LeetCode Blind 75 Questions

A list of Blind 75 Leetcode problems which is very useful, enjoy!

Array

1. Two Sum

java:

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> hashmap = new HashMap<>();
        for (int i = 0; i< nums.length; i++){
            if (hashmap.containsKey(nums[i]))
                return new int[]{hashmap.get(nums[i]), i};
            hashmap.put(target-nums[i], i);
        }
        return new int[2];
    }
}
```

python3:

```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        dic = {}
        for i,num in enumerate(nums):
            if num in dic:
                return [dic[num],i]
            dic[target - num] = i
```

15. 3Sum

java:

```java
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        Arrays.sort(nums);
        for (int i = 0; i < nums.length-2; i++){
            if (i > 0 && nums[i] == nums[i-1]) continue;
            int left = i+1, right = nums.length-1;
            while (left < right){
                int threesome = nums[i] + nums[left] + nums[right];
                if (threesome == 0){
                    res.add(List.of(nums[i], nums[left], nums[right]));
                    left ++;
                    right --;
                    while (nums[left] == nums[left-1] && left < right)
                        left ++;
                    while (nums[right] == nums[right+1] && left < right)
                        right --;
                }else if (threesome > 0)
                    right --;
                else
```

```
                    left ++;
                }
            }
            return res;
        }
    }
```

python3:

```python
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        result = []
        nums.sort()
        for i in range(len(nums)-2):
            firstNum = nums[i]
            if i>0 and nums[i] == nums[i-1]:
                continue
            left, right = i+1, len(nums)-1
            while left < right:
                threeSome = firstNum + nums[left] + nums[right]
                if threeSome > 0:
                    right -= 1
                elif threeSome < 0:
                    left += 1
                else:
                    result.append([firstNum,nums[left],nums[right]])
                    left += 1
                    right -= 1
                    while nums[left] == nums[left-1] and left < right:
                        left += 1
                    while nums[right] == nums[right+1] and left < right:
                        right -= 1
        return result
```

121. Best Time to Buy and Sell Stock

java:

```java
class Solution {
    public int maxProfit(int[] prices) {
        int minprice = prices[0];
        int res = 0;
        for (int price : prices){
            minprice = Math.min(price, minprice);
            res = Math.max(price-minprice, res);
        }
        return res;
    }
}
```

python3:

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        minprice = prices[0]
        res = 0
        for i in range(1, len(prices)):
            if prices[i] < minprice:
                minprice = prices[i]
            res = max(prices[i]-minprice, res)
        return res
```

217. Contains Duplicate

java:

```java
class Solution {
    public boolean containsDuplicate(int[] nums) {
        Set<Integer> set = new HashSet<>();
        for (int num : nums){
            if (set.contains(num))
                return true;
            else
                set.add(num);
        }
        return false;
    }
}
```

python3:

```python
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        res = set()
        for num in nums:
            if num in res: return True
            else: res.add(num)
        return False
```

238. Product of Array Except Self

java:

```java
class Solution {
    public int[] productExceptSelf(int[] nums) {
        int result [] = new int [nums.length];
        int prefixprod = 1;
        for (int i = 0; i < nums.length; i++){
            result[i] = prefixprod;
            prefixprod *= nums[i];
```

```
        }
        prefixprod = 1;
        for (int i = nums.length-1; i > -1; i--){
            result[i] *= prefixprod;
            prefixprod *= nums[i];
        }
        return result;
    }
}
```

python3:

```
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        result = [1 for _ in range(len(nums))]
        prefixprod = 1
        for i,num in enumerate(nums):
            result[i] = prefixprod
            prefixprod *= num
        prefixprod = 1
        for i in range(len(nums)-1, -1, -1):
            result[i] *= prefixprod
            prefixprod *= nums[i]
        return result
```

53. Maximum Subarray

java:

```
class Solution {
    public int maxSubArray(int[] nums) {
        int[] opt = new int[nums.length];
        int maxresult = nums[0];
        opt[0] = nums[0];
        for (int i=1; i<nums.length; i++){
            opt[i] = Math.max(nums[i]+opt[i-1], nums[i]);
            if (opt[i] > maxresult) maxresult = opt[i];
        }
        return maxresult;
    }
}
```

python3:

```python
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        totalSum = 0
        maxSum = nums[0]
        for num in nums:
            totalSum += num
            maxSum = max(totalSum, maxSum)
            if totalSum<0: totalSum = 0
        return maxSum
```

152. Maximum Product Subarray

java:

```java
class Solution {
    public int maxProduct(int[] nums) {
        int maxproduct = Integer.MIN_VALUE;
        int curproduct = 1;
        for (int num : nums){
            curproduct *= num;
            maxproduct = Math.max(maxproduct, curproduct);
            if (num == 0)
                curproduct = 1;
        }
        curproduct = 1;
        for (int i = nums.length-1; i >= 0; i--){
            curproduct *= nums[i];
            maxproduct = Math.max(maxproduct, curproduct);
            if (nums[i] == 0)
                curproduct = 1;
        }
        return maxproduct;
    }
}
```

python3:

```python
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        maxproduct = nums[0]
        curproduct = 1
        for num in nums:
            curproduct *= num
            maxproduct = max(maxproduct, curproduct)
            if num == 0:
                curproduct = 1
        curproduct = 1
        for i in range(len(nums)-1, -1, -1):
            curproduct *= nums[i]
            maxproduct = max(maxproduct, curproduct)
```

```
            if nums[i] == 0:
                curproduct = 1
        return maxproduct
```

153. Find Minimum in Rotated Sorted Array

java:

```
class Solution {
    public int findMin(int[] nums) {
        int left = 0, right = nums.length-1;
        while (left <= right){
            if (nums[left] < nums[right])
                return nums[left];
            int mid = left + (right-left)>>1;
            if (nums[left] < nums[mid])
                left = mid + 1;
            else
                right = mid;
        }
        return nums[left];
    }
}
```

python3:

```
class Solution:
    def findMin(self, nums: List[int]) -> int:
        left,right = 0,len(nums)-1
        while left < right:
            if nums[left] < nums[right]:
                return nums[left]
            mid = (left+right)//2
            if nums[right] < nums[mid]:
                left = mid + 1
            else:
                right = mid
        return nums[left]
```

33. Search in Rotated Sorted Array

java:

```
class Solution {
    public int search(int[] nums, int target) {
        int left = 0, right = nums.length-1;
        while (left <= right){
            int mid = left + (right-left)/2;
            if (nums[mid]==target)
                return mid;
```

```
            if (nums[left] <= nums[mid]) {
                if (nums[left] <= target && target <= nums[mid])
                    right = mid - 1;
                else
                    left = mid + 1;
            } else {
                if (nums[mid] <= target && target <= nums[right])
                    left = mid + 1;
                else
                    right = mid - 1;
            }
        }
        return -1;
    }
}
```

python3:

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums)-1
        while left <= right:
            mid = left + (right-left)//2
            if nums[mid]==target:
                return mid
            if nums[left] <= nums[mid]:
                if nums[left] <= target and target <= nums[mid]:
                    right = mid - 1
                else:
                    left = mid + 1
            else:
                if nums[mid] <= target and target <= nums[right]:
                    left = mid + 1
                else:
                    right = mid - 1
        return -1
```

11. Container With Most Water

java:

```
class Solution {
    public int maxArea(int[] height) {
        int left = 0, right = height.length-1;
        int maxarea = 0;
        while (left < right){
            maxarea = Math.max(maxarea, Math.min(height[left], height[right]) *
(right - left));
            if (height[left] < height[right])
                left ++;
            else
```

```
                right --;
            }
            return maxarea;
        }
    }
```

python3:

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        left, right = 0, len(height)-1
        resarea = 0
        while left < right:
            resarea = max(resarea, (right-left)*min(height[left], height[right]))
            if height[left] < height[right]:
                left += 1
            else:
                right -= 1
        return resarea
```

Binary

371. Sum of Two Integers

java:

```
class Solution {
    public int getSum(int a, int b) {
        while (b != 0){
            int carry = (a & b) << 1;
            a ^= b;
            b = carry;
        }
        return a;
    }
}
```

python3:

```
class Solution:
    def getSum(self, a: int, b: int) -> int:
        while b != 0:
            a, b = (a ^ b) & 0xFFFFFFFF, ((a & b) << 1) & 0xFFFFFFFF
        return a if a <= 0x7FFFFFFF else ~(a^0xFFFFFFFF)
```

191. Number of 1 Bits

java:

```java
public class Solution {
    // you need to treat n as an unsigned value
    public int hammingWeight(int n) {
        int count = 0;
        while (n!=0){
            n &= (n-1);
            count ++;
        }
        return count;
    }
}
```

python3:

```python
class Solution:
    def hammingWeight(self, n: int) -> int:
        count = 0
        while n:
            n &= (n-1)
            count += 1
        return count
```

338. Counting Bits

java:

```java
class Solution {
    public int[] countBits(int n) {
        int[] res = new int[n+1];
        for (int i = 1; i < n+1; i++){
            if ((i&1)==1)
                res[i] = res[i-1] + 1;
            else
                res[i] = res[i/2];
        }
        return res;
    }
}
```

python3:

```python
class Solution:
    def countBits(self, n: int) -> List[int]:
        res = [0]
        for i in range(1, n+1):
            if (i&1) == 1:
                res.append(res[-1] + 1)
            else:
                res.append(res[i//2])
        return res
```

268. Missing Number

java:

```java
class Solution {
    public int missingNumber(int[] nums) {
        int res = 0;
        for (int i = 0; i < nums.length; i ++){
            res ^= i;
            res ^= nums[i];
        }
        res ^= nums.length;
        return res;
    }
}
```

python3:

```python
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        n = 0
        for i,num in enumerate(nums):
            n ^= i
            n ^= num
        n ^= len(nums)
        return n
```

190. Reverse Bits

java:

```java
public class Solution {
    // you need treat n as an unsigned value
    public int reverseBits(int n) {
        int res = 0;
        for (int i = 0; i < 32; i++){
            res = (res<<1) | (n&1);
            n >>= 1;
        }
        return res;
    }
}
```

python3:

```python
class Solution:
    def reverseBits(self, n: int) -> int:
        res = 0
        for i in range(32):
            res = (res << 1) | (n & 1)
            n >>= 1
        return res
```

