode* prev = NULL;
        ListNode* end = head;
        for(int i=0; i<group_num; i++){
            // start a new sublist
            cur = end;
            for(int i=0; i<k; i++){
                end = end->next;
            }
            prev = reverseList(cur,end);
            if(i==0) head = prev;    // store the new return head once
            // link two sublist
            last_cur->next = prev;
            last_cur = cur;
        }
        // link the last <k nodes
        cur->next = end;
        return head;
    }

    ListNode* reverseList(ListNode* head, ListNode* end){
        ListNode* prev = NULL;
        ListNode* next = NULL;
        while(head != end){
            next = head->next;   // store next
            head->next = prev;   // head points to prev

            prev = head;           // move one step forward
            head = next;
        }
```

```
        return prev;          // return prev
    }
```

9 heap, priority queue

- C++ use: std::priority_queue<ListNode*,vector<ListNode*>, Compare> pq; (minHeap)
- std::priority_queue<ListNode*,vector<ListNode*> > pq; (defautl maxHeap)
- check "merge k list " problem, pq.top(), pq.push(), pq.pop()

```
    struct Compare{
        bool operator()( ListNode* lhs, ListNode* rhs){
            return lhs->val > rhs->val;
        }
    };
```

9.1 heap insert, remove, max, min

- insert: T- O(logn): insert at the end of arr, then fixdown()
- remove: T-O(logn): replace the node with the last node, then fixdown
- max() for Maxheap or min() for Minheap -- T - O(1)

```
class Heap{
public:
    Heap(int *b, bool (*c)(int, int)) : A(b), comp(c){ // function pointer
        heapSize = -1;
    }
    virtual ~Heap()
    {
        if( A ){  delete[] A;  }
    }
    virtual bool Insert(int e) = 0;
    virtual int  GetTop() = 0;
    virtual int  ExtractTop() = 0;
    virtual int  GetCount() = 0;

protected:
    int left(int i){
        return 2 * i + 1;
    }
    int right(int i){
        return 2 * (i + 1);
    }
    int parent(int i){
        if( i <= 0 ){
            return -1;
        }
        return (i - 1)/2;
    }
```

```
    int   *A;
    bool  (*comp)(int, int);
    int   heapSize;
    int top(void){
        int max = -1;
        if( heapSize >= 0 ){
            max = A[0];
        }
        return max;
    }
    int count(){
        return heapSize + 1;
    }
    void heapify(int i)
    {
        int p = parent(i);
        if( p >= 0 && comp(A[i], A[p]) ){ // comp - functio pointer for min and
maxheap
            Exch(A[i], A[p]);
            heapify(p);
        }
    }
    int deleteTop(){
        int del = -1;
        if( heapSize > -1){
            del = A[0];
            Exch(A[0], A[heapSize]);
            heapSize--;
            heapify(parent(heapSize+1));
        }
        return del;
    }
    // Helper to insert key into Heap
    bool insertHelper(int key){
        bool ret = false;
        if( heapSize < MAX_HEAP_SIZE ){
            ret = true;
            heapSize++;
            A[heapSize] = key;
            heapify(heapSize);    // after insert at end, heapify from end
        }
        return ret;
    }
};
class MinHeap : public Heap{
public:
    MinHeap() : Heap(new int[MAX_HEAP_SIZE], &Smaller) { }
        //...
};
class MaxHeap : public Heap{
public:
    MaxHeap() : Heap(new int[MAX_HEAP_SIZE], &Greater)  {  }
        //...
```

```
};
// global function, used by fucntion pointer
bool Smaller(int a, int b)
{
    return a < b;
}
```

9.2 heap sort

- heap here is not the heap memory we say, it is a complete binary tree like data structure.
- based on complete binary tree structure (actually no tree build), create a heap by calling fix down from bottom to front based on parent >= two children (now this is max heap).
- then recursive to this: the topest is the largest, replace it to the last position, then this root is at the final position. then fixdown() again to parent >= children, replace again. Finally done.
- check "merge k list problem" which is a example of using heap(priority_queue)

```
int a[n];
Left_child = (2*i)+1;
right_child = (2*i) + 2;
parent = floor( (i-1) / 2);

T: worstcase O(nlgn), S: O(1). not stable
        void heapSort(vector<int>& vec){
                if (vec.size() == 0) return;
                buildHeap(vec);
                for (int i = vec.size() - 1; i >= 0; i--){
                        myswap(vec, 0, i);
                        fixDown(vec, 0, i-1);  // check i-1, i position is fixed
                }
        }
           // at most nlog(n)
        void buildHeap(vector<int>& vec){
                for (int i = vec.size() - 1; i >= 0; i--){    // can be size()/2,
non-leaf start from it
                        fixDown(vec, i, vec.size() - 1);
                }
        }
           // at most log(n)
        void fixDown(vector<int>&vec, int k, int size){
                while (k <= size){
                        int j = 2 * k + 1;      // left child 2*k+1, right child
2*k+2
                        if (2 * k + 2 <= size && vec[2 * k + 2] > vec[j]){
                                j = 2 * k + 2;
                        }
                        if (j <= size && vec[k] < vec[j]){
                                myswap(vec, k, j);
                        }
                        k = j;    // go down to the changed child node
                }
        }
```

```
        void myswap(vector<int>&vec, int lhs, int rhs){
                int temp = vec[lhs];
                vec[lhs] = vec[rhs];
                vec[rhs] = temp;
        }
```

9.3 Median Number

Numbers keep coming, return the median number
Making it clear, when the input size is odd, we take the middle element of sorted data. If the input size is even, we pick average of middle two elements in sorted stream

- http://www.geeksforgeeks.org/median-of-stream-of-integers-running-integers/
- at first use upper class to build heap (maxheap(left tree) and minheap(right tree))
- middle is m, insert e, if left heap and right heap size equal, just insert e and give e to m
- if left size larger than right size by 1, now left top equals middle, than shift left top to right heap and insert e to left. finally, give averge of left and right top to middle

```
int getMedian(int e, int &m, Heap &l, Heap &r){
    // Are heaps balanced? If yes, sig will be 0
    int sig = Signum(l.GetCount(), r.GetCount());
    switch(sig)
    {
    case 1: // There are more elements in left (max) heap
        if( e < m ) { // current element fits in left (max) heap
            r.Insert(l.ExtractTop());
            l.Insert(e);
        }else{
            // current element fits in right (min) heap
            r.Insert(e);
        }
        // Both heaps are balanced
        m = Average(l.GetTop(), r.GetTop());
        break;

    case 0: // The left and right heaps contain same number of elements
        if( e < m ) { // current element fits in left (max) heap
            l.Insert(e);
            m = l.GetTop();
        }else{
            r.Insert(e);
            m = r.GetTop();
        }
        break;

    case -1: // There are more elements in right (min) heap
                // same logic with case 1
        break;
    }
    return m;
```

```
}
void printMedian(int A[], int size){
    int m = 0; // effective median
    Heap *left  = new MaxHeap();
    Heap *right = new MinHeap();
    for(int i = 0; i < size; i++){
        m = getMedian(A[i], m, *left, *right);
        cout << m << endl;
    }
    delete left;
    delete right;
}
```

10 Dynamic Programming

Dynamic programming means temporary calculated results can be stored in memory so that it can be directly used for the calculation afterwards.

There are two ways of dp:

- recursion + hash map
- while loop + List

Difference between two ways:

- recursion is doing divide and conquer mostly, it is from largest to smallest
- while loop is from initial to largest
- when ask all permutations like questions -- recursion
- when ask largest, smallest, yes/no like quesitons -- while loop DP

Dynamic Programming solving approach:

- state : which state it is and how to represent state
- function: function that calculate current state from previous states
- initial: initial state/value (smallest state value)
- answer: last state/value (largest state value)

Type of problem that may use DP:

- one sequence DP (40%)
- two sequence DP (40%)
- Matrix DP (10%)
- Backpack (10%) -- k sum
- cannot sort problem may use DP, longest common sequence can sort, so no DP

10.1 fibinacci number generator

Solution #1:
normal recursion T- O(2^n)

- easy way to understand: T(n) = T(n-1)+T(n-2) < 2*T(n-1)* < 2*T(n-1) => so upper bound if O(2^n)

- actually 2 should be golden ratio from math O(1.618^n) [https://www.mathsisfun.com/numbers/golden-ratio.html]
- link for explatination [https://www.geeksforgeeks.org/time-complexity-recursive-fibonacci-program/]
- S - O(2^n)

```
public int fib(int n){
    if(n == 0) return 0;
    if(n == 1) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

Solution #2:
DP soution => recursion + map T- O(n)

- store already calculated value in case next time needed
- every value only need to be calculated once => T - O(n)
- S - O(n)

```
public int fib(int n){
    Dictionary<int, int> dict = new Dictionary<int, int>();
    return fib(n, dict);
}

protected int fib(int n, Dictionary<int, int> dict){
    if(dict.ContainsKey(n)) return dict[n];
    if(n == 0) return 0;
    if(n == 1) return 1;
    dict[n] = fib(n - 1, dict) + fib(n - 2, dict);
    return dict[n];
}
```

Solution #3:
DP => while loop + iterative + list == preferred DP T- O(n)

```
* state: list to store all states
* function: list[i] = list[i-1] + list[i-2]
* initial: list[0] = 0; list[1] = 1;
* answer: list[n]
* T - O(n), S - O(n)
* finally list contains all fin value from 0 to n
```

```
public int fib(int n){
    List list = new List<int>(new int[n+1]); // list[0] ... list[n]
    list[0] = 0;
```

```
        list[1] = 1;
        int i = 2;
        while(i <= n){
            list[i] = list[i-1] + list[i-2];
            i++;
        }
        return list[n];
    }
```

10.2 Unique Paths

A robot is located at the top-left corner of a m x 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? m and n is <= 100


```
* first row and row colum, each point only has 1 uniqu path to it, so used as
inital value
* starting from second row and second column
* state:  matrix[i][j] to store number of unique path to this grid
* function:  matrix[i][j]  = matrix[i-1][j] + matrix[i][j-1]
* initial:  matrix[0][0]=1; matrix[0][j]=1; matrix[i][0] = 1;
* answer: matrix[m-1][n-1]
* matric[1][2] means second row, third column
```

```
public class Solution {
    public int UniquePaths(int m, int n) {
        if(m <= 1 || n <=1) return 1;
        int[][] arr = new int[m][];
        for(int i=0; i<m; i++)
            arr[i] = Enumerable.Repeat(1, n).ToArray();
        for(int i=1; i<m; i++){
            for(int j=1; j<n; j++){
                arr[i][j] = arr[i-1][j] + arr[i][j-1];
            }
        }
        return arr[m-1][n-1];
```

```
    }
}
```

10.3 Unique Paths with obstacles.

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.

- same with no obstacle problem, just set obstacles as 0, means no path to this grid
- initial is a little tricky

```
int uniquePathsWithObstacles(vector<vector<int> > &obstacleGrid) {
   if(obstacleGrid.size() == 0 ) return 0;
   if(obstacleGrid[0][0] == 1) return 0;
   int m= obstacleGrid.size();
   int n= obstacleGrid[0].size();
   vector<vector<int> > matrix(m,vector<int>(n,1) );
   for(int i=1; i<m; i++){
       if(obstacleGrid[i][0] == 1){         // notice here, set all to 0 after one
obstacle
           matrix[i][0] = 0;
       }else{
           matrix[i][0] = matrix[i-1][0];
       }
   }
   for(int j=1; j<n; j++){
       if(obstacleGrid[0][j] == 1){
           matrix[0][j] = 0;
       }else{
           matrix[0][j] = matrix[0][j-1];
       }
   }
   for(int i=1; i<m; i++){
       for(int j=1; j<n; j++){
           if(obstacleGrid[i][j] == 1){
               matrix[i][j] = 0;
           }else{
               matrix[i][j] = matrix[i-1][j] + matrix[i][j-1];
           }
       }
   }
   return matrix[m-1][n-1];
}
```

10.4 Triangle

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.
For example, given the following triangle

[
[2],
[3,4],
[6,5,7],
[4,1,8,3]
]
The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).
Note:
Bonus point if you are able to do this using only O(n) extra space, where n is the total number of rows in the triangle.

- DP, from bottom to top
- initial vec is last row, and keep using this vec and act as each upper row
- when to the top, return vec[0]
- 1+2+3...+n, so T-O(n^2), S- O(n) -- vec
- if from top to bottom, need care upper index may out of subscript

```
    int minimumTotal(vector<vector<int> > &triangle) {
        int row = triangle.size();
        if(row == 0) return 0;

        vector<int> vec = triangle[row-1];    // store last row, and use it each
time
        for(int i=row-2;i>=0;i--){
            for(int j=0; j<triangle[i].size(); j++){
                vec[j] = triangle[i][j] + std::min(vec[j],vec[j+1]);  // store
values
            }
        }
        return vec[0];
    }
// this recursive will show the recursion concept for this problem, which is time
consuming
// can use map to optimize, but need to use pointer to store x and y
helper(triangle, 0,0);
int helper(vector<int>& triangle, int x, int y){
   if(x==n){
        return 0;
    }
    return std::min(helper(x+1,y), helper(x+1,y+1)) + triangle[i][j];
}
```

10.5 Minimum Path Sum

Given a m x 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.

- state: matrix[i][j]-- min path sum to this point

- function: matrix[i][j] = min(matrix[i-1][j],matrix[i][j-1]) + matrix[i][j];
- initial: matrix[0][0] no change, matrix[0][j] += matrix[0][j-1]; matrix[i][0] += matrix[i-1][0];
- answer: matrix[m-1][n-1]
- create a new matrix if you do not want affect the original matrix, and actually you can use the original matrix for this problem
  https://leetcode.com/problems/minimum-path-sum/description/

```
public class Solution {
    public int MinPathSum(int[,] grid) {
        int m = grid.GetUpperBound(0);
        int n = grid.GetUpperBound(1);
        int[,] a = new int[m+1,n+1];
        a[0,0]=grid[0,0];
        for (int i=1;i<=m;i++){ a[i,0]=a[i-1,0]+grid[i,0];}
        for (int i=1;i<=n;i++){ a[0,i]=a[0,i-1]+grid[0,i];}
        for(int i=1;i<=m;i++){
            for (int j=1;j<=n;j++){
                a[i,j]= Math.Min( a[i,j-1]+grid[i,j], a[i-1,j]+grid[i,j]);
            }
        }
        return a[m,n];
    }
}
```

10.6 Pascal's Triangle

Total Accepted: 19964 Total Submissions: 63348My Submissions
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]
]

- T- (n^2)
- lower level get from upper level

```
    vector<vector<int> > generate(int numRows){
            vector<vector<int> > res;
            if(numRows == 0) return res;
            vector<int> vec1(1,1);
            res.push_back(vec1);
            if(numRows == 1) return res;
            vec1.push_back(1);
```

```
            res.push_back(vec1);
            if(numRows == 2) return res;
            for(int i=1; i< numRows-1; i++){
                vector<int> vec(1,1);
                for(int j=0; j<res[i].size()-1; j++){
                        vec.push_back(res[i][j] + res[i][j+1]);
                    }
                    vec.push_back(1);
                    res.push_back(vec);
            }
            return res;
    }
```

10.7 Climbing Stairs

You are climbing a stair case. It takes n steps to reach to the top.
Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

- state: vec[i] is how many distinct ways to climb to i stair
- function: vec[i] = vec[i-1] + vec[i-2];
- initial: vec[0] = 1, vec[1]=2;
- answer: vec[n-1]

```
    int climbStairs(int n) {
        if(n==0) return 0;
        if(n==1) return 1;
        if(n==2) return 2;
        vector<int> vec(n,0);    // remeber to inital vec
        vec[0] = 1;
        vec[1] = 2;
        for(int i=2; i<n; i++){
            vec[i] = vec[i-1] + vec[i-2];
        }
        return vec[n-1];
    }
```

10.8 Jump Game

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.

- state vec[i] true or false, can I jump to position i
- function: vec[i] = arr[0]....arr[j], j< i any j can reach i
- initial vec[0]=true;

- answer vec[n-1]
- DP - can I reach i from 0 to i-1
- T- O(n^2) and T- O(n)

```
//check O(n) solution after this
bool canJump(int A[], int n){
        if(n<=0) return false;
        if(n==1) return true;
        vector<bool> arr(n,false);  // bool vector, reach or not
        arr[0] = true;
        for(int i=0;i<n;i++){
            for(int j=0;j<i;j++){
                if(arr[j] && j+A[j]>=i){ // check arr[j] can be reach
                    arr[i] = true;
                    break;     // break if reachable
                }
            }
        }
        return arr[n-1];
    }
// This is a O(n) solution
//O(n) solution, at each i, try maxReach, and store it, until i==n-1
bool canJump(int A[],int n){
        if(n<=0) return false;
        int maxReach = 0;
        for(int i=0; i<n; i++){
                if(i<= maxReach){
                        maxReach = std::max(maxReach,A[i]+i);
                        if(i == n-1){
                                return true;
                        }
                }
        }
        return false;
}
```

10.9 Jump Game II (minimum number of jumps to n)

- state vec[i] - min jumps needed to jump to position i
- function: vec[i] = arr[0]....arr[j] + 1 , j< i
- initial vec[0]=0, all others are INT_MAX, means unreachable;
- answer vec[n-1]

```
    int jump(int A[], int n) {
        if(n<=1) return 0;
        vector<int> arr(n,INT_MAX);
        arr[0] = 0; // we are siting at first index, no need to jump
        for(int i=0;i<n;i++){
            for(int j=0;j<i;j++){
```

```
                if(arr[j] < INT_MAX && j+A[j] >= i){
                    arr[i] = arr[j]+1;
                    break;      // break beacuse it is the min jumps needed now
                }
            }
        }
        return arr[n-1];
    }
```

10.10 Palindrome Number

Determine whether an integer is a palindrome. Do this without extra space.

