Crack75

A curated collection of 75 must-practice problems — interview-ready, concise solutions, and tips

What this PDF contains

75 carefully selected problems in Data Structures & Algorithms
Step-by-step solutions implemented in Python
Tagged by difficulty, topic, and relevant companies
Optimized to enhance learning efficiently

By Shaksham Agarwal | Engineer & Mentor | LinkedIn | Topmate

1. Two Sum

Problem :-

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

Topic: Array, Hash Table | **Difficulty:** Easy | **Companies:** Google, Facebook, Amazon

Brute Force :-

```
def twoSum(nums, target):
    for i in range(len(nums)):
        for j in range(i+1, len(nums)):
            if nums[i] + nums[j] == target:
                return [i, j]
```

Time Complexity :- O(n^2)
Space Complexity: O(1)

Optimized :-

```
def twoSum(nums, target):
    lookup = {}
    for i, num in enumerate(nums):
        if target - num in lookup:
            return [lookup[target - num], i]
        lookup[num] = i
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

2. Maximum Subarray (Kadane’s Algorithm)

Problem :-

Find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Topic: Array, Dynamic Programming | **Difficulty:** Medium | **Companies:** Amazon, Google, Microsoft

Brute Force :-

```
def maxSubArray(nums):
    max_sum = float('-inf')
    for i in range(len(nums)):
        sum_ = 0
        for j in range(i, len(nums)):
            sum_ += nums[j]
            max_sum = max(max_sum, sum_)
    return max_sum
```

Time Complexity :- O(n^2)
Space Complexity: O(1)

Optimized :-

```
def maxSubArray(nums):
    max_sum = curr_sum = nums[0]
    for num in nums[1:]:
        curr_sum = max(num, curr_sum + num)
        max_sum = max(max_sum, curr_sum)
    return max_sum
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

3. Move Zeroes

Problem :-

Move all 0's to the end of the array while maintaining the relative order of non-zero elements.

Topic: Array, Two Pointers | **Difficulty:** Easy | **Companies:** Facebook, Microsoft, Apple

Brute Force :-

```
def moveZeroes(nums):
    result = [x for x in nums if x != 0]
    zeros = [0] * (len(nums) - len(result))
    nums[:] = result + zeros
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```python
def moveZeroes(nums):
    last_non_zero = 0
    for i in range(len(nums)):
        if nums[i] != 0:
            nums[last_non_zero], nums[i] = nums[i], nums[last_non_zero]
            last_non_zero += 1
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

4. Best Time to Buy and Sell Stock

Problem :-

Given an array prices where prices[i] is the price of a stock on the ith day, find the maximum profit you can achieve by choosing one day to buy and another to sell later.

Topic: Array, Kadane Variant | **Difficulty:** Easy | **Companies:** Amazon, Google, Bloomberg

Brute Force :-

```python
def maxProfit(prices):
    max_profit = 0
    for i in range(len(prices)):
        for j in range(i+1, len(prices)):
            max_profit = max(max_profit, prices[j] - prices[i])
    return max_profit
```

Time Complexity :- O(n^2)
Space Complexity: O(1)

Optimized :-

```
def maxProfit(prices):
    min_price = float('inf')
    max_profit = 0
    for price in prices:
        min_price = min(min_price, price)
        max_profit = max(max_profit, price - min_price)
    return max_profit
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

5. Valid Parentheses

Problem :-

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

Topic: Stack | **Difficulty:** Easy | **Companies:** Amazon, Google, Adobe

Brute Force :-

```
def isValid(s):
    while '()' in s or '{}' in s or '[]' in s:
        s = s.replace('()', '').replace('{}', '').replace('[]', '')
    return s == ''
```

Time Complexity :- O(n^2)
Space Complexity: O(n)

Optimized :-

```
def isValid(s):
    stack = []
    pairs = {')': '(', ']': '[', '}': '{'}
    for ch in s:
        if ch in pairs.values():
            stack.append(ch)
        elif not stack or stack.pop() != pairs[ch]:
            return False
    return not stack
```

Time Complexity :- O(n)
Space Complexity: O(n)

6. Course Schedule

Problem :-

There are a total of numCourses you have to take, labeled from 0 to numCourses-1. Some courses have prerequisites. Determine if you can finish all courses.

Topic: Graph, Topological Sort | **Difficulty:** Medium | **Companies:** Google, Facebook, Microsoft

Brute Force :-

```
def canFinish(numCourses, prerequisites):
    from itertools import permutations
    # Brute force is not practical for large input; placeholder to show intent
    return False
```

Time Complexity :- O(n!)
Space Complexity: O(n)

Optimized :-

```
from collections import defaultdict, deque

def canFinish(numCourses, prerequisites):
    graph = defaultdict(list)
    indegree = [0] * numCourses
    for u, v in prerequisites:
        graph[v].append(u)
        indegree[u] += 1
    q = deque([i for i in range(numCourses) if indegree[i] == 0])
    count = 0
    while q:
        course = q.popleft()
        count += 1
        for nei in graph[course]:
            indegree[nei] -= 1
            if indegree[nei] == 0:
                q.append(nei)
    return count == numCourses
```

Time Complexity :- O(V + E)
Space Complexity: O(V + E)

7. Number of Islands

Problem :-

Given an m x n 2D grid map of '1's (land) and '0's (water), return the number of islands.

Topic: Graph, DFS, BFS, Disjoint Set | **Difficulty:** Medium | **Companies:** Amazon, Google, Microsoft

Brute Force :-

```
def numIslands(grid):
    # Brute: check every land and perform BFS repeatedly
    visited = set()
    count = 0
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            if grid[i][j] == '1' and (i, j) not in visited:
                count += 1
                q = [(i, j)]
                while q:
                    x, y = q.pop(0)
                    for dx, dy in [(1,0),(-1,0),(0,1),(0,-1)]:
                        nx, ny = x+dx, y+dy
                        if 0<=nx<len(grid) and 0<=ny<len(grid[0]) and grid[nx][ny]=='
                            visited.add((nx,ny))
    return count
```

Time Complexity :- O(m*n)
Space Complexity: O(m*n)

Optimized :-

```
def numIslands(grid):
    if not grid: return 0
    rows, cols = len(grid), len(grid[0])
    def dfs(r, c):
        if r<0 or c<0 or r>=rows or c>=cols or grid[r][c]=='0': return
        grid[r][c] = '0'
        dfs(r+1,c); dfs(r-1,c); dfs(r,c+1); dfs(r,c-1)
    count = 0
    for r in range(rows):
        for c in range(cols):
            if grid[r][c]=='1':
                count += 1
                dfs(r,c)
    return count
```

Time Complexity :- O(m*n)
Space Complexity: O(m*n)

---

8. Clone Graph

Problem :-

Given a reference of a node in a connected undirected graph, return a deep copy (clone) of the graph.

Topic: Graph, BFS, DFS | **Difficulty:** Medium | **Companies:** Amazon, Facebook, Microsoft

Brute Force :-

```
def cloneGraph(node):
    # Brute force cloning is not practical without mapping
    return None
```

Time Complexity :- O(V + E)
Space Complexity: O(V)

Optimized :-

```
def cloneGraph(node):
    if not node:
        return None
    from collections import deque
    clones = {node: Node(node.val, [])}
    q = deque([node])
    while q:
        n = q.popleft()
        for nei in n.neighbors:
            if nei not in clones:
                clones[nei] = Node(nei.val, [])
                q.append(nei)
            clones[n].neighbors.append(clones[nei])
    return clones[node]
```

Time Complexity :- O(V + E)
Space Complexity: O(V)

---

9. Minimum Depth of Binary Tree

Problem :-

Given a binary tree, find its minimum depth. The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

Topic: Tree, BFS | **Difficulty:** Easy | **Companies:** Google, Amazon, Adobe

Brute Force :-

```
def minDepth(root):
    if not root: return 0
    if not root.left: return 1 + minDepth(root.right)
    if not root.right: return 1 + minDepth(root.left)
    return 1 + min(minDepth(root.left), minDepth(root.right))
```

Time Complexity :- O(n)
Space Complexity: O(h)

Optimized :-

```
from collections import deque

def minDepth(root):
    if not root: return 0
    q = deque([(root, 1)])
    while q:
        node, depth = q.popleft()
        if not node.left and not node.right:
            return depth
        if node.left: q.append((node.left, depth+1))
        if node.right: q.append((node.right, depth+1))
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

10. Binary Tree Level Order Traversal

Problem :-

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

Topic: Tree, BFS | **Difficulty:** Medium | **Companies:** Amazon, Google, Meta

Brute Force :-

```
def levelOrder(root):
    if not root: return []
    res = []
    level = [root]
    while level:
        res.append([n.val for n in level])
        next_level = []
        for n in level:
            if n.left: next_level.append(n.left)
            if n.right: next_level.append(n.right)
        level = next_level
    return res
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```
from collections import deque

def levelOrder(root):
    if not root: return []
    res, q = [], deque([root])
    while q:
        level = []
        for _ in range(len(q)):
            node = q.popleft()
            level.append(node.val)
            if node.left: q.append(node.left)
            if node.right: q.append(node.right)
        res.append(level)
    return res
```

Time Complexity :- O(n)
Space Complexity: O(n)

11. Serialize and Deserialize Binary Tree

Problem :-

Design an algorithm to serialize and deserialize a binary tree. Serialization is the process of converting a data structure into a string, and deserialization is the reverse.

Topic: Tree | **Difficulty:** Hard | **Companies:** Google, Amazon, Facebook

Brute Force :-