Dynamic Programming

70. Climbing Stairs

java:

```java
class Solution {
    public int climbStairs(int n) {
        if (n==1) return 1;
        int[] opt = new int[n];
        opt[0] = 1;
        opt[1] = 2;
        for (int i=2; i<n; i++){
            opt[i] = opt[i-1] + opt[i-2];
        }
        return opt[n-1];
    }
}
```

python3:

```python
class Solution:
    def climbStairs(self, n: int) -> int:
        if n==1: return 1
        opt = [0 for _ in range(n)]
        opt[0],opt[1] = 1,2
        for i in range(2,n):
            opt[i] = opt[i-1] + opt[i-2]
        return opt[-1]
```

322. Coin Change

Complete knapsack problem

java:

```java
class Solution {
    public int coinChange(int[] coins, int amount) {
        int maxNum = amount+1;
        int[] opt = new int[maxNum];
        Arrays.fill(opt, maxNum);
        opt[0] = 0;
        for (int coin : coins) {
            for (int j = coin; j < maxNum; j ++) {
                opt[j] = Math.min(opt[j], opt[j-coin] + 1);
            }
        }
        if (opt[amount] < maxNum)
            return opt[amount];
        else return -1;
    }
}
```

python3:

```python
class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        maxNum = amount+1
        opt = [maxNum for _ in range(amount+1)]
        opt[0] = 0
        for coin in coins:
            for j in range(coin, maxNum):
                opt[j] = min(opt[j], opt[j-coin]+1)
        if opt[-1] < maxNum:
            return opt[-1]
        return -1
```

300. Longest Increasing Subsequence

java:

```java
class Solution {
    public int lengthOfLIS(int[] nums) {
        if (nums.length == 1) return 1;
        int result = 0;
        int[] opt = new int[nums.length];
        Arrays.fill(opt, 1);
        for (int i = 1; i < nums.length; i++){
            for (int j = 0; j < i; j ++) {
                if (nums[i] > nums[j])
                    opt[i] = Math.max(opt[i], opt[j]+1);
            }
            result = Math.max(result, opt[i]);
        }
        return result;
    }
}
```

python3:

```python
class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        if len(nums) <= 1:
            return len(nums)
        opt = [1 for _ in range(len(nums))]
        result = 0
        for i in range(1, len(nums)):
            for j in range(i):
                if nums[i] > nums[j]:
                    opt[i] = max(opt[i], opt[j]+1)
            result = max(result, opt[i])
        return result
```

1143. Longest Common Subsequence

java:

```java
class Solution {
    public int longestCommonSubsequence(String text1, String text2) {
        char[] array1 = text1.toCharArray();
        char[] array2 = text2.toCharArray();
        int[][] opt = new int[array1.length+1][array2.length+1];
        for (int i=1; i<array1.length+1; i++){
            for (int j=1; j<array2.length+1; j++){
                if (array1[i-1] == array2[j-1])
                    opt[i][j] = opt[i-1][j-1] + 1;
                else
```

```
                    opt[i][j] = Math.max(opt[i][j-1], opt[i-1][j]);
                }
            }
            return opt[array1.length][array2.length];
        }
    }
```

python3:

```
class Solution:
    def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        opt = [[0 for _ in range(len(text2)+1)] for _ in range(len(text1)+1)]
        for i in range(1, len(text1)+1):
            for j in range(1, len(text2)+1):
                if text1[i-1] == text2[j-1]:
                    opt[i][j] = opt[i-1][j-1] + 1
                else:
                    opt[i][j] = max(opt[i-1][j], opt[i][j-1])
        return opt[-1][-1]
```

139. Word Break

java:

```
class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        boolean[] opt = new boolean[s.length()+1];
        Set<String> set = new HashSet<>(wordDict);
        opt[0] = true;
        for (int i=1; i<s.length()+1; i++){
            for (int j=0; j<i; j++){
                if (opt[j] && set.contains(s.substring(j, i))){
                    opt[i] = true;
                }
            }
        }
        return opt[s.length()];
    }
}
```

python3:

```python
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        opt = [False for _ in range(len(s)+1)]
        opt[0] = True
        for i in range(1,len(s)+1):
            for j in range(i):
                word = s[j:i]
                if opt[j] and word in wordDict:
                    opt[i] = True
        return opt[len(s)]
```

377. Combination Sum IV

Complete knapsack problem

java:

```java
class Solution {
    public int combinationSum4(int[] nums, int target) {
        int[] opt = new int[target+1];
        opt[0] = 1;
        for (int i=1; i<target+1; i++)
            for (int num : nums)
                if (i >= num)
                    opt[i] += opt[i - num];
        return opt[target];
    }
}
```

python3:

```python
class Solution:
    def combinationSum4(self, nums: List[int], target: int) -> int:
        opt = [0 for _ in range(target+1)]
        opt[0] = 1
        for i in range(1, target+1):
            for j in nums:
                if i>=j:
                    opt[i] += opt[i - j]
        return opt[-1]
```

198. House Robber

java:

```java
class Solution {
    public int rob(int[] nums) {
        if (nums.length == 1) return nums[0];
        int[] opt = new int[nums.length];
        opt[0] = nums[0];
        opt[1] = Math.max(nums[0], nums[1]);
        for (int i = 2; i<nums.length; i++){
            opt[i] = Math.max(opt[i-2]+nums[i], opt[i-1]);
        }
        return opt[nums.length-1];
    }
}
```

python3:

```python
class Solution:
    def rob(self, nums: List[int]) -> int:
        if len(nums) == 1:
            return nums[0]
        opt = [0 for _ in range(len(nums))]
        opt[0] = nums[0]
        opt[1] = max(nums[0], nums[1])
        for i in range(2,len(nums)):
            opt[i] = max(opt[i-2]+nums[i], opt[i-1])
        return opt[-1]
```

213. House Robber II

java:

```java
class Solution {
    public int rob(int[] nums) {
        if (nums.length==1) return nums[0];
        if (nums.length==2) return Math.max(nums[0],nums[1]);
        return Math.max(sub_rob(nums, 0, nums.length-2), sub_rob(nums, 1,
nums.length-1));
    }
    private int sub_rob(int[] nums, int start, int end){
        int prev = 0, curr = 0;
        for (int i = start; i < end + 1; i++){
            int temp = prev;
            prev = curr;
            curr = Math.max(temp+nums[i], curr);
        }
        return curr;
    }
}
```

python3:

```python
class Solution:
    def rob(self, nums: List[int]) -> int:
        if len(nums) <= 2: return max(nums)
        def sub_rob(sub_nums, start, end):
            if end == start: return nums[start]
            prev, curr = 0, 0
            for i in range(start, end + 1):
                prev, curr = curr, max(prev+nums[i], curr)
            return curr
        return max(sub_rob(nums, 0, len(nums)-2), sub_rob(nums, 1, len(nums)-1))
```

91. Decode Ways

java:

```java
class Solution {
    public int numDecodings(String s) {
        char[] charArray = s.toCharArray();
        if (charArray[0]=='0') return 0;
        int[] opt = new int[charArray.length+1];
        opt[0] = 1;
        for (int i=1; i<charArray.length+1; i++){
            if (charArray[i-1] != '0')
                opt[i] = opt[i-1];
            if (i>=2) {
                int x = (charArray[i-2]-'0')*10 + (charArray[i-1]-'0');
                if (10<=x && x<=26) opt[i] += opt[i-2];
            }
        }
        return opt[charArray.length];
    }
}
```

python3:

```python
class Solution:
    def numDecodings(self, s: str) -> int:
        if s[0] == '0': return 0
        opt = [0 for _ in range(len(s) + 1)]
        opt[0] = 1
        for i in range(1, len(s)+1):
            if s[i-1] != "0":
                opt[i] = opt[i-1]
            if i > 1:
                num = int(s[i-2: i])
                if 10 <= num <= 26:
                    opt[i] += opt[i-2]
        return opt[-1]
```

62. Unique Paths

java:

```java
class Solution {
    public int uniquePaths(int m, int n) {
        int[] opt = new int[n];
        Arrays.fill(opt, 1);
        for (int i=1; i<m; i++){
            for (int j=1; j<n; j++){
                opt[j] = opt[j-1] + opt[j];
            }
        }
        return opt[n-1];
    }
}
```

python3:

```python
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        opt = [1 for _ in range(n)]
        for i in range(1,m):
            for j in range(1,n):
                opt[j] = opt[j] + opt[j-1]
        return opt[n-1]
```

55. Jump Game

java:

```java
class Solution {
    public boolean canJump(int[] nums) {
        if (nums.length == 1) return true;
        int max = nums[0];
        for (int i = 1; i < nums.length - 1; i++) {
            if (max == 0) return false;
            max--;
            max = Math.max(max, nums[i]);
        }
        return max > 0;
    }
}
```

python3:

```python
class Solution:
    def canJump(self, nums: List[int]) -> bool:
        if len(nums)==1: return True
        steps = nums[0]
        for i in range(1, len(nums)-1):
            if steps == 0: return False
            steps -= 1
            steps = max(steps, nums[i])
        return steps > 0
```

Knapsack recursion formula

- Ask if you can fill the backpack (or how much you can hold at most)
  - `dp[j] = max(dp[j], dp[j - nums[i]] + nums[i]);`
- Ask how many ways to fill a backpack
  - `dp[j] += dp[j - nums[i]]`
- Ask the maximum value of the backpack full
  - `dp[j] = max(dp[j], dp[j - weight[i]] + value[i])`
- Ask the minimum number of all items in a full backpack
  - `dp[j] = min(dp[j - coins[i]] + 1, dp[j])`

Graph

133. Clone Graph

java:

```java
class Solution {
    public Node cloneGraph(Node node) {
        if (node == null) return null;
        Map<Integer, Node> visited = new HashMap<>();
        visited.put(node.val, new Node(node.val));
        Queue<Node> queue = new LinkedList<>();
        queue.add(node);
        while (! queue.isEmpty()){
            Node orinode = queue.poll();
            Node cloneNode = visited.get(orinode.val);
            for (Node ngbr : orinode.neighbors){
                if (! visited.containsKey(ngbr.val)){
                    visited.put(ngbr.val, new Node(ngbr.val));
                    queue.add(ngbr);
                }
                cloneNode.neighbors.add(visited.get(ngbr.val));
            }
        }
        return visited.get(node.val);
    }
```

```
}
```

python3:

```python
class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':
        if not node: return node
        queue = deque([node])
        dic = {node.val: Node(node.val)}
        while queue:
            orinode = queue.popleft()
            clonenode = dic[orinode.val]
            for ngbr in orinode.neighbors:
                if ngbr.val not in dic:
                    dic[ngbr.val] = Node(ngbr.val)
                    queue.append(ngbr)
                clonenode.neighbors.append(dic[ngbr.val])
        return dic[node.val]
```