- std::to_string(x) will works well, but use extra memory
- solution below is better, recursion mean no extra momery

```
    // better way, recursive, so smart solution, no extra memory
    bool isPalindrome(int x){
        if(x<0) return false;
        return check(x,x);
    }
    bool check(int x, int& y){    // x is copy, y is reference
        if(x == 0) return true;
        if(check(x/10,y)){
            if(x%10 == y%10){
                y /= 10;
                return true;
            }else{
                return false;
            }
        }
        return false;
    }
```

10.11 Longest Palindromic Substring

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.

- this is actually a matrix DP problem, build a table
- state: table[i][j] represents if string from i to j is paliindrome
- function: table[i][j] = table[i+1][j-1] && s[i]==s[j]
- initial: table[i][i]= true; table[i][i+1]=true;
- answer: return s.substr(maxStart, maxLen)

```
    string longestPalindrome(string s) {
        if(s.size()==0) return "";
        vector<vector<bool> > table(s.size(),vector<bool>(s.size(),false));
```

```
                int maxLen = 1;
                int maxStart = 0;
        buildLookUpTable(s, table, maxLen, maxStart);
                return s.substr(maxStart,maxLen);
    }
    void buildLookUpTable(string s, vector<vector<bool> >& table, int& maxLen,
int& maxStart){
        for(int i=0;i<s.size(); i++){
            table[i][i] = true;
        }
                for(int i=1; i<s.size(); i++){
                        table[i-1][i] = (s[i-1]==s[i]);
                        if(table[i-1][i] ){
                                maxLen = 2;             // remember maxlen and
start point
                                maxStart = i-1;
                        }
                }

                for(int len=2; len<s.size(); len++){
                        for(int st=0; st+len < s.size(); st++){
                                table[st][st+len] = (table[st+1][st+len-1] &&
(s[st]==s[st+len]) );
                                if(table[st][st+len]){
                                        maxLen = len+1;
                                        maxStart = st;  // remember maxlen and
start point
                                }
                        }
                }
    }
```

10.12 Palindrome Partitioning

Given a string s, partition s such that every substring of the partition is a palindrome.
Return all possible palindrome partitioning of s.
For example, given s = "aab",
Return
[
["aa","b"],
["a","a","b"]
]

- permutation template -- recursion, not DP, but see problem II below
- at index position, try index++ to end, push any palindrom substring into vec

```
    bool isPalindrom(string s, int pos, int index){
        while(pos < index){
            if(s[pos] != s[index]){
                return false;
```

```
            }else{
                pos++;
                index--;
            }
        }
        return true;
    }
vector<vector<string>> partition(string s) {
        vector<vector<string> > res;
        if(s.size() == 0) return res;
        vector<string> vec;
        helper(s,0,vec,res);
        return res;
    }
    void helper(string s, int index, vector<string>& vec, vector<vector<string> >&
res){
        if(index >= s.size()){
            res.push_back(vec);
            return;
        }
        for(int i=index; i<s.size(); i++){
            if(isPalindrom(s, index, i)){         // check if it is
                vec.push_back(s.substr(index,i-index+1));
                helper(s,i+1,vec,res);
                vec.pop_back();
            }
        }
}
```

10.13 Palindrome Partitioning II

Given a string s, partition s such that every substring of the partition is a palindrome.
Return the minimum cuts needed for a palindrome partitioning of s.
For example, given s = "aab",
Return 1 since the palindrome partitioning ["aa","b"] could be produced using 1 cut.

- loopup table isPalindrom use DP
- state: cut[i] stores min cut needed from 0 to i
- function: cut[i] = min(cut[j]... j<i)
- initial cut[0] = 0.
- answer: cut[s.length()] -1
- loopup table is half filled, upper right half filled

```
   int minCut(string s) {
        if (s.size() == 0) {
            return 0;
        }
        int* cut = new int[s.length() + 1];
        bool** isPalindrome = getIsPalindrome(s);
```

```
        cut[0] = 0;
        for (int i = 1; i <= s.length(); i++) {
            cut[i] = INT_MAX;
            for (int j = 1; j <= i; j++) {
                if (isPalindrome[i - j][i - 1] && cut[i - j] != INT_MAX) {
                    cut[i] = std::min(cut[i], cut[i - j] + 1);
                }
            }
        }
        return cut[s.length()] - 1;  // here return length() and minus 1
    }
    bool isPalindrome(string s, int start, int end) {
        for (int i = start, j = end; i < j; i++, j--) {
            if (s[i] != s[j]) {
                return false;
            }
        }
        return true;
    }

    bool** getIsPalindrome(string s) {     // create table, same with longest
substring problem
        bool** isPalindrome = new bool*[s.length()];
        for(int i=0;i<s.length();i++){
            isPalindrome[i] = new bool[s.length()];
        }

        for (int i = 0; i < s.length(); i++) {
            isPalindrome[i][i] = true;
        }
        for (int i = 0; i < s.length() - 1; i++) {
            isPalindrome[i][i + 1] = (s[i] == s[i+1]);
        }
        for (int length = 2; length < s.length(); length++) {
            for (int start = 0; start + length < s.length(); start++) {
                isPalindrome[start][start + length]
                    = isPalindrome[start + 1][start + length - 1] && s[start] ==
s[start + length];
            }
        }
        return isPalindrome;
    }
```

10.14 Valid Palindrome

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.

- just implementation problem
- how to check alphanumeric and lower or uppercase

```
    bool isPalindrome(string s) {
        if (s.size()==0)   {return true;}
        int st = 0;
        int ed = s.size()-1;
        while (st<ed){
            if (isAlNum(s[st])==false){st++; continue;}
            if (isAlNum(s[ed])==false){ed--; continue;}

            if (toLower(s[ed]) != toLower(s[st])){
                return false;
            }else{
                st++;
                ed--;
            }
        }
        return true;
    }

    bool isAlNum(char c){
        if((c >= '0' && c<='9') || (c >='a' &&c <='z')
            || (c>='A' && c<='Z') )
        {
            return true;

        }else{
            return false;
        }
    }
    char toLower(char c){
        if(c>='A' && c<='Z'){
            return c-'A'+'a';
        }else{
            return c;
        }
    }
```

10.15 Word Break

Given a string s and a dictionary of words dict, determine if s can be segmented into a space-separated sequence of one or more dictionary words.
For example, given
s = "leetcode",
dict = ["leet", "code"].
Return true because "leetcode" can be segmented as "leet code".

- state: segment[i] --- can position i be perfect cutted
- function: segment[i] = OR(segment[j]) where j<i, j from 0 to i, and j+1 to i is a word in dict
- initial: segment[0]=0
- answer: segment[s.length()]
- T- O(NL) L is the max length of string dict

```
    bool wordBreak(string s, unordered_set<string> &dict) {
        if (s.length() == 0) {
            return false;
        }
        int maxLength = getMaxLength(dict);
        bool* canSegment = new bool[s.length() + 1];

        canSegment[0] = true;
        for (int i = 1; i <= s.length(); i++) {
            canSegment[i] = false;
            for (int j = 1; j <= maxLength && j <= i; j++) {   // if longer than
longest word in dict
                if (!canSegment[i - j]) {                                    // go
next loop
                    continue;
                }
                string word = s.substr(i - j, j);
                if (dict.find(word)!=dict.end()) {
                    canSegment[i] = true;
                    break;              // just need know true, then go next
                }
            }
        }
        return canSegment[s.length()];
    }

    int getMaxLength(unordered_set<string>& dict) {  // good to have length to
shorten time
        int maxLength = 0;
        for (string word : dict) {
            maxLength = std::max(maxLength, (int)word.length());
        }
        return maxLength;
    }
```

10.16 Word Break II

Given a string s and a dictionary of words dict, add spaces in s to construct a sentence where each word is a valid dictionary word.
Return all such possible sentences.
For example, given
s = "catsanddog",
dict = ["cat", "cats", "and", "sand", "dog"].
A solution is ["cats and dog", "cat sand dog"].

- recursion, use permutation template
- below is a JAVA solution copy from 9chapter

```
    public ArrayList<String> wordBreak(String s, Set<String> dict) {
        // Note: The Solution object is instantiated only once and is reused by
each test case.
        Map<String, ArrayList<String>> map = new HashMap<String,
ArrayList<String>>();
        return wordBreakHelper(s,dict,map);
    }

    public ArrayList<String> wordBreakHelper(String s, Set<String> dict,
Map<String, ArrayList<String>> memo){
        if(memo.containsKey(s)) return memo.get(s);
        ArrayList<String> result = new ArrayList<String>();
        int n = s.length();
        if(n <= 0) return result;
        for(int len = 1; len <= n; ++len){
            String subfix = s.substring(0,len);
            if(dict.contains(subfix)){
                if(len == n){
                    result.add(subfix);
                }else{
                    String prefix = s.substring(len);
                    ArrayList<String> tmp = wordBreakHelper(prefix, dict, memo);
                    for(String item:tmp){
                        item = subfix + " " + item;
                        result.add(item);
                    }
                }
            }
        }
        memo.put(s, result);
        return result;
    }
```

10.17 Word Ladder

Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that:
Only one letter can be changed at a time
Each intermediate word must exist in the dictionary
For example,
Given:
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",
return its length 5.
Note:
Return 0 if there is no such transformation sequence.

All words have the same length.
All words contain only lowercase alphabetic characters.

- use graph level order BFS + queue to traverse the graph
- replace each letter in word with a-z, and check if it exists in dict. if exist, push into queue and remove it from dict to avoid next level to use it again
- return if replaced string equals to end string

```
int ladderLength(string start, string end, unordered_set<string> &dict) {
    if (dict.size() == 0) {
        return 0;
    }
    std::queue<string> Q;
    Q.push(start);
    dict.erase(start);
    int length = 1;
    while(!Q.empty()) {
        int count = Q.size();        // graph level BFS
        for (int i = 0; i<count; i++){
            string current = Q.front();
            Q.pop();
            for (int j=0; j < current.length(); j++) {
                for (char c = 'a'; c <= 'z'; c++) {
                    if (c == current[j]) {
                        continue;
                    }
                    string tmp = replace(current, j, c);
                    if (tmp == end) {
                        return length + 1;
                    }
                    if ( dict.find(tmp) != dict.end() ){
                        Q.push(tmp);
                        dict.erase(tmp);
                    }
                }
            }
        }
        length++;
    }
    return 0;
}
string replace(string cur, int j, char c){
    cur[j] = c;
    return cur;
}
```

10.18 Word Ladder II

Given two words (start and end), and a dictionary, find all shortest transformation sequence(s) from start to end, such that:

Only one letter can be changed at a time
Each intermediate word must exist in the dictionary
For example,
Given:
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
Return
["hit","hot","dot","dog","cog"],
["hit","hot","lot","log","cog"]

- same logic with upper I problem
- this time really build a graph from start to end, but use addPrev to build (level BFS)
- Now, graph only contains all shortest path, just use DFS to print all of them (start from end to start)

```
struct Node{
    string val;
    int no;   // level number
    Node(int no, string val):no(no),val(val){}
    void addPrev(Node* node){
        prev.push_back(node);
    }
    vector<Node*> prev;
};
class Solution {
    vector<vector<string> > answer;
public:
    vector<vector<string>> findLadders(string start, string end,
unordered_set<string> &dict) {
        std::map<string, Node*> map;
        queue<Node*> queue;
        Node* node = new Node(0, start);
        Node* endNode = NULL;
        map[start] =  node;
        queue.push(node);
        bool stop = false;
        while (queue.size() > 0 && !stop) {
            int count = queue.size();
            for (int i = 0; i < count; i++) {
                node = queue.front();
                queue.pop();
                for (int j = 0; j < node->val.length(); j++) {
                    string t = node->val;
                    for (char k = 'a'; k <= 'z'; k++) {
                        t[j] = k;
                        if (dict.find(t) != dict.end() ){
                            Node* v = map[t];
                            if (v == NULL) {
                                Node* temp = new Node(node->no + 1, t);
                                temp->addPrev(node);
                                queue.push(temp);
```

```
                                map[t] = temp;
                                if (t == end) {      // set end node and stop on
this level
                                    endNode = temp;
                                    stop = true;
                                }
                            }
                            else {
                                if (v->no == node->no + 1) {
                                    v->addPrev(node);
                                }
                            }
                        }
                    }
                }
            }
        }

        if (endNode != NULL) {
            vector<string> vec;
            vec.push_back(end);  // insert at front all others
            findPath(endNode, vec, start);
        }
        return answer;
    }
    void findPath(Node* node, vector<string>& cur, string start) {
        if (node->val == start) {
            answer.push_back(cur);
            return;
        }
        for (Node* n : node->prev) {
            cur.insert(cur.begin(),n->val);
            findPath(n, cur, start);
            cur.erase(cur.begin());
        }
    }
};
```

10.19 Word Search

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 =
[
["ABCE"],
["SFCS"],
["ADEE"]
]
word = "ABCCED", -> returns true,

word = "SEE", -> returns true,
word = "ABCB", -> returns false.

- try each i,j pair, if board[i][j]==word[0], then recurse compare the word
- can set original talbe visited as #, and restore it as unvisited
- exponential time complexity, depend on content of table, not easy to say

```
    // mark visited
    bool exist(vector<vector<char> > &board, string word) {
        if(board.size() == 0){
            return false;
        }
        if(word.length() == 0){
            return true;
        }

        for(int i = 0; i< board.size(); i++){
            for(int j=0; j< board[0].size(); j++){
                if(board[i][j] == word[0]){    // start from i,j pair, compare
with whole word
                    bool rst = find(board, i, j, word, 0);
                    if(rst) return true;
                }
            }
        }
        return false;
    }

    bool find(vector<vector<char> > &board, int i, int j, string word, int start){
        if(start == word.length()){ // start is index of word
            return true;
        }

        if (i < 0 || i>= board.size() || j < 0 || j >= board[0].size() || board[i]
[j] != word[start]){
            return false;
         }

        board[i][j] = '#'; // mark as visited
        bool rst = find(board, i-1, j, word, start+1) || find(board, i, j-1, word,
start+1)
                           || find(board, i+1, j, word, start+1) || find(board,
i, j+1, word, start+1) ;
        board[i][j] = word[start];   // retore original value, set unvisited
        return rst;
    }
```

10.20 Substring with Concatenation of All Words

You are given a string, S, and a list of words, L, that are all of the same length. Find all starting indices of substring(s) in S that is a concatenation of each word in L exactly once and without any intervening characters.

For example, given:
S: "barfoothefoobarman"
L: ["foo", "bar"]
You should return the indices: [0,9].
(order does not matter).

10.21 Decode Ways

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.

- state: nums[i] -- number of decoded ways from 0 to i-1
- function: nums[i] = nums[i-1] + (num[i-2] if num[i,i-1] is valid pair)
- initial: nums[0]=1; num[1]=1 or 0;
- answer: nums[s.length()]
- reason for nums[s.length()]: for 12, if i=2, then num[2] = num[1]+num[0] =1+1=2;
  num[0] is a helping value, no real meaning

```
int numDecodings(string s) {
    if (s.length() == 0) {
        return 0;
    }
    vector<int> nums(s.length()+1,0);
    nums[0] = 1;
    nums[1] = s[0] != '0' ? 1 : 0;          // check first number
    for (int i = 2; i <= s.length(); i++) {
        if (s[i - 1] != '0') {
            nums[i] = nums[i - 1];
        }
        int twoDigits = (s[i - 2] - '0') * 10 + s[i - 1] - '0';
        if (twoDigits >= 10 && twoDigits <= 26) {
            nums[i] += nums[i - 2];
        }
    }
    return nums[s.length()];
}
```

10.22 Distinct Subsequences

Given a string S and a string T, count the number of distinct subsequences of T in S.
A subsequence of a string is a new string which is formed from the original string by deleting some (can be

none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).
Here is an example:
S = "rabbbit", T = "rabbit"
Return 3.