```python
def serialize(root):
    if not root: return '[]'
    from collections import deque
    res, q = [], deque([root])
    while q:
        n = q.popleft()
        if n:
            res.append(str(n.val))
            q.append(n.left)
            q.append(n.right)
        else:
            res.append('null')
    return '[' + ','.join(res) + ']'

def deserialize(data):
    if data == '[]': return None
    from collections import deque
    nodes = data[1:-1].split(',')
    root = TreeNode(int(nodes[0]))
    q = deque([root])
    idx = 1
    while q:
        n = q.popleft()
        if nodes[idx] != 'null':
            n.left = TreeNode(int(nodes[idx]))
            q.append(n.left)
        idx += 1
        if nodes[idx] != 'null':
            n.right = TreeNode(int(nodes[idx]))
            q.append(n.right)
        idx += 1
    return root
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```
def serialize(root):
    res = []
    def dfs(node):
        if not node:
            res.append('null')
            return
        res.append(str(node.val))
        dfs(node.left)
        dfs(node.right)
    dfs(root)
    return ','.join(res)

def deserialize(data):
    nodes = iter(data.split(','))
    def dfs():
        val = next(nodes)
        if val == 'null': return None
        node = TreeNode(int(val))
        node.left = dfs()
        node.right = dfs()
        return node
    return dfs()
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

12. Invert Binary Tree

Problem :-

Given the root of a binary tree, invert the tree and return its root.

Topic: Tree | **Difficulty:** Easy | **Companies:** Google, Amazon, Apple

Brute Force :-

```
def invertTree(root):
    if not root: return None
    root.left, root.right = root.right, root.left
    invertTree(root.left)
    invertTree(root.right)
    return root
```

Time Complexity :- O(n)
Space Complexity: O(h)

Optimized :-

```
from collections import deque

def invertTree(root):
    if not root: return None
    q = deque([root])
    while q:
        n = q.popleft()
        n.left, n.right = n.right, n.left
        if n.left: q.append(n.left)
        if n.right: q.append(n.right)
    return root
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

13. Maximum Depth of Binary Tree

Problem :-

Given a binary tree, find its maximum depth. The maximum depth is the number of nodes along the longest path from the root down to the farthest leaf node.

Topic: Tree, DFS | **Difficulty:** Easy | **Companies:** Google, Amazon, Facebook

Brute Force :-

```
def maxDepth(root):
    if not root: return 0
    return 1 + max(maxDepth(root.left), maxDepth(root.right))
```

Time Complexity :- O(n)
Space Complexity: O(h)

Optimized :-

```
from collections import deque

def maxDepth(root):
    if not root: return 0
    q = deque([(root, 1)])
    depth = 0
    while q:
        node, d = q.popleft()
        depth = max(depth, d)
        if node.left: q.append((node.left, d+1))
        if node.right: q.append((node.right, d+1))
    return depth
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

14. Validate Binary Search Tree

Problem :-

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

Topic: Tree, DFS | **Difficulty:** Medium | **Companies:** Google, Amazon, Bloomberg

Brute Force :-

```
def isValidBST(root):
    arr = []
    def inorder(node):
        if not node: return
        inorder(node.left)
        arr.append(node.val)
        inorder(node.right)
    inorder(root)
    return all(arr[i] < arr[i+1] for i in range(len(arr)-1))
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```
def isValidBST(root):
    def dfs(node, low, high):
        if not node: return True
        if not (low < node.val < high): return False
        return dfs(node.left, low, node.val) and dfs(node.right, node.val, high)
    return dfs(root, float('-inf'), float('inf'))
```

Time Complexity :- O(n)
Space Complexity: O(h)

---

15. kth Smallest Element in a BST

Problem :-

Given the root of a binary search tree, and an integer k, return the kth smallest value (1-indexed) of all the values of the nodes in the tree.

Topic: Tree, DFS | **Difficulty:** Medium | **Companies:** Google, Amazon, Microsoft

Brute Force :-

```
def kthSmallest(root, k):
    arr = []
    def inorder(node):
        if not node: return
        inorder(node.left)
        arr.append(node.val)
        inorder(node.right)
    inorder(root)
    return arr[k-1]
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```
def kthSmallest(root, k):
    stack = []
    while True:
        while root:
            stack.append(root)
            root = root.left
        root = stack.pop()
        k -= 1
        if k == 0:
            return root.val
        root = root.right
```

Time Complexity :- O(h + k)
Space Complexity: O(h)

---

16. Lowest Common Ancestor of a BST

Problem :-

Given a binary search tree, find the lowest common ancestor (LCA) of two given nodes in the BST.

Topic: Tree | **Difficulty:** Easy | **Companies:** Google, Amazon, Facebook

Brute Force :-

```
def lowestCommonAncestor(root, p, q):
    path_p, path_q = [], []
    def dfs(node, target, path):
        if not node: return False
        path.append(node)
        if node == target: return True
        if dfs(node.left, target, path) or dfs(node.right, target, path): return True
        path.pop()
        return False
    dfs(root, p, path_p)
    dfs(root, q, path_q)
    lca = None
    for i in range(min(len(path_p), len(path_q))):
        if path_p[i] == path_q[i]: lca = path_p[i]
        else: break
    return lca
```

Time Complexity :- O(n)
Space Complexity: O(h)

Optimized :-

```
def lowestCommonAncestor(root, p, q):
    while root:
        if p.val < root.val and q.val < root.val:
            root = root.left
        elif p.val > root.val and q.val > root.val:
            root = root.right
        else:
            return root
```

Time Complexity :- O(h)
Space Complexity: O(1)

---

17. Path Sum

Problem :-

Given the root of a binary tree and an integer targetSum, return true if the tree has a root-to-leaf path such that adding up all the values equals targetSum.

Topic: Tree, DFS | **Difficulty:** Easy | **Companies:** Amazon, Google, Facebook

Brute Force :-

```
def hasPathSum(root, targetSum):
    if not root: return False
    if not root.left and not root.right: return root.val == targetSum
    return hasPathSum(root.left, targetSum - root.val) or hasPathSum(root.right, targ
```

Time Complexity :- O(n)
Space Complexity: O(h)

Optimized :-

```
from collections import deque

def hasPathSum(root, targetSum):
    if not root: return False
    q = deque([(root, root.val)])
    while q:
        n, s = q.popleft()
        if not n.left and not n.right and s == targetSum:
            return True
        if n.left: q.append((n.left, s + n.left.val))
        if n.right: q.append((n.right, s + n.right.val))
    return False
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

18. Subtree of Another Tree

Problem :-

Given the roots of two binary trees root and subRoot, return true if there is a subtree of root with the same structure and node values of subRoot.

Topic: Tree, DFS | **Difficulty:** Easy | **Companies:** Amazon, Google, Meta

Brute Force :-

```
def isSubtree(root, subRoot):
    def isSame(a, b):
        if not a and not b: return True
        if not a or not b: return False
        return a.val == b.val and isSame(a.left, b.left) and isSame(a.right, b.right)
    if not root: return False
    if isSame(root, subRoot): return True
    return isSubtree(root.left, subRoot) or isSubtree(root.right, subRoot)
```

Time Complexity :- O(m*n)
Space Complexity: O(h)

Optimized :-

```
def isSubtree(root, subRoot):
    if not subRoot: return True
    if not root: return False
    def isSame(a, b):
        if not a and not b: return True
        if not a or not b: return False
        return a.val == b.val and isSame(a.left, b.left) and isSame(a.right, b.right)
    if isSame(root, subRoot): return True
    return isSubtree(root.left, subRoot) or isSubtree(root.right, subRoot)
```

Time Complexity :- O(m*n)
Space Complexity: O(h)

---

19. Implement Trie

Problem :-

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

Topic: Design, Trie | **Difficulty:** Medium | **Companies:** Bloomberg, Uber, Adobe

Brute Force :-

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.isEnd = False

class Trie:
    def __init__(self):
        self.root = TrieNode()
    def insert(self, word):
        node = self.root
        for c in word: node = node.children.setdefault(c, TrieNode())
        node.isEnd = True
    def search(self, word):
        node = self.root
        for c in word:
            if c not in node.children: return False
            node = node.children[c]
        return node.isEnd
    def startsWith(self, prefix):
        node = self.root
        for c in prefix:
            if c not in node.children: return False
            node = node.children[c]
        return True
```

Time Complexity :- O(n) per operation
Space Complexity: O(n*alphabet_size)

Optimized :-

```
class TrieNode:
    def __init__(self):
        self.children = [None]*26
        self.isEnd=False

class Trie:
    def __init__(self): self.root=TrieNode()
    def _charToIndex(self,c): return ord(c)-ord('a')
    def insert(self, word):
        n=node=self.root
        for c in word:
            n=n.children[self._charToIndex(c)]=n.children[self._charToIndex(c)] or Tr
        n.isEnd=True
    def search(self, word):
        n=node=self.root
        for c in word:
            if not n.children[self._charToIndex(c)]: return False
            n=n.children[self._charToIndex(c)]
        return n.isEnd
    def startsWith(self,prefix):
        n=node=self.root
        for c in prefix:
            if not n.children[self._charToIndex(c)]: return False
            n=n.children[self._charToIndex(c)]
        return True
```

Time Complexity :- O(n) per operation
Space Complexity: O(n*alphabet_size)

---

20. Word Dictionary

Problem :-

Design a data structure that supports adding words and searching if a string matches any previously added string, where '.' can match any character.