207. Course Schedule

java:

```java
class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
        ArrayList<Integer>[] preqmap = new ArrayList[numCourses];
        for (int i = 0; i < numCourses; i ++)
            preqmap[i] = new ArrayList<>();
        int[] preqnum = new int[numCourses];
        for (int[] prerequisite : prerequisites){
            preqmap[prerequisite[1]].add(prerequisite[0]);
            preqnum[prerequisite[0]] ++;
        }
        Queue<Integer> queue = new LinkedList<>();
        for (int i = 0; i < preqnum.length; i++){
            if (preqnum[i] == 0)
                queue.add(i);
        }
        int result = 0;

        while (queue.size()>0){
            int u = queue.poll();
            result ++;
            for (int v : preqmap[u]){
                preqnum[v] --;
                if (preqnum[v] == 0)
                    queue.add(v);
            }
        }
        return result == numCourses;
    }
```

```
}
```

python3:

```
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
        graph = defaultdict(list)
        numpreq = [0] * numCourses
        result = 0;
        for u, v in prerequisites:
            graph[v].append(u)
            numpreq[u] += 1
        queue = Deque([u for u in range(numCourses) if numpreq[u] == 0])
        while queue:
            u = queue.popleft()
            result += 1
            for v in graph[u]:
                numpreq[v] -= 1
                if numpreq[v] == 0:
                    queue.append(v)
        return result == numCourses
```

417. Pacific Atlantic Water Flow

java:

```
class Solution {
    public List<List<Integer>> pacificAtlantic(int[][] heights) {
        int rows = heights.length, cols = heights[0].length;
        boolean[][] pacific = new boolean[rows][cols];
        boolean[][] atlantic = new boolean[rows][cols];
        List<List<Integer>> res = new ArrayList<>();
        for (int i = 0; i<rows; i++){
            dfs(heights, i, 0, rows, cols, pacific, heights[i][0]);
            dfs(heights, i, cols-1, rows, cols, atlantic, heights[i][cols-1]);
        }
        for (int j = 0; j< cols; j++){
            dfs(heights, 0, j, rows, cols, pacific, heights[0][j]);
            dfs(heights, rows-1, j ,rows, cols, atlantic, heights[rows-1][j]);
        }
        for (int i = 0; i < rows; i ++){
            for (int j = 0; j < cols; j ++) {
                if (pacific[i][j] & atlantic[i][j])
                    res.add(List.of(i, j));
            }
        }
        return res;
    }
    public void dfs (int[][] heights, int i, int j, int rows, int cols, boolean[][]
visited, int prevHeight){
        if (i < 0 || i >= rows || j < 0 || j >= cols)
```

```
            return;
        if (visited[i][j] || prevHeight > heights[i][j])
            return;
        visited[i][j] = true;
        dfs(heights, i-1, j, rows, cols, visited, heights[i][j]);
        dfs(heights, i+1, j, rows, cols, visited, heights[i][j]);
        dfs(heights, i, j-1, rows, cols, visited, heights[i][j]);
        dfs(heights, i, j+1, rows, cols, visited, heights[i][j]);
    }
}
```

python3:

```
class Solution:
    def pacificAtlantic(self, heights: List[List[int]]) -> List[List[int]]:
            rows, cols = len(heights), len(heights[0])
            pacific = [[False for _ in range(cols)] for _ in range(rows)]
            atlantic = [[False for _ in range(cols)] for _ in range(rows)]
            res = []
            for i in range(rows):
                self.dfs(heights, i, 0, rows, cols, pacific, heights[i][0])
                self.dfs(heights, i, cols-1, rows, cols, atlantic, heights[i][cols-
1])
            for j in range(cols):
                self.dfs(heights, 0, j, rows, cols, pacific, heights[0][j])
                self.dfs(heights, rows-1, j ,rows, cols, atlantic, heights[rows-1]
[j])
            for i in range(rows):
                for j in range(cols):
                    if pacific[i][j] & atlantic[i][j]:
                        res.append([i,j])
            return res

    def dfs(self, heights, i, j, rows, cols, visited, prevHeight):
        if i < 0 or i >= rows or j < 0 or j >= cols:
            return
        if visited[i][j] or prevHeight > heights[i][j]:
            return
        visited[i][j] = True
        self.dfs(heights, i-1, j, rows, cols, visited, heights[i][j])
        self.dfs(heights, i+1, j, rows, cols, visited, heights[i][j])
        self.dfs(heights, i, j-1, rows, cols, visited, heights[i][j])
        self.dfs(heights, i, j+1, rows, cols, visited, heights[i][j])
```

200. Number of Islands

java:

```
class Solution {
    public int numIslands(char[][] grid) {
        int res = 0;
```

```
        for (int i = 0; i < grid.length; i ++) {
            for (int j = 0; j < grid[0].length; j ++) {
                if (grid[i][j] == '1'){
                    dfs(grid, i, j);
                    res ++;
                }
            }
        }
        return res;
    }

    public void dfs(char[][] grid, int i, int j) {
        if (grid[i][j] == '0') return;
        grid[i][j] = '0';
        if (i > 0) dfs(grid, i-1, j);
        if (i < grid.length - 1) dfs(grid, i+1, j);
        if (j > 0) dfs(grid, i, j-1);
        if (j < grid[0].length - 1) dfs(grid, i, j+1);
    }
}
```

python3:

```
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        count = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == '1':
                    self.dfs(grid, i, j)
                    count += 1
        return count

    def dfs(self, grid, i, j):
        if grid[i][j] == '0': return
        grid[i][j] = '0'
        if i > 0:
            self.dfs(grid, i-1, j)
        if i < len(grid) - 1:
            self.dfs(grid, i+1, j)
        if j > 0:
            self.dfs(grid, i, j-1)
        if j < len(grid[0]) - 1:
            self.dfs(grid, i, j+1)
```

128. Longest Consecutive Sequence

java:

```java
class Solution {
    HashMap<Integer,Integer> parents;
    HashMap<Integer,Integer> counts;
    public int longestConsecutive(int[] nums) {
        parents = new HashMap<>();
        counts = new HashMap<>();
        for (int num : nums) counts.put(num, 1);
        for (int num : nums) union(num, num+1);
        int res = 0;
        for (int num : counts.values())
            res = Math.max(res, num);
        return res;
    }

    public int find(int x){
        if (x != parents.getOrDefault(x, x))
            parents.put(x, find(parents.get(x)));
        return parents.getOrDefault(x, x);
    }

    public void union(int x, int y){
        int x_parent = find(x), y_parent = find(y);
        int x_size = counts.getOrDefault(x_parent, 0);
        int y_size = counts.getOrDefault(y_parent, 0);
        if (x_parent != y_parent){
            if (x_size < y_size){
                parents.put(x_parent,y_parent);
                counts.put(y_parent, x_size + y_size);
            }else{
                parents.put(y_parent,x_parent);
                counts.put(x_parent, x_size + y_size);
            }
        }
    }
}
```

python3:

```python
class Solution:
    def __init__(self):
        self.parents = {}
        self.count = {}

    def longestConsecutive(self, nums: List[int]) -> int:
        for num in nums:
            self.count[num] = 1
        for num in nums:
```

```
                self.union(num, num+1)
            if nums:
                return max(self.count.values())
            else:
                return 0

        def find(self, x: int) -> int:
            if x != self.parents.get(x, x):
                self.parents[x] = self.find(self.parents.get(x))
            return self.parents.get(x, x)

        def union(self, a: int, b: int):
            a_parent, b_parent = self.find(a), self.find(b)
            a_size, b_size = self.count.get(a_parent, 0), self.count.get(b_parent, 0)

            if a_parent != b_parent:
                if a_size < b_size:
                    self.parents[a_parent] = b_parent
                    self.count[b_parent] += a_size
                else:
                    self.parents[b_parent] = a_parent
                    self.count[a_parent] += b_size
```

Alien Dictionary

(Leetcode Premium)

Graph Valid Tree

(Leetcode Premium)

Number of Connected Components in an Undirected Graph

(Leetcode Premium)

Interval

57. Insert Interval

java:

```
class Solution {
    public int[][] insert(int[][] intervals, int[] newInterval) {
        List<int[]> array = new ArrayList<>();
        int i = 0;
        while (i < intervals.length && intervals[i][1] < newInterval[0]){
            array.add(intervals[i]);
            i ++;
        }
        while (i < intervals.length && intervals[i][0] <= newInterval[1]){
            newInterval[0] = Math.min(intervals[i][0], newInterval[0]);
```

```
            newInterval[1] = Math.max(intervals[i][1], newInterval[1]);
            i ++;
        }
        array.add(newInterval);
        while (i < intervals.length){
            array.add(intervals[i]);
            i ++;
        }
        return array.toArray(new int [array.size()][]);
    }
}
```

python3:

```
class Solution:
    def insert(self, intervals: List[List[int]], newInterval: List[int]) ->
List[List[int]]:
        result = []
        i = 0
        while i < len(intervals) and intervals[i][1] < newInterval[0]:
            result.append(intervals[i])
            i += 1
        while i < len(intervals) and intervals[i][0] <= newInterval[1]:
            newInterval[0] = min(intervals[i][0], newInterval[0])
            newInterval[1] = max(intervals[i][1], newInterval[1])
            i += 1
        result.append(newInterval)

        while i < len(intervals):
            result.append(intervals[i])
            i += 1
        return result
```

56. Merge Intervals

java:

```
class Solution {
    public int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, new CustomSort());
        List<int[]> result = new ArrayList<>();
        result.add(intervals[0]);
        int[] prev = intervals[0];
        for (int i = 1; i < intervals.length; i++){
            if (intervals[i][0] <= prev[1]){
                prev[1] = Math.max(intervals[i][1], prev[1]);
                result.remove(result.size()-1);
                result.add(prev);
            }else{
                prev = intervals[i];
                result.add(intervals[i]);
```

```
            }
        }
        return result.toArray(new int [result.size()][]);
    }
}
class CustomSort implements Comparator<int[]>{
    public int compare(int[] array1, int[] array2)
    {
        return array1[0] - array2[0];
    }
}
```

python3:

```
class Solution:
    def merge(self, intervals: List[List[int]]) -> List[List[int]]:
        intervals.sort(key=lambda x: x[0])
        result = []
        result.append(intervals[0])
        prev = intervals[0]
        for i in range(1,len(intervals)):
            left, right=prev[0], prev[1]
            if right >= intervals[i][0]:
                result[-1] = [left, max(right, intervals[i][1])]
                prev = result[-1]
            else:
                prev = intervals[i]
                result.append(prev)
        return result
```