- state: num[i][j] --- number of T in S. T is from 0 to j, S is from 0 to i
- function:
- initial: num[i][0]=1 i from 0 to S.length
- answer: nums[S.length][T.length]

```
int numDistinct(string S, string T) {
    if (S.size() == 0 || T.size() == 0) {
        return 0;
    }
    vector<vector<int> > nums(S.size()+1, vector<int>(T.size()+1,0));
    for (int i = 0; i < S.length(); i++) {
        nums[i][0] = 1;
    }
    for (int i = 1; i <= S.length(); i++) {
        for (int j = 1; j <= T.length(); j++) {
            nums[i][j] = nums[i - 1][j];
            if (S[i - 1] == T[j - 1]) {
                nums[i][j] += nums[i - 1][j - 1];
            }
        }
    }
    return nums[S.length()][T.length()];
}
```

10.23 Interleaving String

Given s1, s2, s3, find whether s3 is formed by the interleaving of s1 and s2.
For example,
Given:
s1 = "aabcc",
s2 = "dbbca",
When s3 = "aadbbcbcac", return true.
When s3 = "aadbbbaccc", return false.

- direct compare and use map does not work, we don't know which duplicate we choose (order need to be kept)
- this dp question is very good, imagine we have a grid, then horizental is s1, vertical is s2, starting from (0,0), go to (m-1, n-1) (last cell), see if it possible. return true or false;
- state: A[i][j] -- can s1 from 0 to i and s2 from 0 to j interleaving to be s3 from 0 to i+j
- funtion: A[i][j] = (s3[i+j]==s1[i] && match[i-1][j]) || (s3[i+j] ==s2[j] && match[i][j-1])
- initial: ...
- answer: A[s1.length()][s2.length()]

- good link with similar questions and solutions: https://leetcode.com/problems/distinct-subsequences/discuss/128519/One-C++-dp-solution-for-All-712-583-72-115-97

```
    bool isInterleave(string s1, string s2, string s3) {
        int n1 = s1.size();
        int n2 = s2.size();
        vector<vector<bool> > A(n1+1,vector<bool>(n2+1,false));
        if (n1+n2!=s3.size()){return false;}
        if (s1.empty()&&s2.empty()&&s3.empty()){return true;}

        A[0][0]=true;
        for (int i=1;i<=n1;i++){
            if (s1[i-1]==s3[i-1] && A[i-1][0]){A[i][0]=true;}
        }
        for (int i=1;i<=n2;i++){
            if (s2[i-1]==s3[i-1] && A[0][i-1]){A[0][i]=true;}
        }

        for (int i=1;i<=n1;i++){
            for (int j=1;j<=n2;j++){
                A[i][j]= (A[i][j-1] && (s2[j-1]==s3[i+j-1])) || (A[i-1][j]&&
(s1[i-1]==s3[i+j-1]));
            }
        }
        return A[n1][n2];
    }
```

10.24 Longest Substring Without Repeating Characters

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 "bbbbb" the longest substring is "b", with the length of 1.

- not DP
- use a int arr to store all visited elements, size is 256 because char num is from 0 to 255
- use cur_len to indicate current substring - so smart

```
    int lengthOfLongestSubstring(string s) {
        if(s.size()==0) return 0;
        int n = s.size();
        int cur_len = 1;  // To store the lenght of current substring
        int max_len = 1;  // To store the result
        int prev_index;  // To store the previous index
        int *visited = (int *)malloc(sizeof(int)*256);

         /* Initialize the visited array as -1, -1 is used to indicate that
            character has not been visited yet. */
        for (int i = 0; i < 256;  i++)
            visited[i] = -1;
```

```
          /* Mark first character as visited by storing the index of first
             character in visited array. */
          visited[s[0]] = 0;

         /* Start from the second character. First character is already processed
              (cur_len and max_len are initialized as 1, and visited[str[0]] is set
*/
         for (int i = 1; i < n; i++)
         {
             prev_index =  visited[s[i]];
             /* If the currentt character is not present in the already processed
                 substring or it is not part of the current NRCS, then do cur_len++
*/
             if (prev_index == -1 || i - cur_len > prev_index)
                 cur_len++;
             /* If the current character is present in currently considered NRCS,
                 then update NRCS to start from the next character of previous
instance. */
             else{
                     /* Also, when we are changing the NRCS, we should also check
whether
                     length of the previous NRCS was greater than max_len or not.*/
                 if (cur_len > max_len)
                     max_len = cur_len;
                 cur_len = i - prev_index;    // start from the next on of
duplicated previous one
             }
             visited[s[i]] = i; // update the index of current character
         }
         // Compare the length of last NRCS with max_len and update max_len if
needed
         if (cur_len > max_len)
             max_len = cur_len;
         free(visited); // free memory allocated for visited
         return max_len;
     }
```

10.25 Longest Increasing Subsequence

- differ from LConcecutiveS (unordered), This is LIS
- state list[i] - include pos i, how many increasing number from 0 to i
- lis[i] = 1+ max(lis[j]) where j<i, arr[i] > arr[j]
  = 1 arr[i] <= arr[i]
- initial list[i] = 1;
- return max(list[i])

```
/* lis() returns the length of the longest increasing subsequence in
    arr[] of size n */
int lis( int arr[], int n )
{
   int *lis, i, j, max = 0;
```

```
    lis = (int*) malloc ( sizeof( int ) * n );

    /* Initialize LIS values for all indexes */
    for ( i = 0; i < n; i++ )
       lis[i] = 1;

    /* Compute optimized LIS values in bottom up manner */
    for ( i = 1; i < n; i++ )
       for ( j = 0; j < i; j++ )
          if ( arr[i] > arr[j] && lis[i] < lis[j] + 1)
             lis[i] = lis[j] + 1;

    /* Pick maximum of all LIS values */
    for ( i = 0; i < n; i++ )
       if ( max < lis[i] )
          max = lis[i];

    /* Free memory to avoid memory leak */
    free( lis );

    return max;
}
```

10.26 Longest Common Subsequence

//naive solution: O(2^n) possible subsequences of X, so T- O(2^n * m)
// differ from Longest common substring
// O(n*m) dp solution
state: f[i][j]表示前i个字符配上前j个字符的LCS的长度
function: f[i][j] = f[i-1][j-1] + 1 // a[i] == b[j]
= MAX(f[i-1][j], f[i][j-1]) // a[i] != b[j]
intialize: f[i][0] = 0
f[0][j] = 0
answer: f[a.length()][b.length()]

```
int lcs( char *X, char *Y, int m, int n )
{
   vector<vector<int> > L(m+1,vector<int>(n+1,0));
   int i, j;
   /* Following steps build L[m+1][n+1] in bottom up fashion. Note
      that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] */
   for (i=0; i<=m; i++)
   {
     for (j=0; j<=n; j++)
     {
       if (i == 0 || j == 0)
         L[i][j] = 0;

       else if (X[i-1] == Y[j-1])
         L[i][j] = L[i-1][j-1] + 1;
```

```
        else
          L[i][j] = max(L[i-1][j], L[i][j-1]);
      }
    }

    /* L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1] */
    return L[m][n];
}
```

10.27 Longest Common Substring

// naive solution: C(n,2) possible substring of X, then each compare with Y - T- O(m^2 * n)
// two while loop, DP, T(n*m)*
state: f[i][j]表示前i个字符配上前j个字符的LCS'的长度
(一定以第i个和第j个结尾的LCS')
function: f[i][j] = f[i-1][j-1] + 1 // a[i] == b[j]
= 0 // a[i] != b[j]
intialize: f[i][0] = 0
f[0][j] = 0
answer: MAX(f[0..a.length()][0..b.length()])
strlen(X) will return char string length

```
int LCSubStr(char *X, char *Y,  int m,  int n)
{
    vector<vector<int> > LCSuff(m+1,vector<int>(n+1,0));
    int result = 0;  // To store length of the longest common substring
    for (int i=0; i<=m; i++)
    {
        for (int j=0; j<=n; j++)
        {
            if (i == 0 || j == 0)       // both initilize outer or like this inner
are ok
                LCSuff[i][j] = 0;

            else if (X[i-1] == Y[j-1])
            {
                LCSuff[i][j] = LCSuff[i-1][j-1] + 1;
                result = std::max(result, LCSuff[i][j]);
            }
            else LCSuff[i][j] = 0;
        }
    }
    return result;
}
```

10.28 Edit Distance

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

- matrix DP problem
- state: f[i][j] operations needed to make 0 to i and 0-j to be same
- function: f[i][j]= min(f[i-1][j-1], f[i-1][j]+1, f[i][j-1]) if s[i] = s[j]
  = min(f[i-1][j-1], f[i-1][j], f[i][j-1]) +1 if s[i]!=s[j]
- initial: f[i][0]=i; f[0][j] = j
- answer: f[s.length][s.length]

```
    int minDistance(string word1, string word2) {
        int n = word1.length();
        int m = word2.length();

        vector<vector<int> > dp(n+1,vector<int>(m+1,0));
        for(int i=0; i< m+1; i++){
            dp[0][i] = i;
        }
        for(int i=0; i<n+1; i++){
            dp[i][0] = i;
        }

        for(int i = 1; i<n+1; i++){
            for(int j=1; j<m+1; j++){
                if(word1[i-1] == word2[j-1]){
                    dp[i][j] = dp[i-1][j-1];      // no need to check i-1,j and
i,j-1
                }else{
                    dp[i][j] = 1 + std::min(dp[i-1][j-1],std::min(dp[i-1][j],dp[i]
[j-1]));
                }
            }
        }
        return dp[n][m];    // one pos right, easy to solve problem
    }
```

10.29 Backpack or ksum problem

see backpack/ Ksum problem part, also use DP

10.30 minimum adjust cost

n个数，可以对每个数字进行调整，使得相邻的两个数的差都<=target, 调整的费用为
Sigma(|A[i]-B[i]|)
A[i]原来的序列 B[i]是调整后的序列

A[i] < 200, target < 200
让代价最小

state: f[i][v] 前i个数，第i个数调整为v，满足相邻两数<=target，所需要的最小代价
function: f[i][v] = min(f[i-1][v'] + |A[i]-v|, |v-v'| <= target)
answer: f[n][...]

O(n * A * T)

10.31 Best Time to Buy and Sell Stock

Say you have an array for which the ith element is the price of a given stock on day i.
If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

- actually simple dp problem, do not need array, not maxpofit and minindex
- find max then find min befor max

```
    int maxProfit(vector<int> &prices) {
        if(prices.size()<2) return 0;
        int min_index = 0;
        int profit = 0;
        for(int i=1;i<prices.size();i++){
            int temp_profit = prices[i] - prices[min_index];
            if(temp_profit > profit){
                profit = temp_profit;
            }
            if(prices[min_index] > prices[i]){    // find smallest inside (0 - i)
                min_index = i;
            }
        }
        return profit>0? profit : 0;
    }
```

10.32 Best Time to Buy and Sell Stock II

Say you have an array for which the ith 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). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

- buy at lowest and sell at highest, check increasing subvec

```
    int maxProfit(vector<int> &prices) {
        if(prices.size() < 2) return 0;
        int profit = 0;
        int first = 0;
        int next = 1;
        while(next < prices.size()){
```

```
            if(prices[next] > prices[next-1]){
                next++;
            }else{
                profit+= prices[next-1] - prices[first];
                first = next;
                next++;
            }
        }
        if(next == prices.size()){
            profit+= prices[next-1] - prices[first];
        }
        return profit;
    }
```

10.33 Best Time to Buy and Sell Stock III

Say you have an array for which the ith element is the price of a given stock on day i.
Design an algorithm to find the maximum profit. You may complete at most two transactions.
Note:
You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

- using Dynamic Programming
- left[i]: maxprofit from left to right, right[i] from right to left
- if allowed k transaction. f[i][j] - ith day and j tranction max profit
  = max(0 to i-1 day, j-1 transaction) + maxprofit.

```
    int maxProfit(vector<int> &prices) {
        if(prices.size() < 2) return 0;
        vector<int> left(prices.size(),0);
        vector<int> right(prices.size(),0);
        //dp from left
        int max_profit = 0;
        int min = prices[0];
        for(int i=1;i<prices.size();i++){
            if(prices[i] - min > max_profit){
                max_profit = prices[i]- min;
            }
            if(prices[i] < min){
                min = prices[i];
            }
            left[i] = max_profit;    // left contain left profit of each day
        }
        //dp from right
        max_profit = 0;
        int max = prices[prices.size()-1];
        for(int i= prices.size()-2; i>=0;i--){
            if(max - prices[i] > max_profit){
                max_profit = max - prices[i];
            }
```

```
            if(prices[i] > max) {
                max = prices[i];
            }
            right[i] = max_profit;
        }
        // merge left and right
        max_profit = 0;   // if result is negative, also return 0
        for(int i=0; i< left.size();i++){
            max_profit = std::max(left[i]+right[i], max_profit);
        }
        return max_profit;
    }
```

10.34 Maximum Subarray

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.
click to show more practice.
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.

- sliding window: just add each one, if sum<0, disgard sum, and start from i
- if res>=0, then add A[i], store max. if res<0, res = A[i], store max

```
    int maxSubArray(int A[], int n){
        int max = INT_MIN;
        int res = 0;
        for(int i=0;i<n;i++){
            if(res >= 0){
                res += A[i];
                max = std::max(res, max);
            }else{
                res = A[i];
                max = std::max(res,max);
            }
        }
        return max;
    }
```

- prefix sum: preprocess the array: store sum[i]
- then sum(i to j) = sum[j]-sum[i], find the largest sum[i to j]
- for this problem, just find largetst as w are e preprocessing

```
    int maxSubArray(int A[], int n){
        if (n <= 0){
            return 0;
```

```
        }
        int max = INT_MIN, sum = 0, minSum = 0;
        for (int i = 0; i < n; i++) {
            sum += A[i];
            max = std::max(max, sum - minSum);
            minSum = std::min(minSum, sum);
        }
        return max;
    }
```

For finding minimum subarray:

- find largest as max subarray problem
- make all number in array as negative, then find the maximum
- just plus two max value, and return as result

10.35 Maximum Subarray II

Given an array of integers, find two non-overlapping subarrays which have the largest sum.
The number in each subarray should be contiguous.
Return the largest sum.
Note
The subarray should contain at least one number
Example
For given [1, 3, -1, 2, -1, 2], the two subarrays are [1, 3] and [2, -1, -2] or [1, 3, -1, 2] and [2], they both have the largest sum 7.

- preprocess the array, store sum[i] -- sum from 0 to i
- store matrix[i][j] -- subarray sum from i to j
- find the subarray you need
  better solution:
- like maxprofit allow two tansactions, dp from left to right, and right to left
- then combine them and return the result we need
  // will finish later

10.36 Maximum Product Subarray

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.

- using prefix and product[i] will be more complex
- as follow: we remeber positive max and negative min from any index to i
- then if A[i]<0, we just need to swap min and max, then new max and min will not change
- start from 0 for dealing with A[i]==0 situation

```
    int maxProduct(int A[], int n) {
        if(n==1) return A[0];
        int pMax=0, nMax=0, m = 0;
```

```
        for(int i=0; i<n; i++){
            if(A[i]<0) swap(pMax, nMax);
            pMax = max(pMax*A[i], A[i]);  // if A[i]==0, max ==min==0
            nMax = min(nMax*A[i], A[i]);
            m = max(m, pMax);
        }
        return m;
    }
```

10.37 Maximum Subarray Difference

Given an array with integers.
Find two non-overlapping subarrays A and B, which |SUM(A) - SUM(B)| is the largest.
Return the largest difference.
Note
The subarray should contain at least one number
Example
For [1, 2, -3, 1], return 6
Challenge Expand
O(n) time and O(n) space.

- combine max subarray I and II.
- for each i position, we have left side max sum and min sum, also right side max and min sum.
- then for each i position, store left_max-right_min or right_max-left_min
- return the final result
  // will add code later

10.38 Subarray IV

1. Find the subarray which sum equals to zero. O(n)
2. Find the subarray which sum is closest to zero. O(nlogn)
   For 1:

- use prefix sum
- we need two sum equal, so just push all sum into map, and try to find the same sum
- all can be done in one loop amd T-O(n), S-O(n)

For 2:

- same with first problem
- calculate all sum[i], then sort them which will be O(log(n))
- traverse to find most closest two sum by compare sum[i] and sum[i+1]
  // will add code later

10.39 Scramble String

Given a string s1, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.
Below is one possible representation of s1 = "great":

great
/ \
gr eat
/ \ / \
g r e at
/ \
a t
To scramble the string, we may choose any non-leaf node and swap its two children.
For example, if we choose the node "gr" and swap its two children, it produces a scrambled string "rgeat".
rgeat
/ \
rg eat
/ \ / \
r g e at
/ \
a t
We say that "rgeat" is a scrambled string of "great".
Similarly, if we continue to swap the children of nodes "eat" and "at", it produces a scrambled string "rgtae".
rgtae
/ \
rg tae
/ \ / \
r g ta e
/ \
t a
We say that "rgtae" is a scrambled string of "great".
Given two strings s1 and s2 of the same length, determine if s2 is a scrambled string of s1.

