**Most common asked interview algorithm in python**

* Here are some of the most commonly asked algorithms in technical interviews, especially for Python developers:

1. **Binary Search**

* **Problem**: Search for an element in a sorted array.
* **Why It's Asked**: Tests understanding of divide-and-conquer strategies and algorithm efficiency.

**Example**

**def binary\_search(arr, target):**

**low, high = 0, len(arr) - 1**

**while low <= high:**

**mid = (low + high) // 2**

**if arr[mid] == target:**

**return mid**

**elif arr[mid] < target:**

**low = mid + 1**

**else:**

**high = mid - 1**

**return -1**

**2. Merge Sort**

* **Problem:** Sort an array.
* **Why It's Asked:** Understanding of sorting algorithms and recursion**.**

**Example**:

**def merge\_sort(arr):**

**if len(arr) <= 1:**

**return arr**

**mid = len(arr) // 2**

**left = merge\_sort(arr[:mid])**

**right = merge\_sort(arr[mid:])**

**return merge(left, right)**

**def merge(left, right):**

**sorted\_list = []**

**i = j = 0**

**while i < len(left) and j < len(right):**

**if left[i] < right[j]:**

**sorted\_list.append(left[i])**

**i += 1**

**else:**

**sorted\_list.append(right[j])**

**j += 1**

**sorted\_list.extend(left[i:])**

**sorted\_list.extend(right[j:])**

**return sorted\_list**

**3. Quick Sort**

* **Problem:** Sort an array efficiently.
* **Why It's Asked:** Tests understanding of in-place sorting and partitioning.

**Example:**

**def quick\_sort(arr):**

**if len(arr) <= 1:**

**return arr**

**pivot = arr[len(arr) // 2]**

**left = [x for x in arr if x < pivot]**

**middle = [x for x in arr if x == pivot]**

**right = [x for x in arr if x > pivot]**

**return quick\_sort(left) + middle + quick\_sort(right)**

**4. Depth-First Search (DFS)**

* **Problem:** Traverse or search through a graph/tree.
* **Why It's Asked:** Tests knowledge of graph algorithms and recursion.

**Example:**

**def dfs(graph, start, visited=None):**

**if visited is None:**

**visited = set()**

**visited.add(start)**

**for neighbor in graph[start]:**

**if neighbor not in visited:**

**dfs(graph, neighbor, visited)**

**return visited**

**5.Breadth-First Search (BFS)**

* **Problem:** Traverse or search through a graph/tree.
* **Why It's Asked:** Tests understanding of queue-based traversal and shortest path algorithms.

**Example:**

**from collections import deque**

**def bfs(graph, start):**

**visited = set()**

**queue = deque([start])**

**while queue:**

**vertex = queue.popleft()**

**if vertex not in visited:**

**visited.add(vertex)**

**queue.extend(graph[vertex] - visited)**

**return visited**

**6.Two Sum Problem**

* **Problem:** Find two numbers in a list that add up to a specific target.
* **Why It's Asked:** Tests problem-solving with hash maps and array manipulation.

**Example:**

**def two\_sum(nums, target):**

**num\_map = {}**

**for i, num in enumerate(nums):**

**complement = target - num**

**if complement in num\_map:**

**return [num\_map[complement], i]**

**num\_map[num] = i**

**return []**

**7. Maximum Subarray (Kadane's Algorithm)**

* **Problem:** Find the contiguous subarray with the maximum sum.
* **Why It's Asked:** Tests dynamic programming and optimization**.**

**Example:**

**def max\_subarray(nums):**

**max\_current = max\_global = nums[0]**

**for i in range(1, len(nums)):**

**max\_current = max(nums[i], max\_current + nums[i])**

**if max\_current > max\_global:**

**max\_global = max\_current**

**return max\_global**

**8. Longest Palindromic Substring**

* **Problem:** Find the longest palindromic substring in a string.
* **Why It's Asked:** Tests dynamic programming or two-pointer approach.

**Example:**

**def longest\_palindromic\_substring(s):**

**def expand\_around\_center(left, right):**

**while left >= 0 and right < len(s) and s[left] == s[right]:**

**left -= 1**

**right += 1**

**return s[left+1:right]**

**if len(s) < 2:**

**return s**

**result = ""**

**for i in range(len(s)):**

**result = max(result, expand\_around\_center(i, i), expand\_around\_center(i, i+1), key=len)**

**return result**

**9. Linked List Cycle Detection**

* **Problem:** Detect if a linked list has a cycle.
* **Why It's Asked:** Tests understanding of linked lists and two-pointer techniques.

**Example:**

**class ListNode:**

**def \_\_init\_\_(self, x):**

**self.val = x**

**self.next = None**

**def has\_cycle(head):**

**slow, fast = head, head**

**while fast and fast.next:**

**slow = slow.next**

**fast = fast.next.next**

**if slow == fast:**

**return True**

**return False**

**10. Fibonacci Sequence**

* **Problem: Generate the nth Fibonacci number.**
* **Why It's Asked: Tests recursion, memoization, and dynamic programming.**

**Example:**

**def fib(n, memo={}):**

**if n in memo:**

**return memo[n]**

**if n <= 1:**

**return n**

**memo[n] = fib(n-1, memo) + fib(n-2, memo)**

**return memo[n]**

**11. Knapsack Problem**

* **Problem:** Given a set of items, each with a weight and value, determine the number of items to include in a collection so that the total weight does not exceed a limit.
* **Why It's Asked**: Tests dynamic programming and optimization.

**Example:**

**def knapsack(weights, values, W):**

**n = len(weights)**

**dp = [[0 for \_ in range(W + 1)] for \_ in range(n + 1)]**

**for i in range(1, n + 1):**

**for w in range(W + 1):**

**if weights[i-1] <= w:**

**dp[i][w] = max(dp[i-1][w], dp[i-1][w-weights[i-1]] + values[i-1])**

**else:**

**dp[i][w] = dp[i-1][w]**

**return dp[n][W]**

**12. Merge Intervals**

* **Problem:** Given a collection of intervals, merge all overlapping intervals.
* **Why It's Asked:** Tests sorting and interval manipulation**.**

**Example:**

**def merge\_intervals(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**