435. Non-overlapping Intervals

java:

```
class Solution {
    public int eraseOverlapIntervals(int[][] intervals) {
        Arrays.sort(intervals, new CustomSort());
        int res = 0;
        int prev = intervals[0][1];
        for (int i=1; i<intervals.length; i++){
            if (intervals[i][0] >= prev)
                prev = intervals[i][1];
            else
                res ++;
        }
        return res;
    }
}
class CustomSort implements Comparator<int[]>{
    public int compare(int[] array1, int[] array2)
    {
```

```
        return array1[1] - array2[1];
    }
}
```

python3:

```
class Solution:
    def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:
        intervals.sort(key=lambda x: x[1])
        count = 0
        prev = intervals[0][1]
        for i in range(1, len(intervals)):
            if intervals[i][0] >= prev:
                prev = intervals[i][1]
            else:
                count += 1
        return count
```

252. Meeting Rooms

(Leetcode Premium)

253. Meeting Rooms II

(Leetcode Premium)

Linked List

206. Reverse Linked List

java:

```
class Solution {
    public ListNode reverseList(ListNode head) {
        if ((head == null) || (head.next == null)) return head;
        ListNode prevnode = null, currnode = head, nextnode = head.next;
        while (nextnode!= null){
            currnode.next = prevnode;
            prevnode = currnode;
            currnode = nextnode;
            nextnode = nextnode.next;
        }
        currnode.next = prevnode;
        return currnode;
    }
}
```

python3:

```python
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if head is None or head.next == None:
            return head
        prev, curr, next = None, head, head.next
        while next:
            curr.next = prev
            prev, curr, next = curr, next, next.next
        curr.next = prev
        return curr
```

141. Linked List Cycle

java:

```java
class Solution {
    public boolean hasCycle(ListNode head) {
        if (head == null || head.next == null || head.next.next == null)
            return false;
        ListNode slow = head.next, fast = head.next.next;
        while (slow != fast){
            if (fast.next == null || fast.next.next == null)
                return false;
            slow = slow.next;
            fast = fast.next.next;
        }
        return true;
    }
}
```

python3:

```python
class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        if head == None or head.next == None or head.next.next == None:
            return False
        slow, fast = head.next, head.next.next
        while slow != fast:
            if fast.next is None or fast.next.next is None:
                return False
            slow, fast = slow.next, fast.next.next
        return True
```

21. Merge Two Sorted Lists

java:

```java
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
```

```
        ListNode dummyhead = new ListNode();
        ListNode node = dummyhead;
        while (list1!=null && list2!=null){
            if (list1.val < list2.val){
                node.next = list1;
                list1 = list1.next;
            }else{
                node.next = list2;
                list2 = list2.next;
            }
            node = node.next;
        }
        if (list1==null) node.next = list2;
        else node.next = list1;
        return dummyhead.next;
    }
}
```

python3:

```
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) ->
Optional[ListNode]:
        dummyhead = node = ListNode()
        while list1 and list2:
            if list1.val < list2.val:
                node.next = list1
                list1 = list1.next
            else:
                node.next = list2
                list2 = list2.next
            node = node.next
        node.next = list1 or list2
        return dummyhead.next
```

23. Merge k Sorted Lists

java:

```
class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        if (lists.length == 0) return null;
        PriorityQueue<Pair> pq = new PriorityQueue<>();
        ListNode dummynode = new ListNode();
        ListNode head = dummynode;
        for (int i = 0; i<lists.length; i++){
            if (lists[i] != null){
                pq.add(new Pair(lists[i].val, i));
            }
        }
        while (! pq.isEmpty()){
```

```
            Pair small_pair = pq.poll();

            int small_value = small_pair.a;
            int small_index = small_pair.b;

            lists[small_index] = lists[small_index].next;
            dummynode.next = new ListNode(small_value);
            dummynode = dummynode.next;

            if (lists[small_index] != null){
                pq.add(new Pair(lists[small_index].val, small_index));
            }
        }
        return head.next;
    }
}
class Pair implements Comparable<Pair> {
    int a, b;
    public Pair(int a, int b) {
        this.a = a;
        this.b = b;
    }
    public int compareTo(Pair pair) {
        return a - pair.a;
    }
}
```

python3:

```
class Solution:
    def mergeKLists(self, lists: List[Optional[ListNode]]) -> Optional[ListNode]:
        heap = []
        dummynode = head = ListNode()
        for i, lst in enumerate(lists):
            if lst:
                heapq.heappush(heap, (lst.val, i))
        while heap:
            # find the small index lst
            small_index = heapq.heappop(heap)[1]
            smallnode = lists[small_index]
            # update that index lst without head
            lists[small_index] = smallnode.next
            dummynode.next = smallnode
            dummynode = dummynode.next
            if smallnode.next:
                heapq.heappush(heap, (smallnode.next.val, small_index))
        return head.next
```

19. Remove Nth Node From End of List

java:

```java
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode dummy = new ListNode(0, head);
        ListNode slow = dummy, fast = dummy;
        while (fast.next!=null && n>0){
            fast = fast.next;
            n --;
        }
        while (fast.next!=null) {
            fast = fast.next;
            slow = slow.next;
        }
        slow.next = slow.next.next;
        return dummy.next;
    }
}
```

python3:

```python
class Solution:
    def removeNthFromEnd(self, head: Optional[ListNode], n: int) ->
Optional[ListNode]:
        dummynode = ListNode(next = head)
        slow = fast = dummynode
        for i in range(n):
            fast = fast.next
        while fast.next is not None:
            slow = slow.next
            fast = fast.next
        slow.next = slow.next.next
        return dummynode.next
```

143. Reorder List

java:

```java
class Solution {
    public void reorderList(ListNode head) {
        ListNode left = new ListNode(0, head);
        ListNode right = new ListNode(0, head);
        while (right != null && right.next != null){
            left = left.next;
            right = right.next.next;
        }
        ListNode prev = null, curr = left.next, next;
        left.next = null;
```

```
        while (curr != null) {
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        ListNode curr_forw = head, curr_back = prev;
        while (curr_forw!=null && curr_back!=null){
            ListNode next_forw = curr_forw.next;
            ListNode next_back = curr_back.next;
            curr_forw.next = curr_back;
            curr_back.next = next_forw;
            curr_forw = next_forw;
            curr_back = next_back;
        }



    }
}
```

python3:

```
class Solution:
    def reorderList(self, head: Optional[ListNode]) -> None:
        left = right = ListNode(next = head)
        while right and right.next:
            left = left.next
            right = right.next.next
        prev, curr, next = None, left.next, None
        left.next = None
        while curr:
            next = curr.next
            curr.next = prev
            prev = curr
            curr = next
        curr_forw, curr_back = head, prev
        while curr_forw and curr_back:
            next_forw, next_back = curr_forw.next, curr_back.next
            curr_forw.next, curr_back.next = curr_back, next_forw
            curr_forw, curr_back = next_forw, next_back
```

Matrix

73. Set Matrix Zeroes

java:

```
class Solution {
    public void setZeroes(int[][] matrix) {
```

```
        int[] rowset = new int[matrix.length];
        int[] colset = new int[matrix[0].length];
        for (int i = 0; i < matrix.length; i++){
            for (int j = 0;j < matrix[0].length; j++){
                if (rowset[i]==1 && colset[j]==1)
                    continue;
                if (matrix[i][j] == 0){
                    rowset[i]=1;
                    colset[j]=1;
                }
            }
        }
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                if (rowset[i]==1 || colset[j]==1)
                    matrix[i][j] = 0;
            }
        }
    }
}
```

python3:

```
class Solution:
    def setZeroes(self, matrix: List[List[int]]) -> None:
        m, n = len(matrix), len(matrix[0])
        rows, cols = set(), set()
        for i in range(m):
            for j in range(n):
                if i in rows and j in cols:
                    continue
                if matrix[i][j] == 0:
                    rows.add(i)
                    cols.add(j)
        for i in range(m):
            for j in range(n):
                if i in rows or j in cols:
                    matrix[i][j] = 0
```

54. Spiral Matrix

java:

```
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        int row = matrix.length;
        int col = matrix[0].length;
        List<Integer> lst = new ArrayList<Integer>();
        boolean[][] seen = new boolean[row][col];
        int x=0, y=0, di=0, dj=1;
        while (lst.size() < row*col){
```

```
            seen[x][y] = true;
            lst.add(matrix[x][y]);
            int nextx = x+di, nexty = y+dj;
            if (!(0<=nextx && nextx<row && 0<=nexty && nexty<col && (!seen[nextx]
[nexty]) )){
                int temp = di;
                di = dj;
                dj = -temp;
            }
            x = x+di;
            y = y+dj;
        }
        return lst;
    }
}
```

python3:

```
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        row, col = len(matrix), len(matrix[0])
        lst = []
        seen = [[False for _ in range(col)] for _ in range(row)]
        x, y, di, dj = 0, 0, 0, 1
        while len(lst) < row*col:
            seen[x][y] = True
            lst.append(matrix[x][y])
            nextx, nexty = x+di, y+dj
            if not (0<= nextx <row and 0<= nexty <col and (not seen[nextx][nexty])):
                di, dj = dj, -di
            x, y = x+di, y+dj
        return lst
```

48. Rotate Image

java:

```
class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        int[][] temp = new int[n][n];
        for (int i = 0; i < n; i ++){
            for (int j = 0; j < n; j ++) {
                temp[j][n-1-i] = matrix[i][j];
            }
        }
        for (int i = 0; i < n; i ++){
            for (int j = 0; j < n; j ++) {
                matrix[i][j] = temp[i][j];
            }
        }
```

```
    }
}
```

python3:

```python
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        n = len(matrix)
        res = [[0 for _ in range(n)] for _ in range(n)]
        for i in range(n):
            for j in range(n):
                res[j][n-i-1] = matrix[i][j]
        for i in range(n):
            for j in range(n):
                matrix[i][j] = res[i][j]
```