- partion at a point, then compare two part foward and reverse order
- T-O(3^n) -- sort will trim lots of recursions

```
    bool isScramble(string s1, string s2) {
        if(s1.size() != s2.size()) return false;
        if(s1 == s2) return true;
        string ts1 = s1,ts2 = s2;
        sort(ts1.begin(), ts1.end());   // trim unnecessary recursions
        sort(ts2.begin(), ts2.end());
        if(ts1 != ts2) return false;
        for(int isep = 1; isep < s1.size(); ++ isep) {
            string seg11 = s1.substr(0,isep);
            string seg12 = s1.substr(isep);
            {//see if forward order is ok
                string seg21 = s2.substr(0,isep);
                string seg22 = s2.substr(isep);
                if(isScramble(seg11,seg21) && isScramble(seg12,seg22)) return
true;
            }
            {//see if reverse order is ok
```

```
                string seg21 = s2.substr(s2.size() - isep);
                string seg22 = s2.substr(0,s2.size() - isep);
                if(isScramble(seg11,seg21) && isScramble(seg12,seg22)) return
true;
            }
        }
        return false;
    }
```

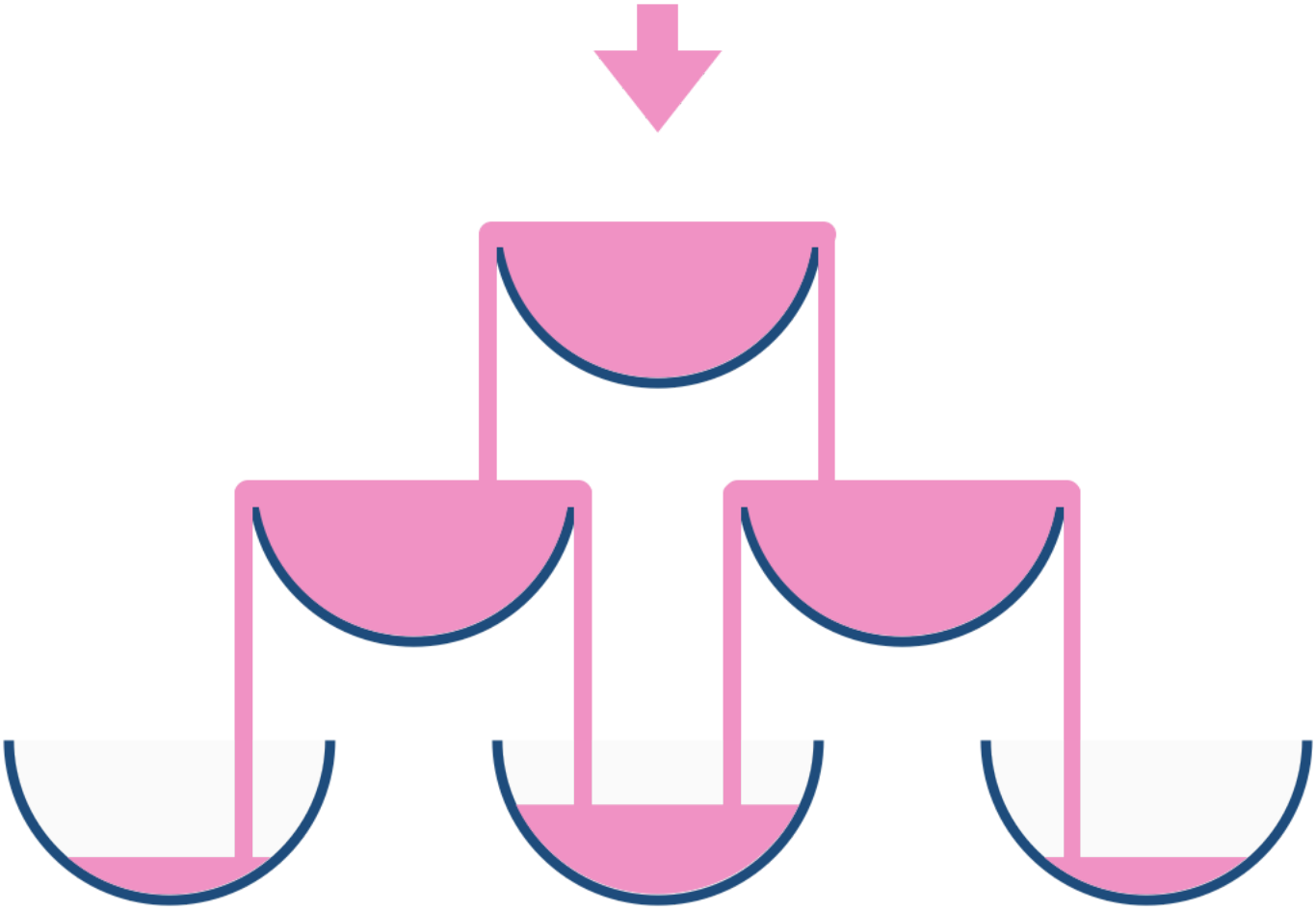
- DP version, 3D dp
- comes up with DP naturally
  f[n][i][j] means isScramble(s1[i: i+n], s2[j: j+n])
  f[n][i][j] = f[k][i][j] && f[n - k][i+k][j+k]
  || f[k][i][j+n-k] && f[n-k][i+k][j]
- initial value f[0][i][j] = true if s1[i]==s2[j];
- return f[n-1][0][0]
- T- O(n^4)

```
    bool isScramble(string s1, string s2) {
        if( s1.length() != s2.length() ){
            return false;
        }
        if( s1==s2 ) {
            return true;
        }
        int n = s1.length();
        vector<vector<vector<bool> > > rst(n,vector<vector<bool> >(n,vector<bool>
(n,false)));
        for(int i=0; i< n; i++){
            for(int j=0;j<n; j++){
                rst[0][i][j] = s1[i] == s2[j];
            }
        }

        for(int len = 2; len <= n; len++){
            for(int i = n - len; i>= 0; i--) {
                for(int j = n - len;  j>=0; j--){
                    bool r = false;
                    for(int k = 1; k < len && r == false; k++){
                        r = (rst[k-1][i][j] && rst[len-k-1][i+k][j+k]) || (rst[k-
1][i][j+len-k] && rst[len-k-1][i+k][j]);
                    }
                    rst[len-1][i][j] = r;
                }
            }
        }

        return rst[n-1][0][0];
    }
```

10.40 Champagne Tower

We stack glasses in a pyramid, where the first row has 1 glass, the second row has 2 glasses, and so on until the 100th row. Each glass holds one cup (250ml) of champagne.

Then, some champagne is poured in the first glass at the top. When the top most glass is full, any excess liquid poured will fall equally to the glass immediately to the left and right of it. When those glasses become full, any excess champagne will fall equally to the left and right of those glasses, and so on. (A glass at the bottom row has it's excess champagne fall on the floor.)



For example, after one cup of champagne is poured, the top most glass is full. After two cups of champagne are poured, the two glasses on the second row are half full. After three cups of champagne are poured, those two cups become full - there are 3 full glasses total now. After four cups of champagne are poured, the third row has the middle glass half full, and the two outside glasses are a quarter full, as pictured below.

Now after pouring some non-negative integer cups of champagne, return how full the j-th glass in the i-th row is (both i and j are 0 indexed.)

Example 1: Input: poured = 1, query_glass = 1, query_row = 1 Output: 0.0 Explanation: We poured 1 cup of champange to the top glass of the tower (which is indexed as (0, 0)). There will be no excess liquid so all the glasses under the top glass will remain empty.

Example 2: Input: poured = 2, query_glass = 1, query_row = 1 Output: 0.5 Explanation: We poured 2 cups of champange to the top glass of the tower (which is indexed as (0, 0)). There is one cup of excess liquid. The glass indexed as (1, 0) and the glass indexed as (1, 1) will share the excess liquid equally, and each will get half cup of champange.

Note:

poured will be in the range of [0, 10 ^ 9]. query_glass and query_row will be in the range of [0, 99].

```
* not a binary tree as each row just has one element, looks like a tree,
  but actually just a grid.
* state: arr[i][j]
* initial: arr[0][0]
* function: arr[i+1][j] = (arr[i][j] - 1) / 2
            arr[i+1][j+1] += (arr[i][j] - 1) / 2
* reset arr[i][j] = 1 after calculation
```

```
public class Solution {
    public double ChampagneTower(int poured, int query_row, int query_glass) {
        double[][] arr = new double[101][];
        for(int i=0;i<arr.Length; i++) arr[i] = new double[101];
        arr[0][0] = poured;
        for(int i=0; i<=query_row; i++){
            for(int j=0; j<=i; j++){
                if(arr[i][j] > 1){
                    arr[i+1][j] += (arr[i][j] - 1) / 2;
                    arr[i+1][j+1] += (arr[i][j] - 1) / 2;
                    arr[i][j] = 1;
                }
            }
        }
        return arr[query_row][query_glass];
    }
}
```

11 Dictionary, HashSet, HashTable

In C# we have Dictionary, HashSet

- Dictinary and HashSet are implemented by hash table, hash function & linked list. insert, delete, search are O(1)
- Dicionary cannot have duplicate key
- HashSet cannot have duplicate element

Dictionary basic:

```
Dictionary<int, string> dict = new Dictionary<int, int>();
dict.Add(1, "test1");
dict[2] = "test2";
if(dict.CotainsKey(2)) return true;
dict.Remove(2);
if(dict.Keys.Count() > 5) return false;
foreach(int k in dict.Keys){
```

```
    Console.WriteLine(k);
}
```

HashSet basics:

```
HashSet hs = new HashSet<int>();
hs.Add(3);
if(!hs.Contains(4)) hs.Add(4);
hs.Remove(3);
Console.WriteLine(hs.Count);
```

11.1 create a hash table structure

- using List as a bucket, and create table_size buckets to store all key value paires
- for each node of list, it will be a <key,value> pair where key is original key which can be used when searching an element
- how to solve collision?
    - using linked list
    - insert to next table position.
    - rehashing: create a new hashtable and put all old hash table key,value pair in new hash table

11.2 Longest Consecutive Sequence (LCS)

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.
For example,
Given [100, 4, 200, 1, 3, 2],
The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.
Your algorithm should run in O(n) complexity.

- using unordered_map for O(1) access time
  2 two traverse, first store all elements, second time pick one element and check its left and right side -- T- O(n)
- another nlog(n) approach: sort it and linear search

```
    int longestConsecutive(vector<int> &num) {
        if(num.size() == 0) return 0;
        std::unordered_map<int,bool> mymap;
        for(int i=0;i<num.size();i++){
            mymap[num[i]] = false;
        }
        int max = 0;
        int res = 1;
        std::unordered_map<int,bool>::iterator iter;
        for(iter=mymap.begin();iter!=mymap.end();iter++){
            if(iter->second) continue;
            int left = iter->first - 1;
            int right = iter->first + 1;
```

```
            while(mymap.find(left)!=mymap.end()){
                res++;
                mymap[left] = true;
                left--;
            }
            while(mymap.find(right)!=mymap.end()){
                res++;
                mymap[right] = true;
                right++;
            }
            if(res > max) max = res;
            res = 1;
        }
        return max;
    }
```

11.3 LRU Cache

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.

- use doubly linked list, when search one element, we need prev node if we want to delete the current node
- use dummy head and tail node to make it simpler
- use map<int, Node> act as map<key, value> to O(1) find each node
- get method: 1 find node 2 remove current node 3 move current node to the end
- set method: 1 use get method 2 if not existed, check capacity, if full, remove the first node.
  3 add new node to the end
- remember update map<key, value> pair when delete or set a node

```
class LRUCache{
private:
    struct Node{
        Node* prev;
        Node* next;
        int key;
        int value;
        Node(int key, int value) {
            this->key = key;
            this->value = value;
            this->prev = NULL;
            this->next = NULL;
        }
    };
    int capacity;
    unordered_map<int, Node*> hs;
    Node* head;
```

```
    Node* tail;
public:
    LRUCache(int capacity) {
        head = new Node(-1, -1);
        tail = new Node(-1, -1);
        tail->prev = head;
        head->next = tail;
        this->capacity = capacity;
    }

    int get(int key) {
        if( hs.find(key) == hs.end()) {
            return -1;
        }
        // remove current
        Node* current = hs[key];
        current->prev->next = current->next;
        current->next->prev = current->prev;

        // move current to tail
        move_to_tail(current);

        return hs[key]->value;
    }

    void set(int key, int value) {
        if( get(key) != -1 ) {    // find the node, remove it, and move to tail
            hs[key]->value = value;
            return;
        }

        if (hs.size() == capacity) {
            hs.erase(head->next->key);
            head->next = head->next->next;
            head->next->prev = head;
        }

        Node* insert = new Node(key, value);
        hs.insert(std::pair<int,Node*>(key, insert));
        move_to_tail(insert);
    }
    void move_to_tail(Node* current) {
        current->prev = tail->prev;
        tail->prev = current;
        current->prev->next = current;
        current->next = tail;
    }
};
```

11.4 good hash function - magic number 33

```
int hashfunc(String key) {
    int sum = 0;
    for (int i = 0; i < key.length(); i++) {
        sum = sum * 33 + (int)(key[i]);
        sum = sum % HASH_TABLE_SIZE;
    }
    return sum
}
```

12 String

12.1 Implement strStr() -- leetcode

Returns a pointer to the first occurrence of needle in haystack, or null if needle is not part of haystack.

- just normal search T(n*m), but deep analysis, T(n*km) k is time first char is the same, and we know m will be smaller than m each time compare substring.
- KMP algorithm O(n+m), more complex
- tree match is another similar problem, use same approach

```
    char *strStr(char *haystack, char *needle) {
        if(!haystack || !needle) return NULL;
        string hs = haystack;
        string nd = needle;
        if(nd.size()==0) return haystack;
        if(hs.size()==0) return NULL;
        if(nd.size() > hs.size()) return NULL;
        int i, j;
        for(i = 0; i < hs.length() - nd.length() + 1; i++) {
            for(j = 0; j < nd.length(); j++) {
                if(hs[i + j] != nd[j]) {
                    break;
                }
            }
            if(j == nd.length()) {
                return haystack+i;
            }
        }
        return NULL;
    }
```

12.2 ZigZag Conversion

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)
P A H N
A P L S I I G

Y I R
And then read line by line: "PAHNAPLSIIGYIR

- create a table, each row is a string
- we just need to return final string, so no need to keep empty chars, just insert new chars
- use zig=false to tell in '|' direction

```
    string convert(string s, int nRows) {
         vector<string> zz(nRows,"");
         if (nRows==1){return s;}
         int i=0; //for string
         int r=0; //for zz vector
         bool zig = false; // zig=false, in '|' direction, true in '/' direction
         while (i<s.size()){
             if (r<nRows && !zig){
                 zz[r]+=s[i];
                 r++;
                 i++;
             }else{
                 if (!zig){
                     zig=true;
                     r--;
                 }else{
                     r--;
                     zz[r] = zz[r]+s[i];
                     if (r==0){zig=false;r++;} // from '/' to '|' direction
                     i++;
                 }
             }
         }
         string res; //get result;
         for (int i=0;i<zz.size();i++){
             for (int j=0;j<zz[i].size();j++){
                     res = res+ zz[i][j];
             }
         }
         return res;
     }
```

12.3 Validate if a given string is numeric.

Some examples:
"0" => true
" 0.1 " => true
"abc" => false
"1 a" => false
"2e10" => true
Note: It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one.
1: 2e10 == *210^10. 2e-12* == 210^-12

2: there bool, num,dot,exp(e) indicates there is number, . , e before
3: spaces at beginning and end, +, - at beginning or end, +,- right after e are ok!

```
    bool isNumber(const char *s) {
        std::string str = s;
        int len = str.length();
        int i = 0, e = len - 1;
        // skip space
        while (i <= e && s[i]==' ') i++;
        if (i > len - 1) return false;
        while (e >= i && s[e]==' ') e--;
        // skip leading +/-
        if (s[i] == '+' || s[i] == '-') i++;
        bool num = false; // is a digit
        bool dot = false; // is a '.'
        bool exp = false; // is a 'e'
        while (i <= e) {
            char c = s[i];
            if (c>='0' && c<='9') {
                num = true;
            }
            else if (c == '.') {  // if there is . , there is no e or any other .
                if(exp || dot) return false;
                dot = true;
            }
            else if (c == 'e') { // if there is e, no other e, and there are num
before e
                if(exp || num == false) return false;
                exp = true;
                num = false;
            }
            else if (c == '+' || c == '-') {  //+ or - will valid if after e
                if (s[i - 1] != 'e') return false;
            }
            else {
                return false;
            }
            i++;
        }
        return num;
    }
```

// we can use regular expression actually

- 12,456,778,332,123,456.456 ----- ^[1-9]\d{0,2}(,\d{3})*(.\d+)?$
- ^....$ start and end of line
- [1-9]\d{0,2} --- first group 12, start from 1 to 9, and can have 0 to 2 more digits
- (,\d{3})* ----- group of full 456 or 778. start by comma, and there are 3 digits
- (.\d+)? ----- if there is dot, then there are at least one digit after dot. ? means this group is optional

12.4 Substring with Concatenation of All Words

You are given a string, S, and a list of words, L, that are all of the same length. Find all starting indices of substring(s) in S that is a concatenation of each word in L exactly once and without any intervening characters.
For example, given:
S: "barfoothefoobarman"
L: ["foo", "bar"]
You should return the indices: [0,9].
(order does not matter).