Topic: Design, Trie | **Difficulty:** Medium | **Companies:** Airbnb, DoorDash, Spotify

Brute Force :-

```
class WordDictionary:
    def __init__(self): self.words=[]
    def addWord(self, word): self.words.append(word)
    def search(self, word):
        for w in self.words:
            if len(w)==len(word) and all(c1==c2 or c2=='.' for c1,c2 in zip(w,word)):
        return False
```

Time Complexity :- O(n*m)
Space Complexity: O(n*m)

Optimized :-

```
class TrieNode:
    def __init__(self): self.children = {}; self.isEnd=False

class WordDictionary:
    def __init__(self): self.root=TrieNode()
    def addWord(self,word):
        n=self.root
        for c in word: n=n.children.setdefault(c,TrieNode())
        n.isEnd=True
    def search(self,word):
        def dfs(node,i):
            if i==len(word): return node.isEnd
            c=word[i]
            if c=='.': return any(dfs(n,i+1) for n in node.children.values())
            return c in node.children and dfs(node.children[c],i+1)
        return dfs(self.root,0)
```

Time Complexity :- O(n)
Space Complexity: O(n*alphabet_size)

21. LRU Cache

Problem :-

Design a data structure that follows the constraints of a Least Recently Used (LRU) cache.

Topic: Design, LinkedHashMap-like | **Difficulty:** Medium | **Companies:** Snapchat, Reddit, Pinterest

Brute Force :-

```
class LRUCache:
    def __init__(self, capacity):
        self.capacity=capacity
        self.cache={}
        self.order=[]
    def get(self,key):
        if key not in self.cache: return -1
        self.order.remove(key); self.order.append(key)
        return self.cache[key]
    def put(self,key,value):
        if key in self.cache: self.order.remove(key)
        elif len(self.cache)==self.capacity: old=self.order.pop(0); del self.cache[ol
        self.cache[key]=value; self.order.append(key)
```

Time Complexity :- O(n) per operation
Space Complexity: O(capacity)

Optimized :-

```
from collections import OrderedDict
class LRUCache(OrderedDict):
    def __init__(self,capacity):
        super().__init__(); self.capacity=capacity
    def get(self,key):
        if key not in self: return -1
        self.move_to_end(key); return self[key]
    def put(self,key,value):
        if key in self: self.move_to_end(key)
        self[key]=value
        if len(self)>self.capacity: self.popitem(last=False)
```

Time Complexity :- O(1) per operation
Space Complexity: O(capacity)

---

22. Min Stack

Problem :-

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

Topic: Design, Stack | **Difficulty:** Easy | **Companies:** LinkedIn, Square, Twitter

Brute Force :-

```
class MinStack:
    def __init__(self): self.stack=[]
    def push(self,x): self.stack.append(x)
    def pop(self): self.stack.pop()
    def top(self): return self.stack[-1]
    def getMin(self): return min(self.stack)
```

Time Complexity :- O(n) for getMin
Space Complexity: O(n)

Optimized :-

```
class MinStack:
    def __init__(self): self.stack=[]
    def push(self,x):
        min_val=x if not self.stack else min(x,self.stack[-1][1])
        self.stack.append((x,min_val))
    def pop(self): self.stack.pop()
    def top(self): return self.stack[-1][0]
    def getMin(self): return self.stack[-1][1]
```

Time Complexity :- O(1) all operations
Space Complexity: O(n)

---

23. Sliding Window Maximum

Problem :-

Given an array nums and a sliding window size k, return the maximum in each sliding window.

Topic: Deque | **Difficulty:** Hard | **Companies:** Stripe, Robinhood, Shopify

Brute Force :-

```
def maxSlidingWindow(nums,k):
    res=[]
    for i in range(len(nums)-k+1): res.append(max(nums[i:i+k]))
    return res
```

Time Complexity :- O(n*k)
Space Complexity: O(n)

Optimized :-

```
from collections import deque

def maxSlidingWindow(nums,k):
    q=deque(); res=[]
    for i,n in enumerate(nums):
        while q and q[0]<i-k+1: q.popleft()
        while q and nums[q[-1]]<n: q.pop()
        q.append(i)
        if i>=k-1: res.append(nums[q[0]])
    return res
```

Time Complexity :- O(n)
Space Complexity: O(k)

---

24. K Closest Points to Origin

Problem :-

Given points on a 2D plane, return the k closest points to the origin.

Topic: Heap, Quickselect | **Difficulty:** Medium | **Companies:** Coinbase, Instacart, Dropbox

Brute Force :-

```
def kClosest(points,k):
    points.sort(key=lambda x: x[0]**2+x[1]**2)
    return points[:k]
```

Time Complexity :- O(n log n)
Space Complexity: O(1)

Optimized :-

```
import random

def kClosest(points,k):
    def partition(l,r):
        pivot=points[r]; pi=l
        for i in range(l,r):
            if points[i][0]**2+points[i][1]**2<=pivot[0]**2+pivot[1]**2:
                points[i],points[pi]=points[pi],points[i]; pi+=1
        points[pi],points[r]=points[r],points[pi]
        return pi
    l,r=0,len(points)-1
    while True:
        pi=random.randint(l,r); pi=partition(l,r)
        if pi==k: break
        elif pi<k: l=pi+1
        else: r=pi-1
    return points[:k]
```

Time Complexity :- O(n) average
Space Complexity: O(1)

---

25. Merge Sorted Array

Problem :-

Merge two sorted arrays nums1 and nums2 into nums1 as one sorted array.

Topic: Two Pointers | **Difficulty:** Easy | **Companies:** Atlassian, Uber, Dropbox

Brute Force :-

```
def merge(nums1,m,nums2,n):
    nums1[m:]=nums2[:]; nums1.sort()
```

Time Complexity :- O((m+n) log(m+n))
Space Complexity: O(1)

Optimized :-

```
def merge(nums1,m,nums2,n):
    p1,m1=p1=m-1; p2=n-1; p=m+n-1
    while p1>=0 and p2>=0:
        if nums1[p1]>nums2[p2]: nums1[p]=nums1[p1]; p1-=1
        else: nums1[p]=nums2[p2]; p2-=1
        p-=1
    while p2>=0: nums1[p]=nums2[p2]; p2-=1; p-=1
```

Time Complexity :- O(m+n)
Space Complexity: O(1)

---

26. Reverse Linked List

Problem :-

Reverse a singly linked list.

Topic: Linked List | **Difficulty:** Easy | **Companies:** Uber, Shopify, Coinbase

Brute Force :-

```
def reverseList(head):
    res=[]
    while head: res.append(head.val); head=head.next
    dummy=ListNode(0); curr=dummy
    for val in reversed(res): curr.next=ListNode(val); curr=curr.next
    return dummy.next
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```
def reverseList(head):
    prev=None; curr=head
    while curr: tmp=curr.next; curr.next=prev; prev=curr; curr=tmp
    return prev
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

27. Merge Two Sorted Lists

Problem :-

Merge two sorted linked lists and return it as a new sorted list.

Topic: Linked List | **Difficulty:** Easy | **Companies:** Atlassian, LinkedIn, Dropbox

Brute Force :-

```
def mergeTwoLists(l1,l2):
    res=[]
    while l1: res.append(l1.val); l1=l1.next
    while l2: res.append(l2.val); l2=l2.next
    res.sort()
    dummy=curr=ListNode(0)
    for val in res: curr.next=ListNode(val); curr=curr.next
    return dummy.next
```

Time Complexity :- O((n+m) log(n+m))
Space Complexity: O(n+m)

Optimized :-

```
def mergeTwoLists(l1,l2):
    dummy=curr=ListNode(0)
    while l1 and l2:
        if l1.val<l2.val: curr.next=l1; l1=l1.next
        else: curr.next=l2; l2=l2.next
        curr=curr.next
    curr.next=l1 or l2
    return dummy.next
```

Time Complexity :- O(n+m)
Space Complexity: O(1)

---

28. Add Two Numbers

Problem :-

Add two numbers represented by linked lists where digits are stored in reverse order.

Topic: Linked List | **Difficulty:** Medium | **Companies:** Stripe, Robinhood, Coinbase

Brute Force :-

```
def addTwoNumbers(l1,l2):
    s1=s2=''
    while l1: s1=str(l1.val)+s1; l1=l1.next
    while l2: s2=str(l2.val)+s2; l2=l2.next
    s=str(int(s1)+int(s2))
    dummy=curr=ListNode(0)
    for c in reversed(s): curr.next=ListNode(int(c)); curr=curr.next
    return dummy.next
```

Time Complexity :- O(max(n,m))
Space Complexity: O(max(n,m))

Optimized :-