79. Word Search

java:

```java
class Solution {
    public boolean exist(char[][] board, String word) {
        int m = board.length, n = board[0].length;
        char[] wordarray = word.toCharArray();
        for (int row = 0; row < m; row ++){
            for (int col = 0; col < n; col ++){
                boolean[][] visited = new boolean[m][n];
                if (board[row][col] == wordarray[0]){
                    if (dfs(board, row, col, visited, wordarray, 0))
                        return true;
                }
            }
        }
        return false;
    }
    public boolean dfs(char[][] board, int row, int col, boolean[][] visited, char[]
word, int idx){
        if (word[idx] != board[row][col] || visited[row][col] == true || idx >=
word.length)
            return false;
        if (word[idx] == board[row][col] && idx == word.length-1)
            return true;
        visited[row][col] = true;
        if (row > 0 && dfs(board, row-1, col, visited, word, idx + 1))
            return true;
        if (row < board.length-1 && dfs(board, row+1, col, visited, word, idx + 1))
            return true;
        if (col > 0 && dfs(board, row, col-1, visited, word, idx + 1))
            return true;
        if (col < board[0].length-1 && dfs(board, row, col+1, visited, word, idx +
1))
```

```
            return true;
        visited[row][col] = false;
        return false;
    }
}
```

python3:

```
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        m,n = len(board),len(board[0])
        for row in range(m):
            for col in range(n):
                if board[row][col] == word[0]:
                    if self.dfs(board, row, col, set(), word, 0):
                        return True
        return False

    def dfs(self, board, row, col, visited, word, idx):
        if board[row][col] != word[idx] or (row, col) in visited or idx >=
len(word):
            return False
        if board[row][col] == word[idx] and idx == len(word) - 1:
            return True
        visited.add((row, col))
        if row > 0 and self.dfs(board, row-1, col, visited, word, idx + 1):
            return True
        if row < len(board) - 1 and self.dfs(board, row+1, col, visited, word, idx +
1):
            return True
        if col > 0 and self.dfs(board, row, col-1, visited, word, idx + 1):
            return True
        if col < len(board[0]) - 1 and self.dfs(board, row, col+1, visited, word,
idx + 1):
            return True
        visited.remove((row, col))
        return False
```

String

3. Longest Substring Without Repeating Characters

java:

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        char[] charArray = s.toCharArray();
        int[] window = new int[100];
        int left = 0;
        int res = 0;
```

```
        for (int right = 0; right < charArray.length; right ++){
            int index = charArray[right] - ' ';
            if (window[index] == 0) {
                window[index] ++;
                res = Math.max(right - left + 1, res);
            }else {
                while (charArray[left] != charArray[right]) {
                    window[charArray[left] - ' '] --;
                    left ++;
                }
                window[index] ++;
                window[charArray[left] - ' '] --;
                left ++;
            }
        }
        return res;
    }
}
```

python3:

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        window = set()
        maxlength = 0
        left = 0
        for right, char in enumerate(s):
            if char not in window:
                window.add(char)
                maxlength = max(maxlength, right-left+1)
            else:
                while s[left] != char:
                    window.remove(s[left])
                    left += 1
                window.remove(s[left])
                left += 1
                window.add(char)
        return maxlength
```

424. Longest Repeating Character Replacement

java:

```
class Solution {
    public int characterReplacement(String s, int k) {
        int[] arr = new int[26];
        char[] charArray = s.toCharArray();
        int maxlen = 0, largestCount = 0;
        for(int i = 0; i < charArray.length; i ++){
            arr[charArray[i] - 'A'] ++;
            largestCount = Math.max(largestCount, arr[charArray[i] - 'A']);
```

```
            if (maxlen - largestCount >= k)
                arr[charArray[i - maxlen] - 'A'] --;
            else
                maxlen ++;
        }
        return maxlen;
    }
}
```

python3:

```
class Solution:
    def characterReplacement(self, s: str, k: int) -> int:
        maxlen, largestCount = 0, 0
        arr = collections.Counter()
        for i in range(len(s)):
            arr[s[i]] += 1
            largestCount = max(largestCount, arr[s[i]])
            if maxlen - largestCount >= k:
                arr[s[i - maxlen]] -= 1
            else:
                maxlen += 1
        return maxlen
```

76. Minimum Window Substring

java:

```
class Solution {
    public String minWindow(String s, String t) {
        if (s==null || t==null || s.length() < t.length())
            return "";
        if (s.equals(t)) return s;
        int[] window = new int[58];
        char[] sArray = s.toCharArray();
        char[] tArray = t.toCharArray();
        for (char c : tArray)
            window[c-'A'] += 1;
        int minlength = sArray.length+1, missChar = tArray.length;
        int left = 0, right = 0, minleft = 0;

        while (right < sArray.length){
            if (window[sArray[right]-'A'] > 0)
                missChar --;
            window[sArray[right]-'A'] --;
            right ++;
            while (missChar == 0){
                if (right-left < minlength){
                    minlength = right - left;
                    minleft = left;
                }
```

```
                window[sArray[left]-'A'] ++;
                if (window[sArray[left]-'A'] > 0)
                    missChar ++;
                left ++;
            }
        }
        if (minlength != sArray.length+1)
            return s.substring(minleft, minleft+minlength);
        return "";
    }
}
```

python3:

```
class Solution:
    def minWindow(self, s: str, t: str) -> str:
        if not s or not t or len(s)<len(t):
            return ""
        if s == t: return s
        dic_t = {}
        for char in s: dic_t[char] = 0
        for char in t:
            if char in dic_t:
                dic_t[char] += 1
            else: return ""
        counter = len(t)
        left,right, minleft, minlength = 0, 0, 0, float('inf')

        while right < len(s):
            if dic_t[s[right]] > 0:
                counter -= 1
            dic_t[s[right]] -= 1
            right+=1
            while counter == 0:
                if right-left < minlength:
                    minlength = right - left
                    minleft = left
                dic_t[s[left]] += 1
                if dic_t[s[left]] > 0:
                    counter += 1
                left += 1

        if minlength != float('inf'):
            return s[minleft:minleft+minlength]
        return ""
```

242. Valid Anagram

java:

```java
class Solution {
    public boolean isAnagram(String s, String t) {
        if (s.length() != t.length())
            return false;
        int[] dic = new int[26];
        for (char c : s.toCharArray())
            dic[c-'a'] ++;
        for (char c : t.toCharArray()){
            dic[c-'a'] --;
            if (dic[c-'a'] < 0) return false;
        }
        for (int i = 0; i < 26; i++) {
            if (dic[i] != 0)
                return false;
        }
        return true;
    }
}
```

python3:

```python
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        if len(s) != len(t):
            return False
        return Counter(s) == Counter(t)
```

49. Group Anagrams

java:

```java
class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        Map<String, ArrayList<String>> map = new HashMap<>();
        for (String str : strs){
            char[] ca = str.toCharArray();
            Arrays.sort(ca);
            String keystr = String.valueOf(ca);
            if (! map.containsKey(keystr))
                map.put(keystr, new ArrayList<String>());
            map.get(keystr).add(str);
        }
        return new ArrayList<>(map.values());
    }
}
```

python3:

```python
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        strs_table = {}
        for string in strs:
            sorted_string = ''.join(sorted(string))
            if sorted_string not in strs_table:
                strs_table[sorted_string] = []
            strs_table[sorted_string].append(string)
        return list(strs_table.values())
```

20. Valid Parentheses

java:

```java
class Solution {
    public boolean isValid(String s) {
        HashMap<Character, Character> map = new HashMap<Character, Character>();
        map.put(')','(');
        map.put('}','{');
        map.put(']','[');
        Stack<Character> stack = new Stack<Character>();
        for (char c : s.toCharArray()){
            if (! map.containsKey(c))
                stack.push(c);
            else if (stack.size() == 0 || map.get(c)!= stack.pop())
                return false;
        }
        if (stack.size() == 0)
            return true;
        return false;
    }
}
```

python3:

```python
class Solution:
    def isValid(self, s: str) -> bool:
        stack = []
        parenthesis = {')':'(',']':'[','}':'{'}
        for char in s:
            if char not in parenthesis:
                stack.append(char)
            elif len(stack) ==0 or stack.pop() != parenthesis[char]:
                return False
        if len(stack) == 0:
            return True
        return False
```

125. Valid Palindrome

java:

```java
class Solution {
    public boolean isPalindrome(String s) {
        char[] charArray = s.toLowerCase().toCharArray();
        int i = 0, j = charArray.length-1;
        while (i < j){
            if (!Character.isLetterOrDigit(charArray[i]))
                i ++;
            else if (!Character.isLetterOrDigit(charArray[j]))
                j --;
            else {
                if (charArray[i] != charArray[j])
                    return false;
                i ++;
                j --;
            }
        }
        return true;
    }
}
```

python3:

```python
class Solution:
    def isPalindrome(self, s: str) -> bool:
        s = ''.join(filter(str.isalnum, str(s).lower()))
        for i in range(len(s)//2):
            if s[i] != s[len(s)-i-1]:
                return False
        return True
```

5. Longest Palindromic Substring

java:

```java
class Solution {
    public String longestPalindrome(String s) {
        int[][] opt = new int[s.length()][s.length()];
        char[] sArray = s.toCharArray();
        int maxlength = 0;
        int left=0, right=0;
        for (int i=0; i<s.length(); i++){
            opt[i][i] = 1;
        }
        for (int i = sArray.length-1; i > -1; i--){
            for (int j = i+1; j < sArray.length; j++){
                if (sArray[i] == sArray[j]){
```

```
                    if (opt[i+1][j-1]>0 || j-i<2)
                        opt[i][j] = opt[i+1][j-1] + 2;
                    if (opt[i][j] > maxlength){
                        maxlength = opt[i][j];
                        left = i;
                        right = j;
                    }
                }
            }
        }
        return s.substring(left, right+1);
    }
}
```

python3:

```
class Solution:
    def longestPalindrome(self, s: str) -> str:
        opt = [[0 for _ in range(len(s))] for _ in range(len(s))]
        maxlenth = 0
        left, right = 0, 0
        for i in range(0, len(s)):
            opt[i][i] = 1
        for i in range(len(s)-1, -1, -1):
            for j in range(i+1, len(s)):
                if s[i] == s[j]:
                    if opt[i+1][j-1]>0 or j-i<2:
                        opt[i][j] = opt[i+1][j-1] + 2
                    if opt[i][j] > maxlenth:
                        maxlenth = j - i + 1
                        left, right = i, j
        return s[left:right + 1]
```