- no dp, no dfs, just use map
- first map store all strings in L, second map store all visited strings whern traversing for making sure that each string occurs only once.
- each string has same length is very important, so we can pick S.substr(i+j*len,len) as each string to compare.

```
vector<int> findSubstring(string S, vector<string> &L) {
    vector<int> res;
    int num = L.size();
    int len = L[0].size();
    if (num==0){return res;}
    map<string,int> mp;
    for (int i=0;i<num;i++){mp[L[i]]++;}

    int i=0;
    while ((i+num*len-1)<S.size()){
        map<string,int> mp2;
        int j=0;
        while (j<num){
            string subs = S.substr(i+j*len,len);
            if (mp.find(subs)==mp.end()){
                    break;
            }else{
                mp2[subs]++;
                if (mp2[subs]>mp[subs]){
                    break;
                }
                j++;  // j is number of strings
            }
        }
        if (j==num){res.push_back(i);}
        i++;
    }

    return res;
}
```

12.5 Simplify Path Total

Given an absolute path for a file (Unix-style), simplify it.
For example,

path = "/home/", => "/home"
path = "/a/./b/../../c/", => "/c"
/../ return / and duplicate //// need to be single /

- just while loop to traverse the path, ignore all slashes /
- just pick one element by find start and end of the element using slash
- push all elements inside stack
- if element is ".." pop top of stack, if is ".", do nothing
- finally, pop all emlemts in stack and connect them and return

```
        string simplifyPath(string path) {
             vector<string> stack;
             assert(path[0]=='/');   // must start with /
             int i=0;
             while(i< path.size())
             {
                  while(path[i] =='/' && i< path.size()) i++; //skip the begining
'////'
                  if(i == path.size())
                       break;
                  int start = i;
                  while(path[i]!='/' && i< path.size()) i++; //decide the end
boundary
                  int end = i-1;
                  string element = path.substr(start, end-start+1); // one element
                  if(element == "..")
                  {
                       if(stack.size() >0)
                       stack.pop_back();
                  }
                  else if(element!="."){   // if element is . , do nothing
                       stack.push_back(element);
                  }
             }
             if(stack.size() ==0) return "/";   // if /../ or /../../../
             string simpPath;
             for(int i =0; i<stack.size(); i++)
             simpPath += "/" + stack[i];   // pop all element and connect them
             return simpPath;
        }
```

12.6 Wildcard Matching

Implement wildcard pattern matching with support for '?' and '*'.
'?' *Matches any single character.*
* Matches any sequence of characters (means abc can match to one *, and including the empty sequence).
The matching should cover the entire input string (not partial).

- star remeber the position of last *
- ss remember the position of last char with star

- if s is empty, p must be all ********
- if one * has been matched, we don't need to consider it again
- use *s or s==NULL to check end of s

```
    bool isMatch(const char *s, const char *p) {
        const char* star=NULL;
        const char* ss=s;
        while (*s){
            if ((*p=='?')||(*p==*s)){s++;p++;continue;}
            if (*p=='*'){star=p++; ss=s;continue;}
            if (star){ p = star+1; s=++ss;continue;}
            return false;
        }
        while (*p=='*'){p++;}
        return !*p;
    }
```

12.7 Multiply Strings

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.

- result string's length will be num1.size() + num2.size(), try multiply and test
- for last element, res[i+j+1] =(num[i]*num[j]) %10, and also for others

```
    string multiply(string num1, string num2) {
        if (num1.size() == 0 || num2.size() == 0) {
            return "";
        }

        int len1 = num1.length(), len2 = num2.length();
        int len3 = len1 + len2;
        int i, j, product, carry;

        vector<int> num3(len3,0);
        for (i = len1 - 1; i >= 0; i--) {
            carry = 0;
            for (j = len2 - 1; j >= 0; j--) {
                product = carry + num3[i + j + 1] + ( (int)(num1[i]-'0') * (int)
(num2[j]-'0') );
                num3[i + j + 1] = product % 10;
                carry = product / 10;
            }
            num3[i + j + 1] = carry;
        }

        string str;
        i = 0;
        while (i < len3 - 1 && num3[i] == 0) {  // if final result is 0, allow it
            i++;
```

```
        }
        while (i < len3) {
            str += std::to_string(num3[i++]);
        }
        return str;
    }
```

```
// this will work for not very long result
long long res =  atoi(num1.c_str())*atoi(num2.c_str());
return std::to_string(res);
```

12.8 Anagrams

Given an array of strings, return all groups of strings that are anagrams.
Note: All inputs will be in lower-case.

- very good problem. -- create signiture
- O(n^2) solution will be pick each string and compare with all others
- O(n) solution will be creating signiture of each string and at the same time find all anagrams by comparing the signiture.

```
    vector<string> anagrams(vector<string> &strs) {
        vector<string> res;
        if(strs.size() == 0) return res;
        if(strs.size() == 1) return res;
        std::map<string,vector<string> > mp; // each signiture contain a list of
strings
        for(int i=0; i<strs.size(); i++){
            string sig = signiture(strs[i]);
            if(mp.find(sig) != mp.end()){
                mp[sig].push_back(strs[i]);
            }else{
                vector<string> str;
                str.push_back(strs[i]);
                mp[sig] = str;
            }
        }
        std::map<string,vector<string> >::iterator iter;
        for(iter=mp.begin(); iter!=mp.end(); iter++){
            vector<string> vec = iter->second;
            if(vec.size() >1){   // if there are groups
                for(int i=0; i<vec.size(); i++){
                    res.push_back(vec[i]);
                }
            }
        }
        return res;
    }
    string signiture(string& str){
```

```
        if(str.size()==0) return "";
        vector<int> vec(26,0);  // totally 26 lower case chars (a - z)
        for(int i=0; i<str.size(); i++){
            int index = (int)(str[i]-'a');
            vec[index]++;
        }
        string res = "";
        for(int i=0; i<vec.size();i++){
            if(vec[i] > 0){
                char c = (char)(vec[i]+'a');
                res =  res + c + std::to_string(vec[i]);
            }
        }
        return res;
    }
```

12.9 Text Justification

Given an array of words and a length L, format the text such that each line has exactly L characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly L characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

For example,
words: ["This", "is", "an", "example", "of", "text", "justification."]
L: 16.

Return the formatted lines as:
[
"This is an",
"example of text",
"justification. "
]

- implementation problem, analyze the problem and conquer it
- steps:
    - find the start and end word for each line --- pack()
    - after finding one line, start format this line as required --- convert()
        - if it is a one word line, left justified, adding extra space at end. no word can exceed L
        - if it it the last line, left justified, wich may contain arbitrary number of words
        - if normal line, evenly distributed spaces into those intervals (end-start)

```

```

```
    vector<string> fullJustify(vector<string> &words, int L) {
                vector<string> ret;
                if (words.size() == 0)
                        return ret;

                int start = 0, end = pack(words, start, L);
                ret.push_back(convert(words, start, end, L));
                while (end != words.size() - 1) {
                        start = end + 1;
                        end = pack(words, start, L);
                        ret.push_back(convert(words, start, end, L));
                }
                return ret;
        }

        // pack a line and return the end of this line
        int pack(vector<string> &words, int start, int L) {
                int next = start; // the index of the next word
                int length = words[next].length();

                // always try to include the next word plus a padding space
(greedy packing)
                while (next + 1 < words.size() && length + 1 + words[next +
1].length() <= L)
                        length += words[++next].length() + 1;
                return next;
        }

        // convert multiple words along with extra padding space into a string of
length L
        string convert(vector<string> &words, int start, int end, int L) {
                string sb;
                // if this line only contains one word
                if (start == end) {
                        sb += words[start];
                        for (int i = 0; i < L - words[start].length(); i++) {
                                sb += " ";
                        }
                        return sb;
                }
                // if the line is the last line, the space distribution rule is
different
                else if (end == words.size() - 1) {
                        int curLen = 0;
                        for (int i = start; i < end; i++) {
                                sb += words[i];
                                sb += " ";
                                curLen += words[i].length() + 1;
                        }
                        sb.append(words[end]);
                        curLen += words[end].length();

                        for (int i = 0; i < L - curLen; i++) {
```

```
                        sb += " ";
                }
                return sb;
        }

        // calculate the lengths of padding space
        int totalLen = 0, numOfSpaces = end - start;
        for (int i = start; i <= end; i++)
                totalLen += words[i].length();
        int lenOfPaddingSpace = (L - totalLen) / numOfSpaces;
        int numOfExtraSpaces = (L - totalLen) % numOfSpaces;

        // construct the line
        int count = 0; // count of the extra spaces
        for (int i = start; i < end; i++) {
                sb += words[i];
                for (int j = 0; j < lenOfPaddingSpace; j++)
                        sb += " ";
                if (count < numOfExtraSpaces)
                        sb += " ";
                count++;
        }
        sb += words[end];

        return sb;
    }
```

12.10 Custom Sort String

S and T are strings composed of lowercase letters. In S, no letter occurs more than once.

S was sorted in some custom order previously. We want to permute the characters of T so that they match the order that S was sorted. More specifically, if x occurs before y in S, then x should occur before y in the returned string.

Return any permutation of T (as a string) that satisfies this property.

Example : Input: S = "cba" T = "abcd" Output: "cbad" Explanation: "a", "b", "c" appear in S, so the order of "a", "b", "c" should be "c", "b", and "a". Since "d" does not appear in S, it can be at any position in T. "dcba", "cdba", "cbda" are also valid outputs.

Note:

S has length at most 26, and no character is repeated in S. T has length at most 200. S and T consist of lowercase letters only. [https://leetcode.com/problems/custom-sort-string/description/]

- just need to sort T based on the order of S
- only need return one possible T
- So just pick all char from T based on S, then return a new T match with S
- if need to return all possible T, then will be like permutation mixed with subsets

```
public class Solution {
    public string CustomSortString(string S, string T) {
        if(S.Length > T.Length) return String.Empty;
        Dictionary<char, List<char>> list = new Dictionary<char, List<char>>();
        List<char> others = new List<char>();
        foreach(char c in T){
            if(S.IndexOf(c) != -1){
                if(!list.ContainsKey(c)){
                    list[c] = new List<char>();
                }
                list[c].Add(c);
            }
            else{
                others.Add(c);
            }
        }
        String res = String.Empty;
        if(list.Keys.Count() == S.Length){
            foreach(char c in S){
                res += new String(list[c].ToArray());
            }
            res += new String(others.ToArray());
        }

        return res;
    }
}
```

13 Permutations, Recursion

- permutation means how many different array can be grouped from the current array. (order matters. But same value may need to be ignored.)
- permutation number is n!. (e.g. [1,2,3] has totally 3! => 6 permutaions)
- if duplidtes elements exists, then it will be less than n! permutations (n factorial)
- if pick m elements from n elemements (m <= n), we have A_n^m => n!/(n-m)! permutations including order
- if ignore order, then we have C_n^m => A_n^m/m!
- basically if consider order, then use A (permutation), else use C(combination, subsets)

13.1 Permutations

Given a collection of numbers, return all possible permutations.

```
For example,
[1,2,3] have the following permutations:
[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], and [3,2,1].
```

[https://leetcode.com/problems/permutations/description/]

- create a new list, try to populate with each possible value for each element via backtracking and DFS recursion.
- no duplicates now, next one contains duplicates
- S - O(n*n!): n! lists and each list size == n
- T - O(n*n!): totally n! lists, and each node may need to be tested n time for visited check.
- another way is by inserting each value to all current lists, finally return all lists (take too long)
- if count needed, then it is n!

```
public class Solution
{
    public IList<IList<int>> Permute(int[] nums)
    {
        IList<IList<int>> res = new List<IList<int>>();
        if (nums == null) return res;
        List<int> cur = new List<int>();
        Dictionary<int, bool> visited = new Dictionary<int, bool>();
        GetPermute(nums, res, cur, visited);
        return res;
    }
    protected void GetPermute(int[] nums, IList<IList<int>> res, List<int> cur,
Dictionary<int, bool> visited)
    {
        if (cur.Count() == nums.Length)
        {
            res.Add(new List<int>(cur));
            return;
        }
        for (int i = 0; i < nums.Length; i++)
        {
            if (visited.ContainsKey(i) && visited[i]) continue;
            cur.Add(nums[i]);
            visited[i] = true;
            GetPermute(nums, res, cur, visited);
            cur.RemoveAt(cur.Count - 1);
            visited[i] = false;
        }
    }
}
```

- Calculate n!

```
protected int facorial(int n)
{
    if (n == 1) return 1;
    return n * facorial(n - 1);
}
```

13.2 Permutations II

Given a collection of numbers that might contain duplicates, return all possible unique permutations.
For example,
[1,1,2] have the following unique permutations:
[1,1,2], [1,2,1], and [2,1,1].

- it contains duplicates
- same as before, just ignore the second value if first same value is not used
- if only need count, then it is the same as need to avoid duplicate list

```
public class Solution
{
    public IList<IList<int>> PermuteUnique(int[] nums)
    {
        IList<IList<int>> res = new List<IList<int>>();
        if (nums == null) return res;
        List<int> cur = new List<int>();
        Dictionary<int, bool> visited = new Dictionary<int, bool>();
        Array.Sort(nums);
        GetPermute(nums, res, cur, visited);
        return res;
    }
    protected void GetPermute(int[] nums, IList<IList<int>> res, List<int> cur,
Dictionary<int, bool> visited)
    {
        if (cur.Count() == nums.Length)
        {
            res.Add(new List<int>(cur));
            return;
        }
        for (int i = 0; i < nums.Length; i++)
        {
            if (visited.ContainsKey(i) && visited[i]) continue;
            // ignore second value if first same value is not used (otherwise will
be duplicates)
            if (i > 0 && nums[i] == nums[i - 1] && !visited[i - 1]) continue;
            cur.Add(nums[i]);
            visited[i] = true;
            GetPermute(nums, res, cur, visited);
            cur.RemoveAt(cur.Count() - 1);
            visited[i] = false;
        }
    }
}
```

13.3 Subsets

Given a set of distinct integers, S, 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 S = [1,2,3], a solution is:

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

- there is no dupicate integer
- as need to be acending, so sort it firstly
- very similar with permutations, just need to start from last next element in each for loop as only need acending order. (permutation need all sets with all orders)
- i=pos, and helper(i+1), res.push_back(vec) each recursion
- totally there are 2^n subsets (math calculate) (no duplicates)
- C_n-0 + C_n-1 .... + C_n-n => 2^n
- T-O(2^n), S-O(2^n * n)

```
public class Solution {
    public IList<IList<int>> Subsets(int[] nums) {
        IList<IList<int>> res = new List<IList<int>>();
        IList<int> cur = new List<int>();
        Array.Sort(nums); // sort for acending order
        GetSubset(res, nums, cur, 0);
        return res;
    }

    protected virtual void GetSubset(IList<IList<int>> res, int[] nums, IList<int>
cur, int pos){
        res.Add(new List<int>(cur));  // always create a copy
        for(int i=pos; i<nums.Length; i++){
            cur.Add(nums[i]);
            GetSubset(res, nums, cur, i+1);
            cur.RemoveAt(cur.Count() - 1);
        }
    }
}
```

```
// sort for list, in place, Linq OrderBy also works, but will be a new list
li.Sort((a, b) => a.CompareTo(b)); // ascending sort
li.Sort((a, b) => -1* a.CompareTo(b)); // descending sort
// sort for array
```

```
Array.Sort(nums);
array = array.OrderByDescending(c => c).ToArray();
```

13.4 Subsets II

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

Note: 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],
  []
]
```

- i=pos, and helper(i+1), res.push_back(vec) each recursion
- go next, if equal, jump to for loop

```
public class Solution {
    public IList<IList<int>> SubsetsWithDup(int[] nums) {
        IList<IList<int>> res = new List<IList<int>>();
        IList<int> cur = new List<int>();
        Array.Sort(nums);
        GetSubset(res, nums, cur, 0);
        return res;
    }
    protected void GetSubset(IList<IList<int>> res, int[] nums, IList<int> cur,
int pos){
        res.Add(new List<int>(cur));
        for(int i=pos; i<nums.Length; i++){
            if(i > pos && nums[i] == nums[i - 1]) continue;
            cur.Add(nums[i]);
            GetSubset(res, nums, cur, i + 1);
            cur.RemoveAt(cur.Count() - 1);
        }
    }
}
```

13.5 Next Permutation

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

1,2,3,7,6,5,4 -> 1,2,4,3,5,6,7

- because permutation template will generate permutations in order, so we may sort this array and then generate all permutations until face current array, return next one. this is time comsuming.
- use math idea, just remember 4 steps
    - Step 1: Find the largest index k, such that A[k]<A[k+1]. If not exist, this is the last permutation. (in this problem just sort the array and return.)
    - Step 2: Find the largest index l, such that A[l]>A[k].
    - Step 3: Swap A[k] and A[l].
    - Step 4: Reverse A[k+1] to the end.