```
def addTwoNumbers(l1,l2):
    dummy=curr=ListNode(0)
    carry=0
    while l1 or l2 or carry:
        x=l1.val if l1 else 0
        y=l2.val if l2 else 0
        sum_=x+y+carry; carry=sum_//10
        curr.next=ListNode(sum_%10); curr=curr.next
        l1=l1.next if l1 else None
        l2=l2.next if l2 else None
    return dummy.next
```

Time Complexity :- O(max(n,m))
Space Complexity: O(1)

---

29. Remove Nth Node From End

Problem :-

Remove the nth node from the end of a linked list and return its head.

Topic: Two-pass, One-pass | **Difficulty:** Medium | **Companies:** Google, Uber, Microsoft

Brute Force :-

```
def removeNthFromEnd(head,n):
    length=0
    node=head
    while node: length+=1; node=node.next
    node=dummy=ListNode(0); dummy.next=head
    for _ in range(length-n): node=node.next
    node.next=node.next.next
    return dummy.next
```

Time Complexity :- O(n)
Space Complexity: O(1)

Optimized :-

```
def removeNthFromEnd(head,n):
    dummy=ListNode(0); dummy.next=head
    fast=slow=dummy
    for _ in range(n+1): fast=fast.next
    while fast: fast=fast.next; slow=slow.next
    slow.next=slow.next.next
    return dummy.next
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

30. Linked List Cycle

Problem :-

Detect if a linked list has a cycle.

Topic: Floyd’s Tortoise and Hare | **Difficulty:** Easy | **Companies:** Facebook, Pinterest, Bloomberg

Brute Force :-

```
def hasCycle(head):
    visited=set()
    while head:
        if head in visited: return True
        visited.add(head)
        head=head.next
    return False
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```
def hasCycle(head):
    slow=fast=head
    while fast and fast.next:
        slow=slow.next; fast=fast.next.next
        if slow==fast: return True
    return False
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

31. Reorder List

Problem :-

Reorder a linked list from L0→L1→…→Ln-1→Ln to L0→Ln→L1→Ln-1→…

Topic: Linked List | **Difficulty:** Medium | **Companies:** Amazon, Airbnb, Google

Brute Force :-

```
def reorderList(head):
    nodes=[]
    n=node=head
    while n: nodes.append(n); n=n.next
    i,j=0,len(nodes)-1
    while i<j:
        nodes[i].next=nodes[j]; i+=1
        nodes[j].next=nodes[i]; j-=1
    nodes[i].next=None
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```
def reorderList(head):
    if not head: return
    slow=fast=head
    while fast and fast.next: slow=slow.next; fast=fast.next.next
    prev=None; curr=slow.next; slow.next=None
    while curr: tmp=curr.next; curr.next=prev; prev=curr; curr=tmp
    first,second=head,prev
    while second: tmp1, tmp2=first.next, second.next; first.next=second; second.next=
```

Time Complexity :- O(n)
Space Complexity: O(1)

32. Copy List with Random Pointer

Problem :-

Given a linked list with a random pointer, make a deep copy of the list.

Topic: Hash, Interleaving | **Difficulty:** Medium | **Companies:** Uber, Facebook, Dropbox

Brute Force :-

```
def copyRandomList(head):
    old_to_new={}
    curr=head
    while curr: old_to_new[curr]=Node(curr.val); curr=curr.next
    curr=head
    while curr:
        old_to_new[curr].next=old_to_new.get(curr.next)
        old_to_new[curr].random=old_to_new.get(curr.random)
        curr=curr.next
    return old_to_new.get(head)
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```
def copyRandomList(head):
    if not head: return None
    curr=head
    while curr: tmp=Node(curr.val); tmp.next=curr.next; curr.next=tmp; curr=tmp.next
    curr=head
    while curr: curr.next.random=curr.random.next if curr.random else None; curr=curr
    curr=head; new=head.next; dummy=new
    while curr: curr.next=curr.next.next; new.next=new.next.next if new.next else Non
    return dummy
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

33. Binary Search Tree Iterator

Problem :-

Implement an iterator over a BST with next() and hasNext() in O(1) average time and O(h) space.

Topic: Design, BST | **Difficulty:** Medium | **Companies:** Microsoft, Airbnb, Google

Brute Force :-

```
class BSTIterator:
    def __init__(self, root):
        self.nodes=[]
        self._inorder(root)
        self.index=0
    def _inorder(self,node):
        if not node: return
        self._inorder(node.left)
        self.nodes.append(node.val)
        self._inorder(node.right)
    def next(self): val=self.nodes[self.index]; self.index+=1; return val
    def hasNext(self): return self.index<len(self.nodes)
```

Time Complexity :- O(n) preprocessing
Space Complexity: O(n)

Optimized :-

```
class BSTIterator:
    def __init__(self, root):
        self.stack=[]; self._pushLeft(root)
    def _pushLeft(self,node):
        while node: self.stack.append(node); node=node.left
    def next(self):
        node=self.stack.pop(); self._pushLeft(node.right); return node.val
    def hasNext(self): return len(self.stack)>0
```

Time Complexity :- O(1) amortized per next()
Space Complexity: O(h)

---

34. Number of Connected Components in an Undirected Graph

Problem :-

Count the number of connected components in an undirected graph.

Topic: Union-Find | **Difficulty:** Medium | **Companies:** Facebook, Coinbase, Stripe

Brute Force :-

```
def countComponents(n,edges):
    adj=[[] for _ in range(n)]
    for u,v in edges: adj[u].append(v); adj[v].append(u)
    visited=set(); count=0
    def dfs(u):
        visited.add(u)
        for v in adj[u]:
            if v not in visited: dfs(v)
    for i in range(n):
        if i not in visited: dfs(i); count+=1
    return count
```

Time Complexity :- O(n+e)
Space Complexity: O(n+e)

Optimized :-

```
def countComponents(n,edges):
    parent=list(range(n))
    def find(x):
        while x!=parent[x]: parent[x]=parent[parent[x]]; x=parent[x]; return x
    for u,v in edges: parent[find(u)]=find(v)
    return len(set(find(i) for i in range(n)))
```

Time Complexity :- O(n*α(n))
Space Complexity: O(n)

---

35. Subarray Sum Equals K

Problem :-

Given an array and integer k, find the total number of continuous subarrays whose sum equals k.

Topic: Hash, Prefix Sum | **Difficulty:** Medium | **Companies:** Amazon, Spotify, Bloomberg

Brute Force :-

```
def subarraySum(nums,k):
    count=0
    for i in range(len(nums)):
        sum_=0
        for j in range(i,len(nums)):
            sum_+=nums[j]
            if sum_==k: count+=1
    return count
```

Time Complexity :- O(n^2)
Space Complexity: O(1)

Optimized :-

```
def subarraySum(nums,k):
    count=0; preSum={0:1}; s=0
    for num in nums:
        s+=num
        count+=preSum.get(s-k,0)
        preSum[s]=preSum.get(s,0)+1
    return count
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

36. Maximum Product Subarray

Problem :-

Find the contiguous subarray within an array which has the largest product.

Topic: DP | **Difficulty:** Medium | **Companies:** Google, Uber, Atlassian

Brute Force :-

```
def maxProduct(nums):
    res=float('-inf')
    for i in range(len(nums)):
        prod=1
        for j in range(i,len(nums)):
            prod*=nums[j]; res=max(res,prod)
    return res
```

Time Complexity :- O(n^2)
Space Complexity: O(1)

Optimized :-

```
def maxProduct(nums):
    max_prod=min_prod=result=nums[0]
    for n in nums[1:]:
        if n<0: max_prod,min_prod=min_prod,max_prod
        max_prod=max(n,max_prod*n)
        min_prod=min(n,min_prod*n)
        result=max(result,max_prod)
    return result
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

37. House Robber

Problem :-

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount you can rob without alerting the police (no two adjacent).

Topic: DP | **Difficulty:** Easy | **Companies:** Facebook, Airbnb, Google

Brute Force :-

```
def rob(nums):
    if not nums: return 0
    res=[]
    def dfs(i,sum_):
        if i>=len(nums): res.append(sum_); return
        dfs(i+1,sum_+nums[i]); dfs(i+2,sum_)
    dfs(0,0)
    return max(res)
```

Time Complexity :- O(2^n)
Space Complexity: O(n)

Optimized :-

```
def rob(nums):
    if not nums: return 0
    prev=curr=0
    for n in nums: prev,curr=curr,max(curr,prev+n)
    return curr
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

38. House Robber II

Problem :-

Houses are in a circle. Determine the maximum you can rob without alerting the police.

Topic: DP, Circular | **Difficulty:** Medium | **Companies:** Amazon, LinkedIn, Coinbase

Brute Force :-

```
def rob(nums):
    if not nums: return 0
    if len(nums)==1: return nums[0]
    return max(rob_linear(nums[:-1]), rob_linear(nums[1:]))

def rob_linear(nums):
    prev=curr=0
    for n in nums: prev,curr=curr,max(curr,prev+n)
    return curr
```

Time Complexity :- O(n)
Space Complexity: O(1)

Optimized :-

```
def rob(nums):
    if len(nums)==1: return nums[0]
    return max(rob_linear(nums[:-1]), rob_linear(nums[1:]))

def rob_linear(nums):
    prev=curr=0
    for n in nums: prev,curr=curr,max(curr,prev+n)
    return curr
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

39. Longest Increasing Subsequence

Problem :-

Given an integer array, find the length of the longest strictly increasing subsequence.