647. Palindromic Substrings

java:

```
class Solution {
    public int countSubstrings(String s) {
        boolean[][] opt = new boolean[s.length()][s.length()];
        char[] sArray = s.toCharArray();
        int result = 0;
        for (int i = sArray.length-1; i > -1; i--){
            for (int j = i; j < sArray.length; j++){
                if (sArray[i] == sArray[j]){
                    if (j-i <= 1){
                        result ++;
                        opt[i][j] = true;
                    }else if (opt[i+1][j-1]){
                        result ++;
                        opt[i][j] = true;
```

```
                    }
                }
            }
        }
        return result;
    }
}
```

python3:

```python
class Solution:
    def countSubstrings(self, s: str) -> int:
        opt = [[0 for _ in range(len(s))] for _ in range(len(s))]
        result = 0
        for i in range(len(s)-1, -1, -1):
            for j in range(i, len(s)):
                if s[i] == s[j]:
                    if j-i < 2 or opt[i+1][j-1]:
                        opt[i][j] = 1
                        result += 1
        return result
```

Encode and Decode Strings

(Leetcode Premium)

Tree

100. Same Tree

java:

```java
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if (p==null && q==null)
            return true;
        else if (p==null || q==null)
            return false;
        else if (p.val == q.val)
            return isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
        return false;
    }
}
```

python3:

```python
class Solution:
    def isSameTree(self, p: Optional[TreeNode], q: Optional[TreeNode]) -> bool:
        if not p and not q:
            return True
        if not p or not q:
            return False
        if p.val == q.val:
            return self.isSameTree(p.left, q.left) and self.isSameTree(p.right,
q.right)
        return False
```

102. Binary Tree Level Order Traversal

java:

```java
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> results = new ArrayList<>();
        if (root==null) return results;
        Deque<TreeNode> deque = new ArrayDeque<>();
        deque.add(root);
        while (! deque.isEmpty()){
            int size = deque.size();
            List<Integer> result = new ArrayList<>();
            for (int i=0; i<size; i++){
                TreeNode node = deque.poll();
                result.add(node.val);
                if (node.left!=null) deque.add(node.left);
                if (node.right!=null) deque.add(node.right);
            }
            results.add(result);
        }
        return results;
    }
}
```

python3:

```python
class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        results = []
        if not root: return results
        queue = deque()
        queue.append(root)
        while queue:
            size = len(queue)
            result = []
            for _ in range(size):
                node = queue.popleft()
                result.append(node.val)
```

```
                if node.left: queue.append(node.left)
                if node.right: queue.append(node.right)
            results.append(result)
        return results
```

104. Maximum Depth of Binary Tree

java:

```
class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) return 0;
        return Math.max(maxDepth(root.left), maxDepth(root.right)) + 1;
    }
}
```

python3:

```
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0
        return max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1
```

105. Construct Binary Tree from Preorder and Inorder Traversal

java:

```
class Solution {
    Map<Integer, Integer> map;

    public TreeNode buildTree(int[] preorder, int[] inorder) {
        map = new HashMap<>();
        for (int i = 0; i < inorder.length; i++)
            map.put(inorder[i], i);
        return helper(preorder, 0, preorder.length-1, inorder, 0, inorder.length-1);
    }

    public TreeNode helper(int[] preorder, int preleft, int preright, int[] inorder,
int inleft, int inright){
        if (preleft>preright || inleft>inright) return null;
        int root_val = preorder[preleft];
        int pos = map.get(root_val);
        TreeNode root = new TreeNode(root_val);
        root.left = helper(preorder, preleft+1, preleft+pos-inleft, inorder, inleft,
pos-1);
        root.right = helper(preorder, preleft+pos-inleft+1, preright, inorder,
pos+1, inright);
        return root;
    }
```

```
}
```

python3:

```
class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) ->
Optional[TreeNode]:
        if not preorder:
            return None
        rootValue = preorder[0]
        root = TreeNode(rootValue)
        rootPos = inorder.index(rootValue)
        root.left = self.buildTree(preorder[1:rootPos+1], inorder[:rootPos])
        root.right = self.buildTree(preorder[rootPos+1:], inorder[rootPos+1:])
        return root
```

226. Invert Binary Tree

java:

```
class Solution {
    public TreeNode invertTree(TreeNode root) {
        if (root==null) return root;
        TreeNode parent = new TreeNode(root.val);
        parent.left = invertTree(root.right);
        parent.right = invertTree(root.left);
        return parent;
    }
}
```

python3:

```
class Solution:
    def invertTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
        if not root: return None
        parent = TreeNode(root.val)
        parent.left, parent.right = self.invertTree(root.right),
self.invertTree(root.left)
        return parent
```

124. Binary Tree Maximum Path Sum

java:

```
class Solution {
    int res;
    public int maxPathSum(TreeNode root) {
        res = root.val;
        traversal(root);
```

```
        return res;
    }

    public int traversal(TreeNode root) {
        if (root == null) return 0;
        int left = traversal(root.left);
        int right = traversal(root.right);
        res = Math.max(res, left + right + root.val);
        return Math.max(root.val + Math.max(left, right), 0);
    }
}
```

python3:

```
class Solution:
    def maxPathSum(self, root: Optional[TreeNode]) -> int:
        self.ans = root.val
        self.maxend(root)
        return self.ans

    def maxend(self, node):
        if not node: return 0
        left = self.maxend(node.left)
        right = self.maxend(node.right)
        self.ans = max(self.ans, left+right+node.val)
        return max(node.val + max(left, right), 0)
```

297. Serialize and Deserialize Binary Tree

java recursion:

```
public class Codec {
    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        StringBuilder result = new StringBuilder();
        buildString(root, result);
        return result.toString();
    }

    private void buildString(TreeNode node, StringBuilder sb) {
        if (node == null) {
            sb.append("null").append(",");
        } else {
            sb.append(node.val).append(",");
            buildString(node.left, sb);
            buildString(node.right,sb);
        }
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
```

```
        Queue<String> queue = new LinkedList<>();
        queue.addAll(Arrays.asList(data.split(",")));
        return buildTree(queue);
    }

    private TreeNode buildTree(Queue<String> queue) {
        String val = queue.poll();
        if (val.equals("null")) return null;
        else {
            TreeNode node = new TreeNode(Integer.parseInt(val));
            node.left = buildTree(queue);
            node.right = buildTree(queue);
            return node;
        }
    }
}
```

java iteratively:

```
public class Codec {
    static Map<String, TreeNode> map = new HashMap<>();

    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        if (root == null) return "";
        StringBuilder result = new StringBuilder();
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()){
            TreeNode node = queue.poll();
            if (node==null){
                result.append("null"+", ");
                continue;
            }
            result.append(node.val+", ");
            queue.add(node.left);
            queue.add(node.right);
        }
        map.put(result.toString(), root);
        return result.toString();
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        if (data == "") return null;
        String[] values = data.split(", ");
        TreeNode root = new TreeNode(Integer.parseInt(values[0]));
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        int i = 0;
        while  (!queue.isEmpty()){
```

```
            TreeNode node = queue.poll();
            if ((i+1<values.length) && (!values[i+1].equals("null"))){
                node.left = new TreeNode(Integer.parseInt(values[i+1]));
                queue.add(node.left);
            }
            if ((i+2<values.length) && (!values[i+2].equals("null"))){
                node.right = new TreeNode(Integer.parseInt(values[i+2]));
                queue.add(node.right);
            }
            i += 2;
        }
        return root;
    }
}
```

python3:

```
class Codec:
    def serialize(self, root):
        """Encodes a tree to a single string."""
        result = []
        self.buildString(root, result)
        return "".join(result)

    def buildString(self, root, result):
        if not root:
            result.append("#")
            result.append(",")
        else:
            result.append(str(root.val))
            result.append(",")
            self.buildString(root.left, result)
            self.buildString(root.right, result)

    def deserialize(self, data):
        """Decodes your encoded data to tree."""
        queue = deque()
        for str in data.split(","):
            queue.append(str)
        return self.buildTree(queue)

    def buildTree(self, queue):
        val = queue.popleft()
        if val == "#": return None
        else:
            node = TreeNode(int(val))
            node.left = self.buildTree(queue)
            node.right = self.buildTree(queue)
            return node
```

572. Subtree of Another Tree

java:

```
class Solution {
    public boolean isSubtree(TreeNode root, TreeNode subRoot) {
        if (root != null && subRoot != null){
            if (root.val==subRoot.val){
                if (sameTree(root, subRoot)) return true;
            }
            return (isSubtree(root.left, subRoot) || isSubtree(root.right,
subRoot));
        }
        return false;
    }

    public boolean sameTree(TreeNode t1, TreeNode t2){
        if (t1 == null && t2 == null)
            return true;
        if (t1 == null || t2 == null || t1.val!=t2.val)
            return false;
        if (! sameTree(t1.left, t2.left))
            return false;
        if (! sameTree(t1.right, t2.right))
            return false;
        return true;
    }
}
```

python3:

```
class Solution:
    def isSubtree(self, root: Optional[TreeNode], subRoot: Optional[TreeNode]) ->
bool:
        if root is not None:
            if root.val == subRoot.val:
                if self.sameTree(root, subRoot):
                    return True
            return self.isSubtree(root.left, subRoot) or self.isSubtree(root.right,
subRoot)
        return False

    def sameTree(self, t1, t2):
        if not t1 and not t2:
            return True
        if not t1 or not t2 or t1.val != t2.val:
            return False
        return self.sameTree(t1.left, t2.left) and self.sameTree(t1.right, t2.right)
```

98. Validate Binary Search Tree

java:

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        return trversal(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }
    public boolean trversal(TreeNode root, long min, long max){
        if (root == null) return true;
        if (root.val <= min || root.val >= max)
            return false;
        return trversal(root.left,min,root.val) &&
trversal(root.right,root.val,max);
    }
}
```

python3:

```
class Solution:
    def isValidBST(self, root: Optional[TreeNode]) -> bool:
        return self.isValid(root, float('-inf'), float('inf'))

    def isValid(self, root, minVal, maxVal):
        if not root: return True
        if root.val <= minVal or root.val >= maxVal:
            return False
        return self.isValid(root.left, minVal, root.val) and
self.isValid(root.right, root.val, maxVal)
```

230. Kth Smallest Element in a BST

java:

```
class Solution {
    public int kthSmallest(TreeNode root, int k) {
        Stack<TreeNode> stack = new Stack<>();
        TreeNode curr = root;
        while (curr != null || ! stack.isEmpty()){
            if (curr != null){
                stack.push(curr);
                curr = curr.left;
            } else {
                curr = stack.pop();
                k --;
                if (k == 0) return curr.val;
                curr = curr.right;
            }
        }
        return curr.val;
```

```
    }
}
```

python3:

```
class Solution:
    def kthSmallest(self, root: Optional[TreeNode], k: int) -> int:
        stack = []
        while root or stack:
            if root:
                stack.append(root)
                root = root.left
            else:
                root = stack.pop()
                k -= 1
                if k == 0: return root.val
                root = root.right
```

235. Lowest Common Ancestor of a Binary Search Tree

java:

```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if (root == null) return null;
        if (root == p || root == q) return root;
        if (root.val > p.val && root.val > q.val)
            return lowestCommonAncestor(root.left, p, q);
        else if (root.val < p.val && root.val < q.val)
            return lowestCommonAncestor(root.right, p, q);
        else return root;
    }
}
```

python3:

```
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -
> 'TreeNode':
        if root==None or root==p or root==q:
            return root
        elif p.val < root.val and q.val < root.val:
            return self.lowestCommonAncestor(root.left, p, q)
        elif p.val > root.val and q.val > root.val:
            return self.lowestCommonAncestor(root.right, p, q)
        else:
            return root
```

208. Implement Trie (Prefix Tree)

java:

```java
class TrieNode {
    boolean isWord;
    TrieNode[] children;
    public TrieNode() {
        isWord = false;
        children = new TrieNode[26];
    }
}
class Trie {
    TrieNode root;
    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            int index = c - 'a';
            if (node.children[index] == null) {
                node.children[index] = new TrieNode();
            }
            node = node.children[index];
        }
        node.isWord = true;
    }

    public boolean search(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            int index = c - 'a';
            if (node.children[index] == null) {
                return false;
            }
            node = node.children[index];
        }
        return node.isWord;
    }

    public boolean startsWith(String prefix) {
        TrieNode node = root;
        for (char c : prefix.toCharArray()) {
            int index = c - 'a';
            if (node.children[index] == null) {
                return false;
            }
            node = node.children[index];
        }
```

```
        return true;
    }
}
```

python3:

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_word = False
class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, word: str) -> None:
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_word = True

    def search(self, word: str) -> bool:
        node = self.root
        for char in word:
            if char not in node.children:
                return False
            node = node.children[char]
        return node.is_word

    def startsWith(self, prefix: str) -> bool:
        node = self.root
        for char in prefix:
            if char not in node.children:
                return False
            node = node.children[char]
        return True
```

211. Design Add and Search Words Data Structure

java:

```
class WordNode {
    boolean isWord;
    WordNode[] children;
    public WordNode() {
        isWord = false;
        children = new WordNode[26];
    }
}
class WordDictionary {
```

```
    WordNode root;
    public WordDictionary() {
        root = new WordNode();
    }

    public void addWord(String word) {
        WordNode node = root;
        for (char c : word.toCharArray()) {
            int index = c - 'a';
            if (node.children[index] == null) {
                node.children[index] = new WordNode();
            }
            node = node.children[index];
        }
        node.isWord = true;
    }

    public boolean search(String word) {
        WordNode node = root;
        return partialSearch(node, word.toCharArray(), 0);
    }

    public boolean partialSearch(WordNode node, char[] word, int start){
        if (start==word.length) return node.isWord;
        if (word[start] != '.'){
            int index = word[start] - 'a';
            if (node.children[index] != null){
                return partialSearch(node.children[index], word, start + 1);
            }
        } else {
            for (int i = 0; i < 26; i++) {
                if (node.children[i] != null) {
                    if (partialSearch(node.children[i], word, start + 1)) {
                        return true;
                    }
                }
            }
        }
        return false;
    }
}
```

python3:

```
class WordNode:
    def __init__(self):
        self.children = {}
        self.is_word = False

class WordDictionary:
    def __init__(self):
```

```
        self.root = WordNode()

    def addWord(self, word: str) -> None:
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = WordNode()
            node = node.children[char]
        node.is_word = True

    def search(self, word: str) -> bool:
        node = self.root
        return self.partialSearch(node, word, 0)

    def partialSearch(self, node: WordNode, word: str, start: int) -> bool:
        if start==len(word): return node.is_word
        char = word[start]
        if char != ".":
            if char in node.children:
                return self.partialSearch(node.children[char], word, start+1)
        else:
            for choice in node.children:
                if self.partialSearch(node.children[choice], word, start+1):
                    return True
        return False
```

212. Word Search II

java:

```
class TrieNode {
    TrieNode[] next = new TrieNode[26];
    String word;
}
class Solution {
    public List<String> findWords(char[][] board, String[] words) {
        List<String> res = new ArrayList<>();
        TrieNode root = buildTrie(words);
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                dfs(board, i, j, root, res);
            }
        }
        return res;
    }
    public TrieNode buildTrie(String[] words) {
        TrieNode root = new TrieNode();
        for (String word : words) {
            TrieNode node = root;
            for (char c : word.toCharArray()) {
                int i = c - 'a';
```

```
                if (node.next[i] == null)
                    node.next[i] = new TrieNode();
                node = node.next[i];
            }
            node.word = word;
        }
        return root;
    }
    public void dfs(char[][] board, int row, int col, TrieNode root, List<String>
res){
        char c = board[row][col];
        if (c == '#' || root.next[c - 'a'] == null)
            return;
        root = root.next[c - 'a'];
        if (root.word != null) {
            res.add(root.word);
            root.word = null;
        }
        board[row][col] = '#';
        if (row > 0)
            dfs(board, row-1, col, root, res);
        if (row < board.length-1)
            dfs(board, row+1, col, root, res);
        if (col > 0)
            dfs(board, row, col-1, root, res);
        if (col < board[0].length-1)
            dfs(board, row, col+1, root, res);
        board[row][col] = c;
    }
}
```

python3:

```
class TrieNode:
    def __init__(self):
        self.next = {}
        self.word = None
class Solution:
    def findWords(self, board: List[List[str]], words: List[str]) -> List[str]:
        self.res = []
        root = self.buildTrie(words)
        for row in range(len(board)):
            for col in range(len(board[0])):
                self.dfs(board, row, col, root)
        return self.res

    def buildTrie(self, words):
        root = TrieNode()
        for word in words:
            node = root
            for char in word:
```

```
                if char not in node.next:
                    node.next[char] = TrieNode()
                node = node.next[char]
            node.word = word
        return root

    def dfs(self, board, row, col, root):
        char = board[row][col]
        if char == "#" or char not in root.next:
            return
        root = root.next[char]
        if root.word:
            self.res.append(root.word)
            root.word = None
        board[row][col] = '#'
        if row > 0:
            self.dfs(board, row-1, col, root)
        if row < len(board) - 1:
            self.dfs(board, row+1, col, root)
        if col > 0:
            self.dfs(board, row, col-1, root)
        if col < len(board[0]) - 1:
            self.dfs(board, row, col+1, root)
        board[row][col] = char
```

Heap

23. Merge k Sorted Lists

java:

```
class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        if (lists.length == 0) return null;
        PriorityQueue<Pair> pq = new PriorityQueue<>();
        ListNode dummynode = new ListNode();
        ListNode head = dummynode;
        for (int i = 0; i<lists.length; i++){
            if (lists[i] != null){
                pq.add(new Pair(lists[i].val, i));
            }
        }
        while (! pq.isEmpty()){
            Pair small_pair = pq.poll();

            int small_value = small_pair.a;
            int small_index = small_pair.b;

            lists[small_index] = lists[small_index].next;
```

```
            dummynode.next = new ListNode(small_value);
            dummynode = dummynode.next;

            if (lists[small_index] != null){
                pq.add(new Pair(lists[small_index].val, small_index));
            }
        }
        return head.next;
    }
}
class Pair implements Comparable<Pair> {
    int a, b;
    public Pair(int a, int b) {
        this.a = a;
        this.b = b;
    }
    public int compareTo(Pair pair) {
        return a - pair.a;
    }
}
```

python3:

```
class Solution:
    def mergeKLists(self, lists: List[Optional[ListNode]]) -> Optional[ListNode]:
        heap = []
        dummynode = head = ListNode()
        for i, lst in enumerate(lists):
            if lst:
                heapq.heappush(heap, (lst.val, i))
        while heap:
            # find the small index lst
            small_index = heapq.heappop(heap)[1]
            smallnode = lists[small_index]
            # update that index lst without head
            lists[small_index] = smallnode.next
            dummynode.next = smallnode
            dummynode = dummynode.next
            if smallnode.next:
                heapq.heappush(heap, (smallnode.next.val, small_index))
        return head.next
```

347. Top K Frequent Elements

java:

```
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> counter = new HashMap<>();
        PriorityQueue<Pair> pq = new PriorityQueue<>();
        int[] result = new int[k];
```

```
        for (int num : nums)
            counter.put(num, counter.getOrDefault(num,0)+1);
        for (int num : counter.keySet())
            pq.add(new Pair(num, counter.get(num)));
        for (int i = 0; i < k; i++)
            result[i] = pq.poll().num;
        return result;
    }
}
class Pair implements Comparable<Pair> {
    int num, times;
    public Pair(int num, int times) {
        this.num = num;
        this.times = times;
    }
    public int compareTo(Pair pair) {
        return pair.times - times;
    }
}
```

python3:

```
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        frac = collections.Counter(nums)
        priorityQueue = []
        for key,value in frac.items():
            heapq.heappush(priorityQueue, (-value, key))
        result = []
        for _ in range(k):
            result.append(heapq.heappop(priorityQueue)[1])
        return result
```

295. Find Median from Data Stream

java:

```
class MedianFinder {
    PriorityQueue<Integer> minHeap = new PriorityQueue<>();
    PriorityQueue<Integer> maxHeap = new PriorityQueue<>
(Collections.reverseOrder());
    boolean isEven = true;

    public MedianFinder() {}

    public void addNum(int num) {
        if (isEven){
            minHeap.add(num);
            maxHeap.add(minHeap.poll());
        } else {
            maxHeap.add(num);
```

```
            minHeap.add(maxHeap.poll());
        }
        isEven = !isEven;
    }

    public double findMedian() {
        if (isEven)
            return (maxHeap.peek() + minHeap.peek())/2.0d;
        return maxHeap.peek();
    }
}
```

python3:

```
class MedianFinder:
    def __init__(self):
        self.maxHeap = []
        self.minHeap = []
        self.isEven = True

    def addNum(self, num: int) -> None:
        if self.isEven:
            heappush(self.minHeap, num)
            heappush(self.maxHeap, -heappop(self.minHeap))
        else:
            heappush(self.maxHeap, -num)
            heappush(self.minHeap, -heappop(self.maxHeap))
        self.isEven = not self.isEven

    def findMedian(self) -> float:
        if self.isEven:
            return (-self.maxHeap[0] + self.minHeap[0]) / 2.0
        return -self.maxHeap[0]
```

14 Patterns to Ace Any Coding Interview Question

- One of the most common points of anxiety developers:
    - **Have I solved enough practice questions? Could I have done more?**
- If you understand the generic patterns, you can use them as a template to solve a myriad of other problems with slight variations. Here, I've laid out the top **14 patterns** that can be used to solve any coding interview question, as well as **how to identify each pattern** for each:

1. **Sliding Window**
    - The Sliding Window pattern is used to perform a required operation on a specific window size of a given array or linked list, such as finding the longest subarray containing all 1s. Sliding Windows start from the 1st element and keep shifting right by one element and adjust the length of the window according to the problem that you are solving. In some

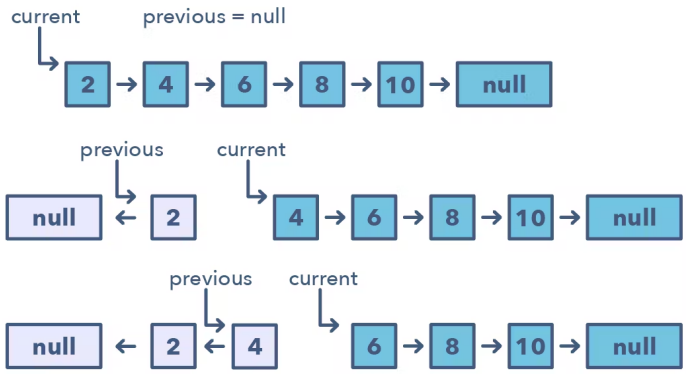
cases, the window size remains constant and in other cases the sizes grows or shrinks.