```
public class Solution {
    public void NextPermutation(int[] nums) {
        if(nums.Length < 2) return;
        // step 1
        int k = -1;
        for(int i=0; i<nums.Length - 1; i++){
            if(nums[i] < nums[i+1]){
                k = i;
            }
        }
        if(k == -1){
            Array.Sort(nums);
            return;
        }
        // step 2
        int l = -1;
        for(int i=0; i<nums.Length; i++){
            if(nums[i] > nums[k]){
                l = i;
            }
        }
        // step 3
        int temp = nums[k];
        nums[k] = nums[l];
        nums[l] = temp;
        // step 4
        Array.Reverse(nums, k+1, nums.Length - k  -1);
    }
}
```

13.6 Permutation Sequence

The set [1,2,3,...,n] contains a total of n! unique permutations.
By listing and labeling all of the permutations in order,
We get the following sequence (ie, for n = 3):
"123"
"132"
"213"
"231"
"312"
"321"
Given n and k, return the kth permutation sequence.
Note: Given n will be between 1 and 9 inclusive.

- just use permutation template to generate permutations in order and return kth one
- there is a math way, which is not easy to understand

```
        void helper(vector<int>& num, string& str, vector<int>& visited, int& k){
             if(str.size() == num.size()){
                 k--;
                     return ;
                 }
             for(int i=0;i<num.size();i++){
                    if(visited[i] == 1) continue;
                    str.push_back(num[i]+'0');
                    visited[i] = 1;
                    helper(num,str,visited,k);
                                if(k==0) return;
                    str.pop_back();
                    visited[i] = 0;
                }
        }
        long long factorial(int n){
            long long res = 1;
            for(int i=1; i<=n; i++){
                res = res * i;
            }
            return res;
        }
```

13.7 Path Sum

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,
5
/ \
4 8

/ / \
11 13 4
/ \ \
7 2 1
return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

- from root to leaf, so if it is leaf, check the sum = leaf->value
- see path sum II below, which is also a permutation problem
- different from finding all path from root to any node

```
    bool hasPathSum(TreeNode *root, int sum) {
       if(!root) return false;
       return hasPath(root,sum);
    }
    bool hasPath(TreeNode* root, int sum){
         if(root && !root->left && !root->right && sum == root->val) return true;
         else if(!root) return false;
        return hasPath(root->left,sum-root->val) || hasPath(root->right,sum-root-
>val);
    }
```

13.8 Path Sum II

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

- from root to leaf, so if it is leaf, check the sum = leaf->value
- different from finding all path from root to any node

```
    vector<vector<int> > allsol;
    vector<int> sol;
   vector<vector<int> > pathSum(TreeNode *root, int sum) {
        sol.clear();
        allsol.clear();
        if (root==NULL) {return allsol;}
        findSol(root, sum);
        return allsol;
    }
    void findSol(TreeNode* root, int sum){
        if ( (root->left==NULL) && (root->right==NULL) && (sum-root->val==0) ) {
            sol.push_back(root->val);
            allsol.push_back(sol);
            return;
        }
        sol.push_back(root->val);
        if (root->left !=NULL){
            findSol(root->left, sum-root->val);
            sol.pop_back();                          // very smart here
        }
        if (root->right!=NULL){
```

```
            findSol(root->right, sum-root->val);
            sol.pop_back();
            }
        }
```

13.9 Letter Combinations of a Phone Number

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"].

- create map<char, vector > to store reflection
- each number's chars are different, no duplicates

```
        vector<string> letterCombinations(string digits) {
                vector<string> res;
                if (digits.size() == 0){
                    res.push_back("");
                        return res;
                }
                std::map<char, vector<char> > mymap;
                vector<char> vec0{};
                vector<char> vec1{};
                vector<char> vec2{ 'a', 'b', 'c' };
                       //….
                vector<char> vec9{ 'w', 'x', 'y', 'z' };
                mymap['0'] = vec0;
                       //….
                mymap['9'] = vec9;
                string str = "";
                helper(digits, 0, mymap, str, res);
                return res;
        }
        void helper(string& digits, int level, map<char, vector<char> >& mymap,
string& str, vector<string>& res){
                if (level == digits.size()){
                        res.push_back(str);
                        return;
                }
                vector<char> vec = mymap[digits[level]];
                for (int i = 0; i< vec.size(); i++){
                        str = str + vec[i];
                        helper(digits, level + 1, mymap, str, res);
                        str.erase(str.end() - 1);
                }
        }
```

13.10 Combination Sum

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 (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).
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]

- permutation template

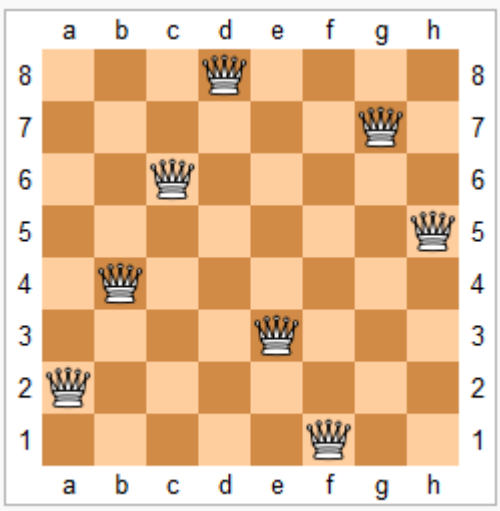
```
    vector<vector<int> > combinationSum(vector<int> &candidates, int target) {
        vector<vector<int> > result;
        if (candidates.size() == 0) {
            return result;
        }

        vector<int> path;
        std::sort(candidates.begin(),candidates.end());
        helper(candidates, target, path, 0, result);
        return result;
    }

    void helper(vector<int>& candidates, int target, vector<int>& path, int index,
vector<vector<int> >& result) {
        if (target == 0) {
            result.push_back(path);
            return;
        }
        int prev = -1;
        for (int i = index; i < candidates.size(); i++) {
            if (candidates[i] > target) {
                return;
            }

            if (prev != -1 && prev == candidates[i]) {  // all elements are
positive
                continue;
            }

            path.push_back(candidates[i]);
            helper(candidates, target - candidates[i], path, i, result);   //
change i to i+1 for II
            path.pop_back();

            prev = candidates[i];
        }
    }
```

13.11 Combination Sum II

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 (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).
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]
just change previous one line
helper(candidates, target - candidates[i], path, i, result);
to helper(candidates, target - candidates[i], path, i+1, result); // change i to i+1 for II

13.12 N-Queens

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other.
Given an integer n, return all distinct solutions to the n-queens puzzle.
Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space respectively.



One solution to the eight queens puzzle

```
class Solution {
private:
    typedef vector<vector<string> > Vecs;
    Vecs vecs;
public:
```

```
    // no same row, coloumn, diagnal
    vector<vector<string> > solveNQueens(int n) {
        if(n==0) return vecs;
        vector<int> vec(n,-1);
        total(0,n,vec);
        return vecs;
    }
    void convertToString(vector<int>& vec,int n){
        vector<string> s_vec;
        for(int i=0; i<vec.size(); i++){
            string str(n,'.');
            str[vec[i]] = 'Q';
            s_vec.push_back(str);
        }
        vecs.push_back(s_vec);
    }
    void total(int row, int n,vector<int>& vec){
        if(row == n){
            convertToString(vec,n);
            return;
        }
        for(int col=0;col<n;col++){
            if(isOK(vec, row, col)){
                vec[row] = col;
                total(row+1,n,vec);
            }
        }
    }
    bool isOK(vector<int>& vec, int row, int col){
        for(int i=0;i< row; i++){
            if(vec[i] == col){
                return false;
            }
            if(std::abs(vec[i]-col) == std::abs(i - row)){
                return false;
            }
        }
        return true;
    }
};
```

13.13 N-Queens II

Follow up for N-Queens problem. Now, instead outputting board configurations, return the total number of distinct solutions.

- just same with II version

```
class Solution {
public:
    int totalNQueens(int n) {
```

```
        if(n==0) return 0;
        vector<int> vec(n,-1);
        int count = 0;
        total(0,n,count,vec);
        return count;
    }
    void total(int row, int n, int& count,vector<int>& vec){
        if(row == n){
            count++;
            return;
        }

        for(int col=0;col<n;col++){
            if(isOK(vec, row, col)){
                vec[row] = col;
                total(row+1,n,count,vec);
            }
        }
    }
    bool isOK(vector<int>& vec, int row, int col){
        for(int i=0;i< row; i++){
            if(vec[i] == col){
                return false;
            }
            if(std::abs(vec[i]-col) == std::abs(i - row)){
                return false;
            }
        }
        return true;
    }
};
```

13.14 Regular Expression Matching

Implement regular expression matching with support for '.' and '*'.
'.' *Matches any single character.*
'*' Matches zero or more of the preceding element.
The matching should cover the entire input string (not partial).
The function prototype should be:
bool isMatch(const char *s, const char *p)*
Some examples:
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "a") → true
isMatch("aa", ".") → true
isMatch("ab", ".") → *true*
*isMatch("aab", "*ca*b") →* true

- represents 0 or more previous element

- represents any element, so if pattern elem is . , compare return true.
- if face " a* " in pattern, just recurse from 0 and will stop when cur elem not equal
- the key step is recurse and at the same time match check after a* pair

```
    bool isMatch(const char *s, const char *p) {
        std::string ss = s;
        std::string pp = p;
        return matchHelper(ss,pp,0,0);
    }
    bool matchHelper(string& s, string& p, int i, int j){
        if (s.length() == i) {
            return checkEmpty(p,j);  // p need to be a*b*c*, whhic means * apear
i+2
        }
        if (p.length() == j) {
            return false;
        }
        char c1 = s[i];
        char d1 = p[j], d2 = '\0'; // if j is last index of p, avoid d2 be *
        if (p.length() > j+1){
            d2 = p[j+1];         // set next index of p to d2
        }

        if (d2 == '*') {
            if (compare(c1, d1)) {
                return matchHelper(s, p,i+1,j) || matchHelper(s, p, i, j+2); //
key step
            }
            else {
                return matchHelper(s, p,i,j+2);
            }
        }
        else {
            if (compare(c1, d1)) {
                return matchHelper(s, p,i+1,j+1);
            }
            else {
                return false;
            }
        }
    }
    bool compare(char c1, char d1){
        return d1 == '.' || c1 == d1;
    }

    bool checkEmpty(string& p, int j) {
        if ((p.length()-j)%2 != 0) {
            return false;
        }
        for (int i = j+1; i < p.length(); i+=2) {
            if (p[i] != '*') {
                return false;
```

```
            }
        }
        return true;
    }
```

13.15 Restore IP Addresses

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)

- permutation template

```
    vector<string> restoreIpAddresses(string s) {
        vector<string> result;
        vector<string> list;

        if(s.length() <4 || s.length() > 12) // 0.0.0.0 is smallest ip
            return result;
        helper(result, list, s , 0); // list hold 4 part of ip
        return result;
    }
    void helper(vector<string>& result, vector<string>& list, string s, int start)
{
        if(list.size() == 4){
            if(start != s.length())
                return;
            string str = list[0];
            for(int i=1; i<list.size();i++){
                str = str + "." + list[i];
            }
            result.push_back(str);
            return;
        }

        for(int i=start; i<s.length() && i< start+3; i++){
            string tmp = s.substr(start, i-start+1);
            if(isvalid(tmp)){
                list.push_back(tmp);
                helper(result, list, s, i+1);
                list.pop_back();
            }
        }
    }

    bool isvalid(string& s){
        if(s[0] == '0')
            return s == "0";    // to eliminate cases like "00", "10"
        int digit = atoi(s.c_str());
```

```
            return digit >= 0 && digit <= 255;
        }
```

13.16 All Paths From Source to Target

Given a directed, acyclic graph of N nodes. Find all possible paths from node 0 to node N-1, and return them in any order.

The graph is given as follows: the nodes are 0, 1, ..., graph.length - 1. graph[i] is a list of all nodes j for which the edge (i, j) exists.

```
Example:
Input: [[1,2], [3], [3], []]
Output: [[0,1,3],[0,2,3]]
Explanation: The graph looks like this:
0--->1
|    |
v    v
2--->3
There are two paths: 0 -> 1 -> 3 and 0 -> 2 -> 3.
```

Note:

The number of nodes in the graph will be in the range [2, 15]. You can print different paths in any order, but you should keep the order of nodes inside one path.

[https://leetcode.com/problems/all-paths-from-source-to-target/description/]

- similar as permutation, using DFS
- time and space complexity need to be reviewed. ?
- Time Complexity: O(2^N* N^2). We can have exponentially many paths, and for each such path, our prepending operation path.add(0, node) will be O(N^2).
- Space Complexity: O(2^N * N), the size of the output dominating the final space complexity.

```
public class Solution {
    public IList<IList<int>> AllPathsSourceTarget(int[][] graph) {
        IList<IList<int>> res = new List<IList<int>>();
        Dictionary<int, List<int>> dict = new Dictionary<int, List<int>>();
        for(int i=0; i<graph.Length; i++){
            for(int j=0; j<graph[i].Length; j++){
                List<int> list;
                if(dict.ContainsKey(i))
                    list = dict[i];
                else
                    list = new List<int>();
                list.Add(graph[i][j]);
                dict[i] = list;
            }
        }
```

```
        List<int> cur = new List<int>();
        cur.Add(0);
        GetAllPath(graph, dict, res, cur, 0);
        return res;
    }

    protected void GetAllPath(int[][] graph, Dictionary<int, List<int>> dict,
IList<IList<int>> res, List<int> cur, int node){
        if(node == graph.Length -1){
            res.Add(new List<int>(cur));
            return;
        }
        if(!dict.ContainsKey(node)) return;
        foreach(int nextNode in dict[node]){
            cur.Add(nextNode);
            GetAllPath(graph, dict, res, cur, nextNode);
            cur.RemoveAt(cur.Count() - 1);
        }
    }
}
```

14 Backpack or Ksum

n个数，取k个数，组成和为target
state: f[i][j][t]前个数取j个数出来能否和为t
function: f[i][j][t] = f[i - 1][j - 1][t - a[i]] or
f[i - 1][j][t]
possible? (DP)
total number (DP)
all solutions (recursion)

14.1 Two Sum

Given an array of integers, find two numbers such that they add up to a specific target number.
The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.
You may assume that each input would have exactly one solution.
Input: numbers={2, 7, 11, 15}, target=9
Output: index1=1, index2=2

```
    // using map to store indexes which will be used when return index
    vector<int> twoSum(vector<int>& numbers, int target){
        map<int,int> mymap;
        vector<int> result;
        for(int i=0;i<numbers.size();i++){
            if(mymap.find(target-numbers[i]) == mymap.end()){
                mymap[numbers[i]] = i;
            }else{
```

```
                result.push_back(mymap[target-numbers[i]] + 1);
                result.push_back(i+1);
                return result;
            }
        }
        return result;
    }
```

14.2 3Sum

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 ≤ b ≤ 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)

```
class Solution {
public:
    // T - O(n^3) if use three loops
    // T - O(n^2) if use two pointers
    // generally, for k sum problems, T-O( n^(k-1) )
    vector<vector<int> > threeSum(vector<int> &num) {
        vector<vector<int> > res;
        if(num.size() < 3) return res;
        std::sort(num.begin(),num.end());

        for(int i=0; i<num.size()-2;i++){
            if(i!=0 && num[i] == num[i-1]){     // avoid duplicates
                continue;
            }
            int start = i+1;
            int end = num.size()-1;
            while(start < end){
                int sum = num[start]+num[end]+num[i];
                if(sum == 0){
                    vector<int> vec{num[i],num[start],num[end]};
                    res.push_back(vec);
                    start++;
                    end--;
                    while(start<end && num[start] == num[start-1]){     // avoid
duplicates
                        start++;
                    }
                    while(end > start && num[end] == num[end+1]){       // avoid
duplicates
                        end--;
```

```
                    }
                }else if(sum < 0){
                    start++;
                }else{
                    end--;
                }
            }
        }
        return res;
    }
};
```

14.3 3Sum Closest

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution.
For example, given array S = {-1 2 1 -4}, and target = 1.

```
The sum that is closest to the target is 2. (-1 + 2 + 1 = 2)
```

```
class Solution {
public:
    int threeSumClosest(vector<int> &num, int target) {
        int res = 0;
        if(num.size() < 3) return res;
        int diff = INT_MAX;
        std::sort(num.begin(),num.end());
        for(int i=0;i<num.size()-2;i++){
            if(i!=0 && num[i-1] == num[i]){
                continue;
            }
            int left = i + 1;
            int right = num.size() -1;
            while(left < right){
                int sum = num[i] + num[left] + num[right];
                if(sum == target){
                    return sum;         // notice it wants return sum not diff
                }
                else if(sum < target){
                    left++;
                    while(left < right && num[left] == num[left-1]){    // better
to check duplicates
                        left++;
                    }
                }else{
                    right--;
                    while(right > left && num[right] == num[right+1]){
                        right--;
                    }
```

```
                }
                if(std::abs(diff) > std::abs(sum-target)){
                    diff = sum - target;
                }
            }
        }
        res = target + diff;
        return res;
    }
};
```

14.4 4Sum

Given an array S of n integers, are there elements a, b, c, and d in S such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.
Note:
Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, a ≤ b ≤ c ≤ d)
The solution set must not contain duplicate quadruplets.
For example, given array S = {1 0 -1 0 -2 2}, and target = 0.