Topic: DP, Patience Sorting | **Difficulty:** Medium | **Companies:** Google, Microsoft, Airbnb

Brute Force :-

```
def lengthOfLIS(nums):
    n=len(nums)
    dp=[1]*n
    for i in range(n):
        for j in range(i):
            if nums[i]>nums[j]: dp[i]=max(dp[i],dp[j]+1)
    return max(dp)
```

Time Complexity :- O(n^2)
Space Complexity: O(n)

Optimized :-

```
import bisect

def lengthOfLIS(nums):
    sub=[]
    for x in nums:
        i=bisect.bisect_left(sub,x)
        if i==len(sub): sub.append(x)
        else: sub[i]=x
    return len(sub)
```

Time Complexity :- O(n log n)
Space Complexity: O(n)

---

40. Coin Change

Problem :-

Given coins of different denominations and an amount, compute the fewest coins needed to make up that amount.

Topic: DP | **Difficulty:** Medium | **Companies:** Amazon, Facebook, Stripe

Brute Force :-

```
def coinChange(coins,amount):
    res=[]
    def dfs(rem,count):
        if rem==0: res.append(count); return
        for c in coins:
            if rem-c>=0: dfs(rem-c,count+1)
    dfs(amount,0)
    return min(res) if res else -1
```

Time Complexity :- O(n^amount)
Space Complexity: O(amount)

Optimized :-

```
def coinChange(coins,amount):
    dp=[float('inf')]*(amount+1); dp[0]=0
    for i in range(1,amount+1):
        for c in coins:
            if i>=c: dp[i]=min(dp[i],dp[i-c]+1)
    return dp[amount] if dp[amount]!=float('inf') else -1
```

Time Complexity :- O(n*amount)
Space Complexity: O(amount)

---

41. Combination Sum

Problem :-

Given a set of candidate numbers (without duplicates) and a target, find all unique combinations where the candidate numbers sum to target. You may use the same number multiple times.

Topic: Backtracking | **Difficulty:** Medium | **Companies:** Uber, LinkedIn, Coinbase

Brute Force :-

```
def combinationSum(candidates,target):
    res=[]
    def backtrack(start,path,total):
        if total==target: res.append(path[:]); return
        if total>target: return
        for i in range(start,len(candidates)):
            path.append(candidates[i]); backtrack(i,path,total+candidates[i]); path.p
    backtrack(0,[],0)
    return res
```

Time Complexity :- O(2^t)
Space Complexity: O(t)

Optimized :-

```
def combinationSum(candidates,target):
    candidates.sort()
    res=[]
    def dfs(start,path,total):
        if total==target: res.append(path[:]); return
        for i in range(start,len(candidates)):
            if total+candidates[i]>target: break
            dfs(i,path+[candidates[i]],total+candidates[i])
    dfs(0,[],0)
    return res
```

Time Complexity :- O(2^t)
Space Complexity: O(t)

---

42. Decode Ways

Problem :-

A message containing digits is encoded with 'A'=1, 'B'=2, ..., 'Z'=26. Determine the total number of ways to decode it.

Topic: DP | **Difficulty:** Medium | **Companies:** Google, Spotify, Bloomberg

Brute Force :-

```
def numDecodings(s):
    res=[]
    def dfs(i,path):
        if i==len(s): res.append(path[:]); return
        if i<len(s): dfs(i+1,path+s[i])
        if i+1<len(s) and int(s[i:i+2])<=26: dfs(i+2,path+s[i:i+2])
    dfs(0,'')
    return len(res)
```

Time Complexity :- O(2^n)
Space Complexity: O(n)

Optimized :-

```
def numDecodings(s):
    if not s or s[0]=='0': return 0
    dp=[0]*(len(s)+1); dp[0]=1; dp[1]=1
    for i in range(2,len(s)+1):
        if s[i-1]!='0': dp[i]+=dp[i-1]
        if '10'<=s[i-2:i]<='26': dp[i]+=dp[i-2]
    return dp[-1]
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

43. Word Break

Problem :-

Given a string and a dictionary of words, determine if the string can be segmented into a space-separated sequence of dictionary words.

Topic: DP | **Difficulty:** Medium | **Companies:** Facebook, Airbnb, Stripe

Brute Force :-

```
def wordBreak(s,wordDict):
    res=[]
    def dfs(i):
        if i==len(s): res.append(True); return
        for j in range(i+1,len(s)+1):
            if s[i:j] in wordDict: dfs(j)
    dfs(0)
    return bool(res)
```

Time Complexity :- O(2^n)
Space Complexity: O(n)

Optimized :-

```
def wordBreak(s,wordDict):
    dp=[False]*(len(s)+1); dp[0]=True
    wordSet=set(wordDict)
    for i in range(1,len(s)+1):
        for j in range(i):
            if dp[j] and s[j:i] in wordSet: dp[i]=True; break
    return dp[-1]
```

Time Complexity :- O(n^2)
Space Complexity: O(n)

---

44. Unique Paths

Problem :-

A robot is located at the top-left corner of an m x n grid and wants to reach the bottom-right corner. Find the number of unique paths.

Topic: DP, Combinatorics | **Difficulty:** Medium | **Companies:** Amazon, Google, Pinterest

Brute Force :-

```
def uniquePaths(m,n):
    res=[]
    def dfs(i,j):
        if i==m-1 and j==n-1: res.append(1); return
        if i<m-1: dfs(i+1,j)
        if j<n-1: dfs(i,j+1)
    dfs(0,0)
    return len(res)
```

Time Complexity :- O(2^(m+n))
Space Complexity: O(m+n)

Optimized :-

```
def uniquePaths(m,n):
    dp=[1]*n
    for i in range(1,m):
        dp=[dp[j]+dp[j-1] if j>0 else 1 for j in range(n)]
    return dp[-1]
```

Time Complexity :- O(m*n)
Space Complexity: O(n)

---

45. KMP / String Matching

Problem :-

Implement strStr() — return the index of the first occurrence of needle in haystack, or -1 if not found.

Topic: String Matching | **Difficulty:** Medium | **Companies:** Facebook, Microsoft, Uber

Brute Force :-

```
def strStr(haystack,needle):
    for i in range(len(haystack)-len(needle)+1):
        if haystack[i:i+len(needle)]==needle: return i
    return -1
```

Time Complexity :- O(n*m)
Space Complexity: O(1)

Optimized :-

```
def computeLPS(needle):
    lps=[0]*len(needle); length=0; i=1
    while i<len(needle):
        if needle[i]==needle[length]: length+=1; lps[i]=length; i+=1
        else: length=0 if length==0 else lps[length-1]
    return lps

def strStr(haystack,needle):
    if not needle: return 0
    lps=computeLPS(needle); i=j=0
    while i<len(haystack):
        if haystack[i]==needle[j]: i+=1; j+=1
            if j==len(needle): return i-j
        elif j>0: j=lps[j-1]
        else: i+=1
    return -1
```

Time Complexity :- O(n+m)
Space Complexity: O(m)

---

46. Binary Tree Zigzag Level Order Traversal

Problem :-

Return the zigzag level order traversal of a binary tree's nodes values.

Topic: Tree, BFS | **Difficulty:** Medium | **Companies:** Google, Amazon, Dropbox

Brute Force :-

```
def zigzagLevelOrder(root):
    if not root: return []
    res=[]; level=[root]
    while level:
        res.append([node.val for node in level])
        level=[child for node in level for child in [node.left,node.right] if child]
    return res
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```
def zigzagLevelOrder(root):
    if not root: return []
    res=[]; level=[root]; left_to_right=True
    while level:
        res.append([node.val for node in (level if left_to_right else level[::-1])])
        level=[child for node in level for child in [node.left,node.right] if child]
        left_to_right=not left_to_right
    return res
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

47. Flatten Binary Tree to Linked List

Problem :-

Flatten a binary tree to a linked list in-place following preorder traversal.