- **Ways to identify when to use the sliding window pattern**:
  - The problem input is a linear data structure such as a linked list, array, or string
  - You're asked to find the longest/shortest substring, subarray, or a desired value

2. **Two Pointers or Iterators**
   - Two Pointers is a pattern where two pointers iterate through the data structure in tandem until one or both of the pointers hit a certain condition. Two Pointers is often useful when searching pairs in a sorted array or linked list; for example, when you have to compare each element of an array to its other elements.
   - Two pointers are needed because with just pointer, you would have to continually loop back through the array to find the answer. This back and forth with a single iterator is inefficient for time and space complexity — a concept referred to as asymptotic analysis. While the brute force or naive solution with 1 pointer would work, it will produce something along the lines of $O(n^2)$. In many cases, two pointers can help you find a solution with better space or runtime complexity.
   - **Ways to identify when to use the Two Pointer pattern:**
     - It will feature problems where you deal with sorted arrays (or Linked Lists) and need to find a set of elements that fulfill certain constraints
     - The set of elements in the array is a pair, a triplet, or even a subarray
3. **Fast and Slow pointers**
   - The Fast and Slow pointer approach, also known as the **Hare & Tortoise algorithm**, is a pointer algorithm that uses two pointers which move through the array (or sequence/linked list) at different speeds. **This approach is quite useful when dealing with cyclic linked lists or arrays.**
   - **Ways to identify when to use the Fast and Slow pattern:**
     - The problem will deal with a loop in a linked list or array
     - When you need to know the position of a certain element or the overall length of the linked list.
4. **Merge Intervals**
   - The Merge Intervals pattern is an efficient technique to deal with overlapping intervals. In a lot of problems involving intervals, you either need to find overlapping intervals or merge intervals if they overlap. The pattern works like this:
   - Given two intervals ('a' and 'b'), there will be six different ways the two intervals can relate to each other
   - Understanding and recognizing these six cases will help you help you solve a wide range of problems from inserting intervals to optimizing interval merges.
   - **Ways to identify when to use the Merge Intervals pattern:**
     - If you're asked to produce a list with only mutually exclusive intervals
     - If you hear the term "overlapping intervals".
5. **Cyclic sort**

- This pattern describes an interesting approach to deal with problems involving arrays containing numbers in a given range. The Cyclic Sort pattern iterates over the array one number at a time, and if the current number you are iterating is not at the correct index, you swap it with the number at its correct index. You could try placing the number in its correct index, but this will produce a complexity of O(n^2) which is not optimal, hence the Cyclic Sort pattern.
- **Ways to identify when to use the Cyclic sort pattern:**
  - They will be problems involving a sorted array with numbers in a given range
  - If the problem asks you to find the missing/duplicate/smallest number in an sorted/rotated array

6. **In-place reversal of linked list**
   - In a lot of problems, you may be asked to reverse the links between a set of nodes of a linked list. Often, the constraint is that you need to do this in-place, i.e., using the existing node objects and without using extra memory. This is where the above mentioned pattern is useful.
   - This pattern reverses one node at a time starting with one variable (current) pointing to the head of the linked list, and one variable (previous) will point to the previous node that you have processed. In a lock-step manner, you will reverse the current node by pointing it to the previous before moving on to the next node. Also, you will update the variable "previous" to always point to the previous node that you have processed.



   - **Ways to identify when to use the reversal pattern:**
     - If you're asked to reverse a linked list without using extra memory

7. **Tree BFS**
   - This pattern is based on the Breadth First Search (BFS) technique to traverse a tree and uses a queue to keep track of all the nodes of a level before jumping onto the next level. Any problem involving the traversal of a tree in a level-by-level order can be efficiently solved using this approach.
   - The Tree BFS pattern works by pushing the root node to the queue and then continually iterating until the queue is empty. For each iteration, we remove the node at the head of the queue and "visit" that node. After removing each node from the queue, we also insert all of its children into the queue.
   - **Ways to identify when to use the Tree BFS pattern:**
     - If you're asked to traverse a tree in a level-by-level fashion (or level order traversal)

8. **Tree DFS**
    - Tree DFS is based on the Depth First Search (DFS) technique to traverse a tree.
    - You can use recursion (or a stack for the iterative approach) to keep track of all the previous (parent) nodes while traversing.
    - The Tree DFS pattern works by starting at the root of the tree, if the node is not a leaf you need to do three things:
        - Decide whether to process the current node now (pre-order), or between processing two children (in-order) or after processing both children (post-order).
        - Make two recursive calls for both the children of the current node to process them.
    - **Ways to identify when to use the Tree DFS pattern:**
        - If you're asked to traverse a tree with in-order, preorder, or postorder DFS
        - If the problem requires searching for something where the node is closer to a leaf
9. **Two heaps**
    - In many problems, we are given a set of elements such that we can divide them into two parts. To solve the problem, we are interested in knowing the smallest element in one part and the biggest element in the other part. This pattern is an efficient approach to solve such problems.
    - This pattern uses two heaps; A Min Heap to find the smallest element and a Max Heap to find the biggest element. The pattern works by storing the first half of numbers in a Max Heap, this is because you want to find the largest number in the first half. You then store the second half of numbers in a Min Heap, as you want to find the smallest number in the second half. At any time, the median of the current list of numbers can be calculated from the top element of the two heaps.
    - **Ways to identify the Two Heaps pattern:**
        - Useful in situations like Priority Queue, Scheduling
        - If the problem states that you need to find the smallest/largest/median elements of a set
        - Sometimes, useful in problems featuring a binary tree data structure
10. **Subsets**
    - A huge number of coding interview problems involve dealing with Permutations and Combinations of a given set of elements. The pattern Subsets describes an efficient Breadth First Search (BFS) approach to handle all these problems.
    - **Ways to identify the Subsets pattern:**
        - Problems where you need to find the combinations or permutations of a given set
11. **Modified binary search**
    - Whenever you are given a sorted array, linked list, or matrix, and are asked to find a certain element, the best algorithm you can use is the Binary Search. This pattern describes an efficient way to handle all problems involving Binary Search.
    - The patterns looks like this for an ascending order set:

        - First, find the middle of start and end. An easy way to find the middle would be: middle = (start + end) / 2. But this has a good chance of producing an integer overflow so it's recommended that you represent the middle as: middle = start + (end — start) / 2
        - If the key is equal to the number at index middle then return middle
        - If 'key' isn't equal to the index middle:
        - Check if key < arr[middle]. If it is reduce your search to end = middle - 1
        - Check if key > arr[middle]. If it is reduce your search to end = middle + 1
12. **Top K elements**
    - Any problem that asks us to find the top/smallest/frequent 'K' elements among a given set falls under this pattern.
    - The best data structure to keep track of 'K' elements is Heap. This pattern will make use of the Heap to solve multiple problems dealing with 'K' elements at a time from a set of given elements. The pattern looks like this:
        - Insert 'K' elements into the min-heap or max-heap based on the problem.
        - Iterate through the remaining numbers and if you find one that is larger than what you have in the heap, then remove that number and insert the larger one.
    - **Ways to identify when to use the Top 'K' Elements pattern:**
        - If you're asked to find the top/smallest/frequent 'K' elements of a given set
        - If you're asked to sort an array to find an exact element
13. **K-way Merge**
    - K-way Merge helps you solve problems that involve a set of sorted arrays.
    - Whenever you're given 'K' sorted arrays, you can use a Heap to efficiently perform a sorted traversal of all the elements of all arrays. You can push the smallest element of each array in a Min Heap to get the overall minimum. After getting the overall minimum, push the next element from the same array to the heap. Then, repeat this process to make a sorted traversal of all elements.
    - **Ways to identify when to use the K-way Merge pattern:**
        - The problem will feature sorted arrays, lists, or a matrix
        - If the problem asks you to merge sorted lists, find the smallest element in a sorted list.
14. **Topological sort**
    - Topological Sort is used to find a linear ordering of elements that have dependencies on each other. For example, if event 'B' is dependent on event 'A', 'A' comes before 'B' in topological ordering.
    - This pattern defines an easy way to understand the technique for performing topological sorting of a set of elements.
    - The pattern works like this:

1. Initialization
   a) Store the graph in adjacency lists by using a HashMap
   b) To find all sources, use a HashMap to keep the count of in-degreesBuild the graph and find in-degrees of all vertices
2. Build the graph from the input and populate the in-degrees HashMap.
3. Find all sources
   a) All vertices with ‘0’ in-degrees will be sources and are stored in a Queue.
4. Sort
   a) For each source, do the following things:
   —i) Add it to the sorted list.
   — ii)Get all of its children from the graph.
   — iii)Decrement the in-degree of each child by 1.
   — iv)If a child’s in-degree becomes ‘0’, add it to the sources Queue.
   b) Repeat (a), until the source Queue is empty.

- **Ways to identify when to use the Topological Sort pattern:**
  - The problem will deal with graphs that have no directed cycles
  - If you’re asked to update all objects in a sorted order
  - If you have a class of objects that follow a particular order