```
A solution set is:
(-1,  0, 0, 1)
(-2, -1, 1, 2)
(-2,  0, 0, 2)
```

notes: split to num[i] and 3sum

```
    vector<vector<int> > fourSum(vector<int> &num, int target) {
        vector<vector<int> > res;
        if(num.size() < 4) return res;
        std::sort(num.begin(),num.end());

        for(int i=0; i<num.size()-3;i++){
            if(i!=0 && num[i] == num[i-1]){     // avoid duplicates
                continue;
            }
            for(int j=i+1; j<num.size()-2; j++){
                if(j!=i+1 && num[j] == num[j-1]){
                    continue;
                }
                int left = j+1;
                int right = num.size()-1;
                while(left < right){
                    int sum = num[left]+num[right]+num[i]+num[j];
                    if(sum == target){                              // copy will
end your future
                        vector<int> vec{num[i],num[j],num[left],num[right]};
                        res.push_back(vec);
```

```
                        left++;
                        right--;
                        while(left < right && num[left] == num[left-1]){     //
avoid duplicates
                            left++;
                        }
                        while(right > left && num[right] == num[right+1]){
// avoid duplicates
                            right--;
                        }
                    }else if(sum < target){     // here is target
                        left++;
                    }else{
                        right--;
                    }
                }
            }
        }
        return res;
    }
```

14.5 KSum II

给n个互不相同的数，让你取k个数，问这k个数之和是target的，有多少种方案
For all ksum vectors:

- use map, store k/2 number of elements, then search other k/2 elements. T-O(n^(k/2))
- like 3sum,4sum, first sort, then try all k number groups. T(n^k-1 + O(nlogn)
  For this problem, all possible numbers:
  state: f[n][k][target] 前n个数，取k个数，组成和为target的方案有多少个
  function: f[n][k][target] = f[n-1][k-1][target-a[n]] + f[n-1][k][target]
  // will add code later

15 Stack

- used in DFS traversal iterative version, simulate recursion

15.1 build a stack

- push pop top are O(1), use singly linked list, use one Node top as variable
- implement push, pop, top, empty

```
namespace ConsoleApp
{
    // Stack implemented via Singly linked list
    // Supports: Push, Pop, Peek, Clear, Contains
    public class MyStack
    {
        private Node root;
        public MyStack()
```

```
        {
            root = null;
        }
        public void Push(int value)
        {
            if (root == null)
            {
                root = new Node(value);
                return;
            }
            Node node = new Node(value);
            node.next = root;
            root = node;
        }
        public int Pop()
        {
            if (root == null) return -1;
            Node node = root;
            root = root.next;
            return node == null ? -1 : node.val;
        }
        public int Peek()
        {
            return root == null ? -1 : root.val;
        }
        public void Clear()
        {
            root = null;
        }

        public bool Contains(int value)
        {
            Node node = root;
            while (node != null)
            {
                if (node.val == value)
                    return true;
                node = node.next;
            }
            return false;
        }
    }
}
```

15.2 Min-Stack, implement a min() method

Implement a stack, enable O(1) Push, Pop Top, Min. Where Min() will return the value of minimum number in the stack.

- implement a normal stack using list

- at the same time create a new list (includes prev, cur, next), cur store the current min value, use prev, next to replace
- change if need when push new element and pop if needed
- or if std::stack allowed, just use two stacks

15.3 Evaluate Reverse Polish Notation

Evaluate the value of an arithmetic expression in Reverse Polish Notation.
Valid operators are +, -, , /. *Each operand may be an integer or another expression.*
Some examples:
*["2", "1", "+", "3", ""] -> ((2 + 1) * 3) -> 9*
["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

- push number into stack
- when facing operator, pop two numbers from stack, operator it, then push result into stack
- return st.top();

```
int evalRPN(vector<string> &tokens) {
    stack<int> st;
    string op = "+-*/"; //to check the operator
    if (tokens.size()==0){return 0;}
    for (int i = 0; i<tokens.size();i++){
        string tok = tokens[i];
        int o =op.find(tok); //operator number
        if (o!=-1){
            if (st.size()<2){return -1;}
            else{
                int a = st.top();
                st.pop();
                int b = st.top();
                st.pop();
                if (o==0){st.push(b+a);} //remember the order is b -> a
                if (o==1){st.push(b-a);}
                if (o==2){st.push(b*a);}
                if (o==3){st.push(b/a);}
            }
        }else{
            st.push(atoi(tok.c_str()));
        }
    }
    return st.top();
}
```

16 Queue

- FIFO -- First in First out
- used in customer service, store respondents, and BFS traversal

16.1 build queue

- push pop front empty O(1), build from list
- use two Node pointer, head and end

```
namespace ConsoleApp
{
    // implemented by singly linked list
    // supports Enqueue, Dequeue, Clear, Contains
    public class MyQueue
    {
        private Node root;
        private Node end;
        public MyQueue()
        {
            root = null;
            end = null;
        }

        public void Enqueue(int value)
        {
            if (root == null)
            {
                root = end = new Node(value);
                return;
            }
            end.next = new Node(value);
            end = end.next;
        }
        public int Dequeue()
        {
            if (root == null) return -1;
            int res = root.val;
            root = root.next;
            return res;
        }

        public int Peek()
        {
            if (root == null) return -1;
            return root.val;
        }

        public void Clear()
        {
            root = end = null;
        }
        public bool Contains(int value)
        {
            Node node = root;
            while (node != null)
            {
                if (node.val == value)
                    return true;
```

```
                node = node.next;
            }
            return false;
        }
    }
}
```

16.2 implement a queue by two stacks

```
Q.Push(x):
   S1-Push(x)
Q.Pop():
   if S2.empty then pop s1 and push poped elements in s2
   S2.pop()
Q.Top():
    Similar with Q.Pop()
```

16.3

```

```

17 OOD design

17.1 polymophism

For example: Person* object = new Student(0);

- constructor order: base -> inherited
- destructor order: inherited -> base (base virtual destructor)
  because if base class destrucor is not virtual, delete pointer will just call base class destructor and derived class destructor will not be called. as a result, we want to ensure that the destructor for the most derived class is called.
- execute result:
  Person class constructor called
  student class contructor called
  I am a student
  my name is jaden
  my birthday is 1989
  student class virtual destructor called
  person class virtual destructor called.

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
```

```
struct Person{
        int ID;
        string name;
        virtual void test()=0;
        Person(){
                cout << "Person class constructor called" <<endl;
        }
        virtual ~Person(){
                cout << "person class virtual destructor called." <<endl;
        }
};

class Student : public Person{
private:
        string birth_day;
public:
        Student(int ID){        // cannot use like :ID(ID)
                cout << "student class contructor called" <<endl;
                this->ID = ID;
        }
        void test(){
                cout << "I am a student" << endl;
                name = "jaden";
                cout << "my name is " << name << endl;
                birth_day = "1989"; // birh_day can be use by polymophism in
inherited method
                cout << "my birthday is "<<birth_day <<endl;
        }
        void test2(){}
        ~Student(){
                cout << "student class virtual destructor called"<< endl;
        }
};

int main(){
        Person* object = new Student(1);
        object->test();
        //object->test2();  // person cannot find test2 method
        delete object;
        return 0;
}
```

18 Container

- may use left++ right--, dp, stack...

18.1 Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.
For example,

Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.

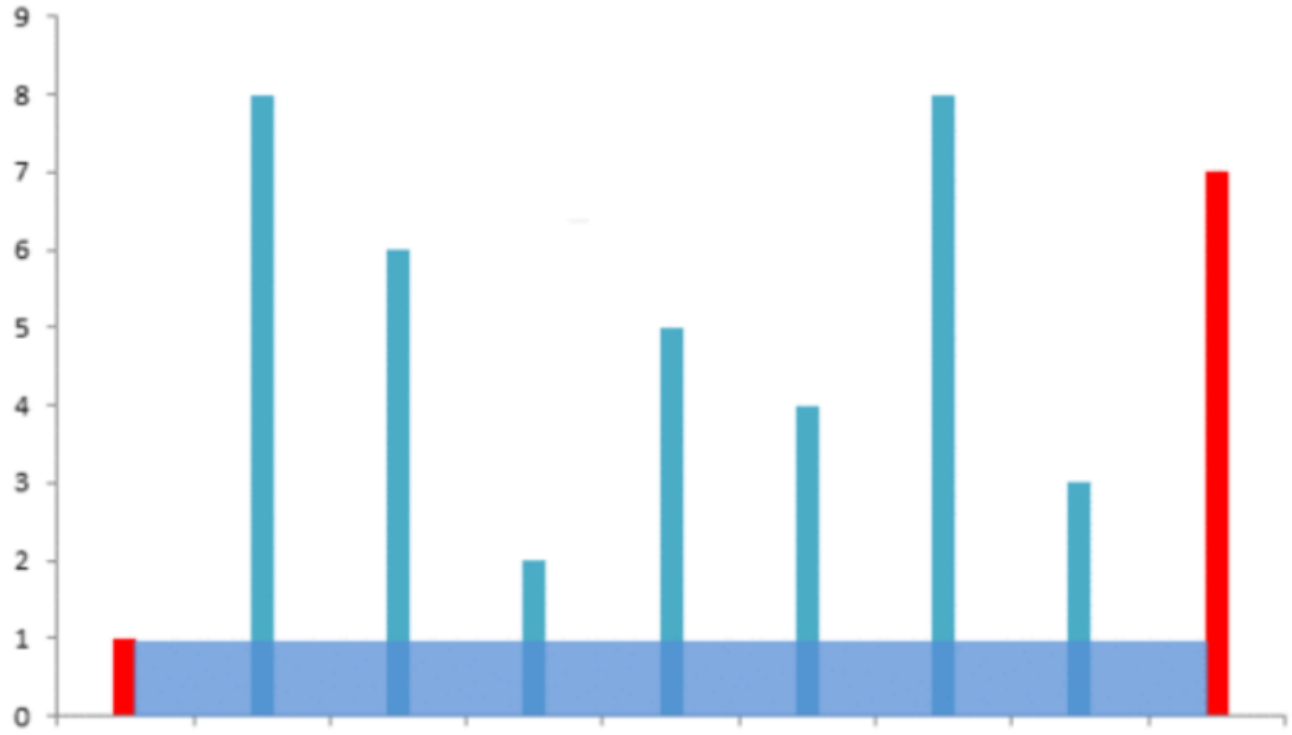
https://leetcode.com/problems/trapping-rain-water/description/

- first solution: level by level, check left and right of curent, if ok, plus 1. However, if highest is way too high, there will be two many levels
- better solution: T-O(n)
    - for cur bar, its min(left highest - right highest) - cur will be the water can be above cur bar.
    - so, traverse from left to remember left highest of cur, and right vice versa
    - finally traverse again, and use min(left highest - right heighest) - cur to give the result

```
int Trap(int A[]) {
    int n = A.Length;
    if(n <= 1) return 0;
    int[] left = new int[n];
    int[] right = new int[n];
    int max = A[0];
    for(int i=1;i<n;i++){
        left[i] = max;
        if(A[i] > max){
            max = A[i];
        }
    }
    max = A[n-1];
    for(int i=n-2;i>=0;i--){
        right[i] = max;
        if(A[i] > max){
            max = A[i];
        }
    }
    int res = 0;
    for(int i=0;i<n;i++){
        int temp = Math.Min(left[i],right[i])-A[i];
        if(temp > 0){     // temp <0, means no water can be above A[i] bar
            res += temp;
        }
    }
    return res;
}
```

18.2 Container With Most Water

Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.



https://leetcode.com/problems/container-with-most-water/description/

- simplest container, does not need to consider other nodes when calculate curent node
- the cur (lower height) with the furthest height have the most water for this lower height
- for any i, the maxium area will be the farthest j that has a[j] > a[i];
- directly start from left and right as farthest means bigiest area
- T - O(n)

```
public class Solution {
    public int MaxArea(int[] height) {
        if(height == null) return 0;
        if(height.Length < 2) return 0;
        int max = 0;
        int left = 0;
        int right = height.Length -1;
        while(left < right){
            int cur = Math.Abs(left - right) * Math.Min(height[left],
height[right]);
            if(cur > max)
                max = cur;
            if(height[left] <= height[right])
                left++;
            else
                right--;
        }
        return max;
```

```
    }
}
```

18.3 largest rectangle in histogram

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.

- use stack to push each element, if st.top() <= curr, continue push next one
- if st.top()>curr, pop all until st.top < curr, at the same time, calculate the max rectangle
- pay attention the end of vector, add -1 at the end to make sure stack will be empty at last
- stack actually holds the index of each element

```
    int largestRectangleArea(vector<int> &height) {
        if (height.size() == 0) {
            return 0;
        }
        std::stack<int> st;
        int max = 0;
        for (int i = 0; i <= height.size(); i++) {
            int curt = (i == height.size()) ? -1 : height[i];     //detect the end
of vector
            while (!st.empty() && curt < height[st.top()]) {
                int h = height[st.top()];
                st.pop();
                int w = st.empty() ? i : i - st.top() - 1;   // how many element h
we have here
                max = std::max(max, h * w);
            }
            st.push(i);
        }
        return max;
    }
```

18.4 Maximal Rectangle

Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing all ones and return its area.

- use previous largerst rectangle area method
- for each row, use a vector to store its 1s, so in this vector, each index and its value is a small rectangle, its actually a largest rectangle area problem.
- then we have just keep a max area when invoke previous method to calculate the area for each line.

```
    int maximalRectangle(vector<vector<char> > &matrix) {
        if(matrix.size() == 0) return 0;
        int rowCount = matrix.size();
        int columnCount = matrix[0].size();
```

```
        vector<vector<int> > rectangle(rowCount, vector<int>(columnCount, 0));
        for(int i = 0; i < rowCount; i++){
                for(int j = 0; j < columnCount;j++){
                        if(i == 0){
                                rectangle[i][j] = matrix[i][j] == '1' ? 1 : 0;
                        }else{
                                rectangle[i][j] = matrix[i][j] == '1' ? 1 +
rectangle[i - 1][j] : 0;
                        }
                }
        }

        int maxArea = 0;
        for(int i = 0; i < rowCount; i++){
                int tem = largestRectangleArea(rectangle[i]); // invoke prev
method
                   maxArea = std::max(tem, maxArea);
        }
        return maxArea;
    }
```

18.5 Gas Station

There are N gas stations along a circular route, where the amount of gas at station i is gas[i].
You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i to its next station (i+1). You begin the journey with an empty tank at one of the gas stations.
Return the starting gas station's index if you can travel around the circuit once, otherwise return -1.