Topic: Tree, DFS | **Difficulty:** Medium | **Companies:** Google, Uber, Microsoft

Brute Force :-

```
def flatten(root):
    if not root: return
    nodes=[]
    def preorder(node):
        if not node: return
        nodes.append(node)
        preorder(node.left); preorder(node.right)
    preorder(root)
    for i in range(1,len(nodes)):
        nodes[i-1].left=None
        nodes[i-1].right=nodes[i]
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```
def flatten(root):
    curr=root
    while curr:
        if curr.left:
            pre=curr.left
            while pre.right: pre=pre.right
            pre.right=curr.right
            curr.right=curr.left
            curr.left=None
        curr=curr.right
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

48. Merge Intervals

Problem :-

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals and return an array of non-overlapping intervals.

Topic: Intervals, Sorting | **Difficulty:** Medium | **Companies:** Google, Amazon, Microsoft

Brute Force :-

```
def merge(intervals):
    intervals.sort(key=lambda x: x[0])
    merged = []
    for interval in intervals:
        if not merged or merged[-1][1] < interval[0]:
            merged.append(interval)
        else:
            merged[-1][1] = max(merged[-1][1], interval[1])
    return merged
```

Time Complexity :- O(n log n)
Space Complexity: O(n)

Optimized :-

```
def merge(intervals):
    intervals.sort(key=lambda x: x[0])
    res = [intervals[0]]
    for start, end in intervals[1:]:
        if start <= res[-1][1]:
            res[-1][1] = max(res[-1][1], end)
        else:
            res.append([start, end])
    return res
```

Time Complexity :- O(n log n)
Space Complexity: O(1)

---

49. Insert Interval

Problem :-

You are given an array of non-overlapping intervals sorted by their start times and a new interval. Insert the new interval into the list and merge if necessary.

Topic: Intervals | **Difficulty:** Medium | **Companies:** Google, Amazon, Apple

Brute Force :-

```
def insert(intervals, newInterval):
    intervals.append(newInterval)
    intervals.sort(key=lambda x: x[0])
    res = [intervals[0]]
    for start, end in intervals[1:]:
        if start <= res[-1][1]:
            res[-1][1] = max(res[-1][1], end)
        else:
            res.append([start, end])
    return res
```

Time Complexity :- O(n log n)
Space Complexity: O(n)

Optimized :-

```
def insert(intervals, newInterval):
    res = []
    for i, (start, end) in enumerate(intervals):
        if newInterval[1] < start:
            res.append(newInterval)
            return res + intervals[i:]
        elif newInterval[0] > end:
            res.append([start, end])
        else:
            newInterval = [min(newInterval[0], start), max(newInterval[1], end)]
    res.append(newInterval)
    return res
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

50. Meeting Rooms

Problem :-

Given an array of meeting time intervals consisting of start and end times, determine if a person could attend all meetings.

Topic: Intervals | **Difficulty:** Easy | **Companies:** Google, Amazon, Facebook

Brute Force :-

```
def canAttendMeetings(intervals):
    for i in range(len(intervals)):
        for j in range(i+1, len(intervals)):
            if intervals[i][0] < intervals[j][1] and intervals[j][0] < intervals[i][1
                return False
    return True
```

Time Complexity :- O(n^2)
Space Complexity: O(1)

Optimized :-

```
def canAttendMeetings(intervals):
    intervals.sort(key=lambda x: x[0])
    for i in range(1, len(intervals)):
        if intervals[i][0] < intervals[i-1][1]:
            return False
    return True
```

Time Complexity :- O(n log n)
Space Complexity: O(1)

---

51. Product of Array Except Self

Problem :-

Given an array nums, return an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i]. Solve without division and in O(n).

Topic: Array, Prefix-Suffix | **Difficulty:** Medium | **Companies:** Amazon, Google, Microsoft

Brute Force :-

```
def productExceptSelf(nums):
    n = len(nums)
    res = []
    for i in range(n):
        prod = 1
        for j in range(n):
            if i != j:
                prod *= nums[j]
        res.append(prod)
    return res
```

Time Complexity :- O(n^2)
Space Complexity: O(1)

Optimized :-

```
def productExceptSelf(nums):
    n = len(nums)
    res = [1]*n
    prefix = 1
    for i in range(n):
        res[i] = prefix
        prefix *= nums[i]
    suffix = 1
    for i in range(n-1, -1, -1):
        res[i] *= suffix
        suffix *= nums[i]
    return res
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

52. Top K Frequent Elements

Problem :-

Given an integer array nums and an integer k, return the k most frequent elements.

Topic: Hash, Heap | **Difficulty:** Medium | **Companies:** Amazon, Google, Facebook

Brute Force :-

```
def topKFrequent(nums, k):
    from collections import Counter
    count = Counter(nums)
    return sorted(count.keys(), key=lambda x: count[x], reverse=True)[:k]
```

Time Complexity :- O(n log n)
Space Complexity: O(n)

Optimized :-

```
def topKFrequent(nums, k):
    from collections import Counter
    import heapq
    count = Counter(nums)
    return heapq.nlargest(k, count.keys(), key=count.get)
```

Time Complexity :- O(n log k)
Space Complexity: O(n)

---

53. Valid Anagram

Problem :-

Given two strings s and t, return true if t is an anagram of s, and false otherwise.

Topic: Hash | **Difficulty:** Easy | **Companies:** Amazon, Google, Microsoft

Brute Force :-

```
def isAnagram(s, t):
    return sorted(s) == sorted(t)
```

Time Complexity :- O(n log n)
Space Complexity: O(n)

Optimized :-

```
def isAnagram(s, t):
    from collections import Counter
    return Counter(s) == Counter(t)
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

54. Group Anagrams

Problem :-

Given an array of strings, group the anagrams together. You can return the answer in any order.

Topic: Hash, Sorting | **Difficulty:** Medium | **Companies:** Amazon, Google, Facebook

Brute Force :-

```
def groupAnagrams(strs):
    res = []
    used = [False]*len(strs)
    for i, s1 in enumerate(strs):
        if used[i]: continue
        group = [s1]
        used[i] = True
        for j in range(i+1, len(strs)):
            if sorted(s1) == sorted(strs[j]):
                group.append(strs[j])
                used[j] = True
        res.append(group)
    return res
```

Time Complexity :- O(n * k log k)
Space Complexity: O(n*k)

Optimized :-

```
def groupAnagrams(strs):
    from collections import defaultdict
    res = defaultdict(list)
    for s in strs:
        res[tuple(sorted(s))].append(s)
    return list(res.values())
```

Time Complexity :- O(n * k log k)
Space Complexity: O(n*k)

---

55. Longest Consecutive Sequence

Problem :-

Given an unsorted array of integers, return the length of the longest consecutive elements sequence. Must be O(n).

Topic: Hash | **Difficulty:** Hard | **Companies:** Amazon, Google, Microsoft

Brute Force :-

```
def longestConsecutive(nums):
    if not nums: return 0
    nums = sorted(set(nums))
    max_len = curr = 1
    for i in range(1, len(nums)):
        if nums[i] == nums[i-1]+1:
            curr += 1
        else:
            curr = 1
        max_len = max(max_len, curr)
    return max_len
```

Time Complexity :- O(n log n)
Space Complexity: O(n)

Optimized :-

```
def longestConsecutive(nums):
    nums = set(nums)
    max_len = 0
    for num in nums:
        if num-1 not in nums:
            curr = num
            length = 1
            while curr+1 in nums:
                curr += 1
                length += 1
            max_len = max(max_len, length)
    return max_len
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

56. Word Search

Problem :-

Given a 2D board and a word, find if the word exists in the grid. The word can be constructed from letters of sequentially adjacent cells.

Topic: Backtracking | **Difficulty:** Medium | **Companies:** Google, Amazon, Facebook

Brute Force :-

```
def exist(board, word):
    rows, cols = len(board), len(board[0])
    def dfs(r, c, i):
        if i == len(word): return True
        if r<0 or c<0 or r>=rows or c>=cols or board[r][c] != word[i]: return False
        tmp = board[r][c]; board[r][c] = '#'
        res = dfs(r+1,c,i+1) or dfs(r-1,c,i+1) or dfs(r,c+1,i+1) or dfs(r,c-1,i+1)
        board[r][c] = tmp
        return res
    for r in range(rows):
        for c in range(cols):
            if dfs(r,c,0): return True
    return False
```

Time Complexity :- O(m*n*4^k)
Space Complexity: O(k)

Optimized :-

```
def exist(board, word):
    rows, cols = len(board), len(board[0])
    def dfs(r, c, i):
        if i == len(word): return True
        if not (0<=r<rows and 0<=c<cols) or board[r][c] != word[i]: return False
        board[r][c], tmp = '#', board[r][c]
        res = any(dfs(r+dr, c+dc, i+1) for dr,dc in [(1,0),(-1,0),(0,1),(0,-1)])
        board[r][c] = tmp
        return res
    return any(dfs(r,c,0) for r in range(rows) for c in range(cols))
```

Time Complexity :- O(m*n*4^k)
Space Complexity: O(k)

---

57. Word Ladder

Problem :-

Given beginWord, endWord, and a wordList, return the length of the shortest transformation sequence from beginWord to endWord, changing only one letter at a time and each transformed word must exist in wordList.

Topic: BFS on words | **Difficulty:** Hard | **Companies:** Amazon, Google, Microsoft

Brute Force :-

```
def ladderLength(beginWord, endWord, wordList):
    from itertools import permutations
    # Brute force exponential search
    return 0
```

Time Complexity :- O(n!)
Space Complexity: O(n)

Optimized :-

```
def ladderLength(beginWord, endWord, wordList):
    from collections import deque
    wordSet = set(wordList)
    if endWord not in wordSet: return 0
    q = deque([(beginWord,1)])
    while q:
        word, steps = q.popleft()
        if word == endWord: return steps
        for i in range(len(word)):
            for c in 'abcdefghijklmnopqrstuvwxyz':
                next_word = word[:i]+c+word[i+1:]
                if next_word in wordSet:
                    wordSet.remove(next_word)
                    q.append((next_word, steps+1))
    return 0
```

Time Complexity :- O(n * m * 26)
Space Complexity: O(n)

---

58. Minimum Window Substring

Problem :-

Given strings s and t, return the minimum window in s which will contain all the characters in t. If no such window exists, return empty string.

Topic: Sliding Window | **Difficulty:** Hard | **Companies:** Amazon, Google, Facebook

Brute Force :-

```python
def minWindow(s, t):
    res = ''
    min_len = float('inf')
    for i in range(len(s)):
        for j in range(i+1, len(s)+1):
            window = s[i:j]
            if all(window.count(c) >= t.count(c) for c in t):
                if j-i < min_len: res = window; min_len = j-i
    return res
```

Time Complexity :- O(n^2 * m)
Space Complexity: O(n)

Optimized :-

```python
def minWindow(s, t):
    from collections import Counter
    dict_t = Counter(t)
    required = len(dict_t)
    l,r = 0,0
    formed = 0
    window_counts = {}
    ans = float('inf'), None, None
    while r < len(s):
        c = s[r]
        window_counts[c] = window_counts.get(c,0)+1
        if c in dict_t and window_counts[c]==dict_t[c]: formed +=1
        while l<=r and formed==required:
            c = s[l]
            if r-l+1 < ans[0]: ans = (r-l+1, l, r)
            window_counts[c] -=1
            if c in dict_t and window_counts[c]<dict_t[c]: formed -=1
            l+=1
        r+=1
    return '' if ans[0]==float('inf') else s[ans[1]:ans[2]+1]
```

Time Complexity :- O(n + m)
Space Complexity: O(n + m)

---

59. Longest Substring Without Repeating Characters

Problem :-

Given a string s, find the length of the longest substring without repeating characters.

Topic: Sliding Window | **Difficulty:** Medium | **Companies:** Amazon, Google, Facebook

Brute Force :-

```
def lengthOfLongestSubstring(s):
    max_len = 0
    for i in range(len(s)):
        seen = set()
        for j in range(i, len(s)):
            if s[j] in seen: break
            seen.add(s[j])
            max_len = max(max_len, j-i+1)
    return max_len
```

Time Complexity :- O(n^2)
Space Complexity: O(n)

Optimized :-

```
def lengthOfLongestSubstring(s):
    seen = {}
    l = 0
    max_len = 0
    for r, c in enumerate(s):
        if c in seen and seen[c]>=l: l = seen[c]+1
        seen[c] = r
        max_len = max(max_len, r-l+1)
    return max_len
```

Time Complexity :- O(n)
Space Complexity: O(n)

---

60. Longest Palindromic Substring

Problem :-

Given a string s, return the longest palindromic substring in s.

Topic: DP, Expand Around Center | **Difficulty:** Medium | **Companies:** Amazon, Google, Facebook

Brute Force :-

```
def longestPalindrome(s):
    def isPalindrome(sub):
        return sub == sub[::-1]
    max_len = 0
    res = ''
    for i in range(len(s)):
        for j in range(i, len(s)):
            sub = s[i:j+1]
            if isPalindrome(sub) and len(sub) > max_len:
                res = sub
                max_len = len(sub)
    return res
```

Time Complexity :- O(n^3)
Space Complexity: O(1)

Optimized :-

```
def longestPalindrome(s):
    start = end = 0
    for i in range(len(s)):
        for j in [0,1]:
            l, r = i, i+j
            while l>=0 and r<len(s) and s[l]==s[r]:
                if r-l > end-start:
                    start, end = l, r
                l -=1; r +=1
    return s[start:end+1]
```

Time Complexity :- O(n^2)
Space Complexity: O(1)

---

61. Valid Palindrome

Problem :-

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

Topic: Two Pointers | **Difficulty:** Easy | **Companies:** Amazon, Google, Microsoft

Brute Force :-

```
def isPalindrome(s):
    s = ''.join(c.lower() for c in s if c.isalnum())
    return s == s[::-1]
```

Time Complexity :- O(n)
Space Complexity: O(n)

Optimized :-

```
def isPalindrome(s):
    l, r = 0, len(s)-1
    while l<r:
        while l<r and not s[l].isalnum(): l+=1
        while l<r and not s[r].isalnum(): r-=1
        if s[l].lower() != s[r].lower(): return False
        l+=1; r-=1
    return True
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

62. Palindrome Partitioning

Problem :-

Given a string s, partition s such that every substring of the partition is a palindrome. Return all possible palindrome partitioning of s.

Topic: Backtracking, DP | **Difficulty:** Medium | **Companies:** Amazon, Google, Facebook

Brute Force :-

```
def partition(s):
    res = []
    def isPalindrome(sub):
        return sub == sub[::-1]
    def dfs(start, path):
        if start == len(s): res.append(path[:]); return
        for end in range(start+1, len(s)+1):
            if isPalindrome(s[start:end]): dfs(end, path+[s[start:end]])
    dfs(0,[])
    return res
```

Time Complexity :- O(n*2^n)
Space Complexity: O(n)

Optimized :-

```
def partition(s):
    res = []
    n = len(s)
    dp = [[False]*n for _ in range(n)]
    for i in range(n-1,-1,-1):
        for j in range(i,n):
            dp[i][j] = s[i]==s[j] and (j-i<=2 or dp[i+1][j-1])
    def dfs(start,path):
        if start==n: res.append(path[:]); return
        for end in range(start,n):
            if dp[start][end]: dfs(end+1, path+[s[start:end+1]])
    dfs(0,[])
    return res
```

Time Complexity :- O(n*2^n)
Space Complexity: O(n^2)

63. Merge k Sorted Lists

Problem :-

Merge k sorted linked lists and return it as one sorted list.

Topic: Heap, Divide & Conquer | **Difficulty:** Hard | **Companies:** Amazon, Google, Facebook

Brute Force :-

```
def mergeKLists(lists):
    s = []
    for l in lists:
        while l:
            s.append(l.val)
            l = l.next
    s.sort()
    head = cur = ListNode(0)
    for val in s:
        cur.next = ListNode(val)
        cur = cur.next
    return head.next
```

Time Complexity :- O(N log N)
Space Complexity: O(N)

Optimized :-

```
def mergeKLists(lists):
    import heapq
    heap = []
    for l in lists:
        if l: heapq.heappush(heap,(l.val,l))
    head = cur = ListNode(0)
    while heap:
        val,node = heapq.heappop(heap)
        cur.next = node
        cur = cur.next
        if node.next: heapq.heappush(heap,(node.next.val,node.next))
    return head.next
```

Time Complexity :- O(N log k)
Space Complexity: O(k)

---

64. Kth Largest Element in an Array

Problem :-

Find the kth largest element in an unsorted array.

Topic: Heap, Quickselect | **Difficulty:** Medium | **Companies:** Amazon, Google, Microsoft

Brute Force :-

```
def findKthLargest(nums,k):
    nums.sort()
    return nums[-k]
```

Time Complexity :- O(n log n)
Space Complexity: O(1)

Optimized :-

```
def findKthLargest(nums,k):
    import random
    def partition(left,right,pivot_index):
        pivot=nums[pivot_index]
        nums[pivot_index],nums[right]=nums[right],nums[pivot_index]
        store_index=left
        for i in range(left,right):
            if nums[i]<pivot:
                nums[store_index],nums[i]=nums[i],nums[store_index]
                store_index+=1
        nums[right],nums[store_index]=nums[store_index],nums[right]
        return store_index
    left,right=0,len(nums)-1
    k=len(nums)-k
    while True:
        pivot_index=random.randint(left,right)
        pi=partition(left,right,pivot_index)
        if pi==k: return nums[pi]
        elif pi<k: left=pi+1
        else: right=pi-1
```

Time Complexity :- O(n) average
Space Complexity: O(1)

---

65. Sort Colors

Problem :-

Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, using integers 0,1,2.

Topic: Dutch National Flag, Two Pointers | **Difficulty:** Medium | **Companies:** Amazon, Google, Microsoft

Brute Force :-

```
def sortColors(nums):
    nums.sort()
```

Time Complexity :- O(n log n)
Space Complexity: O(1)

Optimized :-

```
def sortColors(nums):
    p0, curr = 0, 0
    p2 = len(nums)-1
    while curr <= p2:
        if nums[curr]==0: nums[curr],nums[p0]=nums[p0],nums[curr]; p0+=1; curr+=1
        elif nums[curr]==2: nums[curr],nums[p2]=nums[p2],nums[curr]; p2-=1
        else: curr+=1
```

Time Complexity :- O(n)
Space Complexity: O(1)

---

66. Find Peak Element

Problem :-

A peak element is an element that is strictly greater than its neighbors. Find any peak element in O(log n) time.

Topic: Binary Search | **Difficulty:** Medium | **Companies:** Amazon, Google, Microsoft

Brute Force :-

```
def findPeakElement(nums):
    for i in range(len(nums)):
        if (i==0 or nums[i]>nums[i-1]) and (i==len(nums)-1 or nums[i]>nums[i+1]):
            return i
```

Time Complexity :- O(n)
Space Complexity: O(1)

Optimized :-

```
def findPeakElement(nums):
    l,r=0,len(nums)-1
    while l<r:
        m=(l+r)//2
        if nums[m]<nums[m+1]: l=m+1
        else: r=m
    return l
```

Time Complexity :- O(log n)
Space Complexity: O(1)

---

67. Search in Rotated Sorted Array

Problem :-

Search a target in a rotated sorted array and return its index, otherwise -1. O(log n) time.