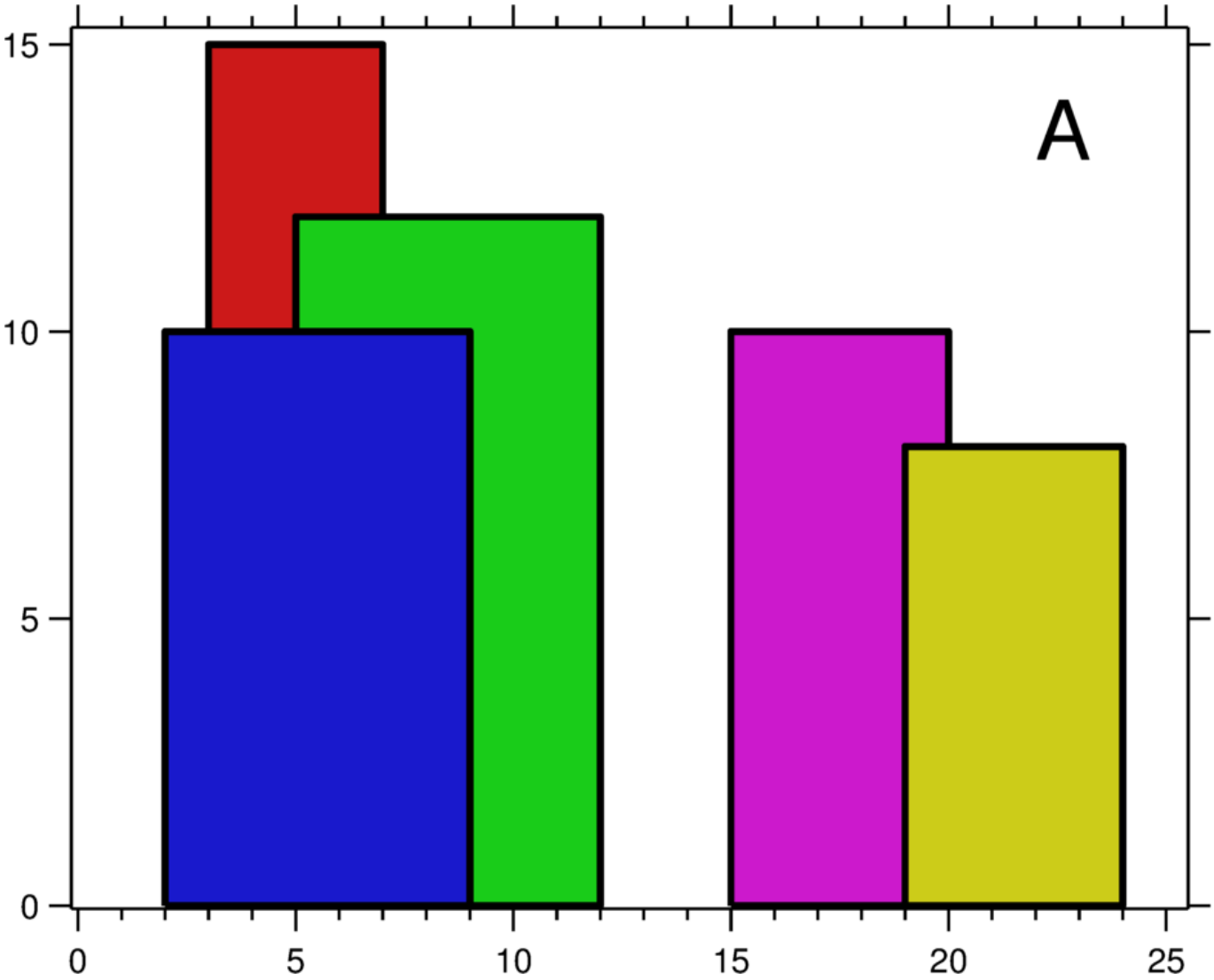
- The solution is guaranteed to be unique.
- O(n^2) two loop solution will work
- O(n) solution:
    - for O(n^2) solution, start from i, we cannot reach j, this mean any index from i to j cannot reach j. so we can just jump starting index from i to j+1.
    - so we can just have one loop to calculate the total tank left, if its positive, it mean there is one starting point.
    - in this loop, if we found we cannot go from gas[i] to gas[i+1], we just save i+1 as a new starting point and goon. total sum can be used to indicated finally do we have this kind of starting point

```
    int canCompleteCircuit(vector<int> &gas, vector<int> &cost) {
        if (gas.size() == 0 || cost.size() == 0 || gas.size() != cost.size()) {
            return -1;
        }
        int sum = 0;
        int total = 0;
        int index = 0;
        for(int i = 0; i<gas.size(); i++) {
            sum += gas[i] - cost[i];   // check can we go from i to i+1 station
            total += gas[i] - cost[i];
            if(sum < 0) {
                index = i+1;  // save new starting point
```

```
                sum = 0;      // reset sum as zero
            }
        }
        return total < 0 ? -1 : index;
    }
```

18.6 The Skyline Problem

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 `[Li, Ri, Hi]`, where Li and Ri are the x coordinates of the left and right edge of the ith building, respectively, and Hi is its height. It is guaranteed that `0 ≤ Li, Ri ≤ INT_MAX, 0 < Hi ≤ INT_MAX, and Ri - Li > 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 `[ [x1,y1], [x2, y2], [x3, y3], ... ]` 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], ...]
```

https://leetcode.com/problems/the-skyline-problem/description/

- skyline problem is easy to understand but complex to solve
- easy way is to add each build one time and modify the points each time T-O(n^2)
- another way is use priority queue and merge sort (need to read the exlaination) from https://leetcode.com/problems/the-skyline-problem/discuss/61192/once-for-all-explanation-with-clean-java-codeon2time-on-space
- easy way should cover company interview

```
// need to be added
```

19 system design

19.1 design twitter

design twitter, qps, redis, memcache, database, master, labor server...

20 Others

20.1 do you have any questions?

Question to HR:

- How would you describe the company culture?
- What type of employees tend to excel at this company?
- Can you tell me more about the interview process?
- How would you describe the work enviroment here—collaborative or independent?

Hiring Manager: Your Future Boss

- What is the ideal candidate?
- What are some challenges one might face in this position?
- What are the most important skillset for the job?
- What are the backgrounds of people in the team?
- What's a typical career path at the company for someone in this role?
- If I am luck enough to get the job, what preparation would you suggest me do?
- Learning/training opportunities

The Executive or high level expert

- How do you think this industry will change in the next five years?
- What do you think is the competitive advantage of our company?
- What's the company's biggest challenge? How is it planning to meet that challenge?

The Coworker

- Could you please describe a typical day?
- How would you describe the work environment at the company?
- Share something about your background.

General quetions：

- What do you particularly like about the company?
- What do you dislike about the company if there is any?
- Could you tell me something about the projects that you are working on? The size of the team. The language the team is adopting?

20.2 briefly describe yourself

20.3 Questions asked

why this company (why google)

- like the product
- like the engineer culture

what is the best or most chalenging work you have done

- most achieved employee of the year 2017

20.4 How to approach a coding problem

- scan the function sigature frist, what input and output
- understand the problem and requirement
- think of a general (slow) solution
- think of a quicker (better) solution
- write the code with corner case in mind
- test the code with 1 to 2 given use case
- explain the time and space complexity

20.10 image reference

21 C# basics

21.1 C# basic syntax

namespace

```
using System;
using System.Linq;
using System.Collections;
using System.Collections.Generics;
```

char

```
char c = 'a';
string str = c.ToString();
char[] chars = new char[] {'a','b','c'};
string str = new string(chars);
```

string

```
string str = "this is a test";
string str = @"this is a test";
for(int i=0; i<str.Length; i++){
    Console.WriteLine(str[i]);
}
string[] strs = str.Split(' ');
string[] strs = str.Split(new string[]{" "}, StringSplitOptions.None);
```

array

```
int[] arr = new int[3];
int[] arr = new int[] {1,2,3,4};
int len = arr.Length;
for(int i=0; i<len; i++){
    Console.WriteLine(arr[i]);
}
arr.IndexOf(3);
arr.LastIndexOf(4);

// sort for array
Array.Sort(nums);
array = array.OrderByDescending(c => c).ToArray();
```

```
int num;
int[] arr = str.Split(' ').Where(o => int.TryParse(o, out num))
                                .Select(o => int.Parse(o))
                                .ToArray();


// 2-dimension array
int[][] arr = new int[m][];
for(int i=0; i<m; i++)
    arr[i] = Enumerable.Repeat(1, n).ToArray();
for(int i=1; i<m; i++){
    for(int j=1; j<n; j++){
        arr[i][j] = arr[i-1][j] + arr[i][j-1];
    }
}

// another way of 2-dimensional array
public class Solution {
    public int MinPathSum(int[,] grid) {
        int m = grid.GetUpperBound(0);
        int n = grid.GetUpperBound(1);
        int[,] a = new int[m+1,n+1];
        a[0,0]=grid[0,0];
        for (int i=1;i<=m;i++){ a[i,0]=a[i-1,0]+grid[i,0];}
        for (int i=1;i<=n;i++){ a[0,i]=a[0,i-1]+grid[0,i];}
        for(int i=1;i<=m;i++){
            for (int j=1;j<=n;j++){
                a[i,j]= Math.Min( a[i,j-1]+grid[i,j], a[i-1,j]+grid[i,j]);
            }
        }
        return a[m,n];
    }
}
```

collections

```
using System.Collections;
using System.Collections.Generics;

Count, Clear(), Contains() => all from ICollection<T>
all collections namepace including Dictionary interited from ICollection<T>


List<int> list = new List<int>();
int len = list.Count;

list.Add(2);
list.Remove(2);
list.RemoveAt(list.Count - 1);
list.IndexOf(2);
```

```
list.LastIndexOf(3);
list.Insert(2, "test"); // insert string "test" into index 2

// sort for list, in place, Linq OrderBy also works, but will be a new list
list.Sort();
list.Sort((a, b) => a.CompareTo(b)); // ascending sort
list.Sort((a, b) => -1* a.CompareTo(b)); // descending sort

// Dictionary and HashSet
Dictionary<string, int> dict = new Dictionary<string, int>();
foreach(string key in dict.Keys)
    if(key == "key1")
        dict[key]++;
}

HashSet<int> hs = new HashSet<int>();
hs.Add(3);
hs.Contains(3);


// Queue
Queue<TreeNode> queue = new Queue<TreeNode>();
queue.Enqueue(node);
while(queue.Count > 0)
    TreeNode curNode = queue.Dequeue();
queue.Clear();

//Stack
Stack<Node> st = new Stack<Node>();
st.Push(node);
st.Pop(node);
st.Peek();
st.Clear();
```

Console input and output

```
string str = Console.ReadLine();
while(Console.ReadLine()){

}
```

21.2 C++ basics

```
MyClass myClass; // allocate in stack
myClass.DoSomething();
myClass.MyValue;

MyClass *myPointer = new MyClass();
```

```
myPointer->DoSometing();
myPointer->MyValue;

MyClass *myPointer = &myClass;
myPointer->DoSomething()
```

22 Company

22.1 Microsoft

LeetCode high frequency: 1 Copy List with Random Pointer

- first way
  - use Dictionary<Node, Node> to store mapping between new node and old node
  - iterate first time for dict, second time for add random pointer link
  - O(n) space (except new list need to be created)
- second way O(1) space
  - iterate first time to create copy of each node, and directly add copy after its original node
  - then cur.next.random = cur.random.next will assign random node to copy node
  - then finally pull out copy node and return a new list

2 Design Tic-Tac-Toe

- how to O(n) check win
  - if n*n grid, then single row, column, diagnal, reverse diagnal has n same value, win
  - normally need to traverse all rows and columns to check win, O(n^2)
  - if two array row[n], column[n], int diag, int rdiag, each time one step will also populate them with (1 or -1) if they any one = n or -n, then win

3 Reverse Linked List

- reverse each pointer by connecting cur to prev
- second way, recursive

4 Battlefield in a Board basically grid with X and ., find connected X piece count, and each piece is not connected

- trick is to only need to find total head
- head is the one that has grid[r][c]=='X' && grid[r-1][c]=='.' && grid[r][c-1]=='.'

5 Integer to English Words

- trick is to use num%1000 and num/1000
- all corner cases 0? Or 1000010? (middle chunk is zero and should not be printed out)

```
// very clean Java code
private final String[] LESS_THAN_20 = {"", "One", "Two", "Three", "Four", "Five",
"Six", "Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve", "Thirteen",
"Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"};
private final String[] TENS = {"", "Ten", "Twenty", "Thirty", "Forty", "Fifty",
```

```
"Sixty", "Seventy", "Eighty", "Ninety"};
private final String[] THOUSANDS = {"", "Thousand", "Million", "Billion"};

public String numberToWords(int num) {
    if (num == 0) return "Zero";

    int i = 0;
    String words = "";

    while (num > 0) {
        if (num % 1000 != 0)
            words = helper(num % 1000) +THOUSANDS[i] + " " + words;
        num /= 1000;
        i++;
    }

    return words.trim();
}

private String helper(int num) {
    if (num == 0)
        return "";
    else if (num < 20)
        return LESS_THAN_20[num] + " ";
    else if (num < 100)
        return TENS[num / 10] + " " + helper(num % 10);
    else
        return LESS_THAN_20[num / 100] + " Hundred " + helper(num % 100);
}
```

6 Remove Comment remove comment from C++ code // and /* */ *carefully check // and /* and */, then ignore comment, return new array as results

7 Excel sheet column number

- easy one, AA -> 27, just need (c-'A'+1)*26^i

8 Add Two Numbers two linked list with reverse order add each node from left to right, send the carry value to next node

9 Add Two Numbers II correct order, either reverse linked list then like Q8, or push both to Stack and then plus each node

10 Merge Sorted Array two sorted array, merge them

- as array 1 has enouogh room for both array, so start filling in end of array 1 by comparing end of both array
- similar as merge linked lists, just start from end
- create a new array is also an option but with O(n) space

11 Reverse words in a string

- first way O(1) space:
  - reverse string #1, reverse each word #2, reverse last word if no space at end #3
- second way, O(n) space, store in list
  - just parse string, find each word by finding space, then store in list, finally output it reversely
  - use curword variable to check cur==' ' && curword != "", then new word found, cur!='', add to curword

12 Spiral Matrix need to iterate through grid in Spiral order

- define rowbegin, rowend, columnbegin, columend, while(rb <re && cb>), 4 for loop for 4 directions inside
- O(N) time, as total n elements

13 Binary tree Level order traversal

- type 1: traverse from left to right
  - use BFS Queue, count of current queue to define level

22.2 Square

1 design json model and replace all digits 0-9 with * for all values

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp
{
    public enum JsonValueType
    {
        String,
        Integer,
        Decimal,
        Boolean,
        Object,
        Array,
        Null
    }

    public class JsonPair
    {
        public string Key { get; set;}

        private JsonValueType _type = JsonValueType.Null;
        public JsonValueType Type
        {
            get
            {
                return _type;
            }
```

```
            private set
            {
                _type = value;
            }
        }

        private object _value;
        public object Value
        {   get
            {
                return _value;
            }
            set
            {
                if (value == null)
                {
                    _value = null;
                    _type = JsonValueType.Null;
                    return;
                }
                if(value is String)
                {
                    _value = value;
                    _type = JsonValueType.String;
                }
                else if(value is Boolean)
                {
                    _value = value;
                    _type = JsonValueType.Boolean;
                }
                else if(value is int)
                {
                    _value = value;
                    _type = JsonValueType.Boolean;
                }
                else if(value is JsonObject)
                {
                    _value = value;
                    _type = JsonValueType.Object;
                }
                else if(value is IEnumerable<JsonObject>)
                {
                    _value = value;
                    _type = JsonValueType.Array;
                }
            }
        }
    }

    public class JsonObject
    {
        private List<JsonPair> _list = new List<JsonPair>();
        public virtual Object this[string key]
        {
```

```
            get
            {
                foreach(JsonPair jp in _list)
                {
                    if (jp.Key == key)
                        return jp.Value;
                }
                throw new Exception("Cannot find key " + key);
            }
            set
            {
                foreach(JsonPair jp in _list)
                {
                    if(jp.Key == key)
                    {
                        jp.Value = value;
                    }
                }
            }
        }

        public virtual List<JsonPair> GetList()
        {
            return _list;
        }

        public virtual void Add(JsonPair jp)
        {
            _list.Add(jp);
        }
    }

    /*
{
'key1' : 'value1',
'key2' : {
            'key2.1' : 'value 2.1',
            'key2.2' : 'value 2.2'
         }
'key3' : {
            'key3.1' : 'value 3.1',
            'key3.2' : 'value 3.2'
        }
}
     */


/*
Issue during interview:
* string.Replace will not modify the string, just return a new string
* directly modify input object as did not have clone method to use directly
* cannot return as it will break the for loop
* JsonObject.list is bad, should give indexer to use
* Only support string and JSONObject, should also support other types (int, bool
```

```
etc.)

*/
    public class JsonModel
    {
        public void Test()
        {
            JsonObject jo = new JsonObject();
            JsonPair jp = new JsonPair();
            jp.Key = "key1";
            jp.Value = "value 1.000";
            jo.Add(jp);

            JsonPair jp2 = new JsonPair();
            jp2.Key = "key2";
            JsonObject jo2 = new JsonObject();
            JsonPair jp3 = new JsonPair();
            jp3.Key = "key2.1";
            jp3.Value = "valuetest 2.1";
            jo2.Add(jp3);
            jp2.Value = jo2;
            jo.Add(jp2);

            List<JsonObject> jsArray = new List<JsonObject>();
            JsonObject jo3 = new JsonObject();
            JsonPair jp4 = new JsonPair() { Key = "key3.1", Value = "valuetest
3.1" };
            jo3.Add(jp4);
            jsArray.Add(jo3);
            JsonPair jp5 = new JsonPair() { Key = "key3", Value = jsArray };
            jo.Add(jp5);

            EncodeJsonString(jo);
            Console.WriteLine(jo["key1"]);
            Console.WriteLine((jo["key2"] as JsonObject)["key2.1"]);
            Console.WriteLine(((jo["key3"] as IList<JsonObject>)[0] as JsonObject)
["key3.1"]);
            System.Diagnostics.Debug.Assert(1 == 2, "1 does not equal to 2");
        }

        public void EncodeJsonString(JsonObject jo)
        {
            if (jo == null) return;
            foreach(JsonPair jp in jo.GetList())
            {
                if (jp.Type == JsonValueType.String)
                {
                    string curValue = jp.Value as String;
                    for (int i = 0; i < curValue.Length; i++)
                    {
                        if (curValue[i] >= '0' && curValue[i] <= '9')
                            curValue = curValue.Replace(curValue[i], '*');
                    }
                    jp.Value = curValue;
```

```
                }
                else if (jp.Type == JsonValueType.Object)
                {
                    EncodeJsonString(jp.Value as JsonObject);
                }
                else if (jp.Type == JsonValueType.Array)
                {
                    foreach(JsonObject v in jp.Value as IEnumerable<JsonObject>)
                    {
                        EncodeJsonString(v);
                    }
                }
            }
        }
    }
}
```