Topic: Binary Search | **Difficulty:** Medium | **Companies:** Amazon, Google, Microsoft

Brute Force :-

```
def search(nums,target):
    for i,v in enumerate(nums):
        if v==target: return i
    return -1
```

Time Complexity :- O(n)
Space Complexity: O(1)

Optimized :-

```
def search(nums,target):
    l,r=0,len(nums)-1
    while l<=r:
        m=(l+r)//2
        if nums[m]==target: return m
        if nums[l]<=nums[m]:
            if nums[l]<=target<nums[m]: r=m-1
            else: l=m+1
        else:
            if nums[m]<target<=nums[r]: l=m+1
            else: r=m-1
    return -1
```

Time Complexity :- O(log n)
Space Complexity: O(1)

---

68. First Bad Version

Problem :-

Given n versions [1..n] and an API isBadVersion(version), find the first bad version.

Topic: Binary Search | **Difficulty:** Easy | **Companies:** Google, Microsoft, Facebook

Brute Force :-

```
def firstBadVersion(n):
    for i in range(1,n+1):
        if isBadVersion(i): return i
```

Time Complexity :- O(n)
Space Complexity: O(1)

Optimized :-

```
def firstBadVersion(n):
    l,r=1,n
    while l<r:
        m=l+(r-l)//2
        if isBadVersion(m): r=m
        else: l=m+1
    return l
```

Time Complexity :- O(log n)
Space Complexity: O(1)

---

69. Median of Two Sorted Arrays

Problem :-

Given two sorted arrays nums1 and nums2, return the median of the two sorted arrays. Overall run time O(log(min(m,n))).

Topic: Binary Search, Partition | **Difficulty:** Hard | **Companies:** Google, Amazon, Microsoft

Brute Force :-

```
def findMedianSortedArrays(nums1, nums2):
    nums = sorted(nums1+nums2)
    n=len(nums)
    return (nums[(n-1)//2]+nums[n//2])/2
```

Time Complexity :- O((m+n) log (m+n))
Space Complexity: O(m+n)

Optimized :-

```
def findMedianSortedArrays(nums1, nums2):
    A,B=nums1,nums2
    m,n=len(A),len(B)
    if m>n: A,B,m,n=B,A,n,m
    imin,imax,half_len=0,m,(m+n+1)//2
    while imin<=imax:
        i=(imin+imax)//2
        j=half_len-i
        if i<m and B[j-1]>A[i]: imin=i+1
        elif i>0 and A[i-1]>B[j]: imax=i-1
        else:
            max_of_left = max(A[i-1], B[j-1]) if i>0 and j>0 else B[j-1] if i==0 else
            if (m+n)%2==1: return max_of_left
            min_of_right = min(A[i], B[j]) if i<m and j<n else B[j] if i==m else A[i]
            return (max_of_left+min_of_right)/2
```

Time Complexity :- O(log min(m,n))
Space Complexity: O(1)

70. Word Ladder II

Problem :-

Find all shortest transformation sequences from beginWord to endWord using a given word list.

Topic: Graph, BFS + Backtracking | **Difficulty:** Hard | **Companies:** Facebook, Amazon, Coinbase

Brute Force :-

```
def findLadders(beginWord,endWord,wordList):
    res=[]
    def dfs(path,used):
        if path[-1]==endWord: res.append(path[:]); return
        for w in wordList:
            if w not in used and sum(a!=b for a,b in zip(path[-1],w))==1:
                used.add(w); dfs(path+[w],used); used.remove(w)
    dfs([beginWord],set([beginWord]))
    return res
```

Time Complexity :- Exponential
Space Complexity: O(n)

Optimized :-

```
from collections import defaultdict,deque

def findLadders(beginWord,endWord,wordList):
    wordSet=set(wordList); res=[]; layer={beginWord:[[beginWord]]}
    while layer:
        newLayer=defaultdict(list)
        for w,paths in layer.items():
            if w==endWord: res.extend(paths)
            for i in range(len(w)):
                for c in 'abcdefghijklmnopqrstuvwxyz':
                    newW=w[:i]+c+w[i+1:]
                    if newW in wordSet:
                        newLayer[newW]+=[path+[newW] for path in paths]
        layer=newLayer
        wordSet-=set(layer.keys())
    return res
```

Time Complexity :- O(N*L^{2})
Space Complexity: O(N*L)

---

71. Reconstruct Itinerary

Problem :-

Given a list of airline tickets represented by pairs of departure and arrival airports, reconstruct the itinerary in order. Use all tickets once.

Topic: Graph, Hierholzer | **Difficulty:** Medium | **Companies:** Google, LinkedIn, Airbnb

Brute Force :-

```
def findItinerary(tickets):
    tickets.sort(); used=[False]*len(tickets); res=[]
    def dfs(path):
        if len(path)==len(tickets)+1: res.append(path[:]); return True
        for i,(src,dst) in enumerate(tickets):
            if not used[i] and path[-1]==src: used[i]=True; if dfs(path+[dst]): retur
        return False
    dfs([tickets[0][0]])
    return res[0]
```

Time Complexity :- O(N!)
Space Complexity: O(N)

Optimized :-

```
from collections import defaultdict

def findItinerary(tickets):
    graph=defaultdict(list)
    for src,dst in sorted(tickets,reverse=True): graph[src].append(dst)
    res=[]
    def visit(airport):
        while graph[airport]: visit(graph[airport].pop())
        res.append(airport)
    visit('JFK')
    return res[::-1]
```

Time Complexity :- O(E log E)
Space Complexity: O(E)

72. Topological Sort (Generic)

Problem :-

Return a valid topological ordering of a directed acyclic graph.

Topic: Graph, Topological Sort | **Difficulty:** Medium | **Companies:** Microsoft, Facebook, Amazon

Brute Force :-

```
def topoSort(adj):
    visited=set(); res=[]
    def dfs(u):
        visited.add(u)
        for v in adj[u]:
            if v not in visited: dfs(v)
        res.append(u)
    for node in adj:
        if node not in visited: dfs(node)
    return res[::-1]
```

Time Complexity :- O(V+E)
Space Complexity: O(V)

Optimized :-

```
from collections import deque

def topoSort(adj):
    inDegree={u:0 for u in adj}
    for u in adj:
        for v in adj[u]: inDegree[v]+=1
    q=deque([u for u in adj if inDegree[u]==0])
    res=[]
    while q:
        u=q.popleft(); res.append(u)
        for v in adj[u]:
            inDegree[v]-=1
            if inDegree[v]==0: q.append(v)
    return res
```

Time Complexity :- O(V+E)
Space Complexity: O(V)

73. Design Phone Directory / Circular Queue

Problem :-

Design a phone directory with get, check, and release operations, supporting circular reuse of numbers.

Topic: Design, HashSet / Queue | **Difficulty:** Medium | **Companies:** Apple, Google, Uber

Brute Force :-

```
class PhoneDirectory:
    def __init__(self,maxNumbers):
        self.numbers=set(range(maxNumbers))
    def get(self):
        if not self.numbers: return -1
        return self.numbers.pop()
    def check(self,number):
        return number in self.numbers
    def release(self,number):
        self.numbers.add(number)
```

Time Complexity :- O(1)
Space Complexity: O(n)

Optimized :-

```
from collections import deque

class PhoneDirectory:
    def __init__(self,maxNumbers):
        self.available=deque(range(maxNumbers)); self.allocated=set()
    def get(self):
        if not self.available: return -1
        n=self.available.popleft(); self.allocated.add(n); return n
    def check(self,number):
        return number not in self.allocated
    def release(self,number):
        if number in self.allocated: self.allocated.remove(number); self.available.ap
```

Time Complexity :- O(1)
Space Complexity: O(n)

74. Maximum Gap / Skyline

Problem :-

Given an unsorted array, find the maximum difference between successive elements (Maximum Gap), or compute the skyline of buildings.

Topic: Sorting / Heap / Advanced | **Difficulty:** Hard | **Companies:** Bloomberg, Google, Facebook

Brute Force :-

```
def maximumGap(nums):
    if len(nums)<2: return 0
    nums.sort()
    return max(nums[i+1]-nums[i] for i in range(len(nums)-1))
```

Time Complexity :- O(n log n)
Space Complexity: O(1)

Optimized :-

```
def maximumGap(nums):
    if len(nums)<2: return 0
    minNum, maxNum, n = min(nums), max(nums), len(nums)
    bucketSize = max(1,(maxNum-minNum)//(n-1))
    bucketCount = (maxNum-minNum)//bucketSize+1
    buckets=[[float('inf'),float('-inf')] for _ in range(bucketCount)]
    for num in nums:
        idx=(num-minNum)//bucketSize
        buckets[idx][0]=min(buckets[idx][0],num)
        buckets[idx][1]=max(buckets[idx][1],num)
    prev=maxNum; maxGap=0
    for mn,mx in buckets:
        if mn==float('inf'): continue
        maxGap=max(maxGap,mn-prev)
        prev=mx
    return maxGap
```

Time Complexity :- O(n)
Space Complexity: O(n)