1 General

1.1 Interviewer Considerations

Notes:

- How did the candidate **analyze** the problem?
- Did the candidate miss any special or **edge** cases?
- Did the candidate approach the problem **methodically** and logically?
- Does the candidate have a strong foundation in basic computer science **concepts**?
- Did the candidate produce **working code**? Did the candidate **test** the code?
- Is the candidate's code clean and easy to read and **maintain**?
- Can the candidate **explain** their ideas clearly?

1.2 Steps for Success During the Technical Interview

Summary:

1. **Clarify the question**
   (a) Understand what the question is asking and gather example inputs and outputs.
   (b) Clarify constraints such as:
      i. Can numbers be negative or repeated?
      ii. Are values sorted or do we need to sort them?
      iii. Can we assume input validity?
   (c) Asking clarifying questions shows communication skills and prevents missteps.
2. **Design a solution**
   (a) Avoid immediate coding; propose an initial approach and refine it.
   (b) Analyze the algorithm's time and space complexity.
   (c) Consider and address edge cases.
   (d) Think aloud to demonstrate logical reasoning and collaboration.
   (e) Discuss non-optimal ideas to show your thought process.
3. **Write your code**
   (a) Structure the solution using helper functions.
   (b) Confirm API details when uncertain.
   (c) Use your strongest programming language and full syntax.
   (d) Write complete, working code—not pseudocode.
4. **Test your code**
   (a) Validate your solution with 1–2 example test cases.
   (b) Walk through each line using inputs.
   (c) Do not assume correctness—prove it through testing.
   (d) Discuss any further optimizations and their trade-offs.

1.3 Common Mistakes to Avoid

Warning:

1. Starting to code without clarifying the problem.
2. Failing to write or discuss sample inputs and outputs.
3. Using pseudocode instead of fully functional code.
4. Misunderstanding the problem or optimizing prematurely.

2 Arrays

3 Hashing

3.1 When to Use?

Summary:

-

3.2 Hashing

Algorithm:

3.3 Common Problems

Summary:

Problem	Description:
1. Two Sum	Given an array of integers, return indices of the two numbers s.t. they add up to a specific target.

- **Tricks:**
  - See if target - nums[i] is in the map.
  - If it is, return the index of the target - nums[i] (from prevMap) and i.
    - `prevMap[nums[i]] = i`

4 Two Pointers

4.1 When to Use?

Summary:
- If we need to find a pair of elements that satisfy a condition.
- If we need to find a subarray that satisfies a condition.

4.2 Slow and Fast Pointers

Algorithm:
1.

4.2.1 Common Problems

Summary:

Problem	Description:
15. 3Sum	Given an array of integers, return all the triplets [nums[i], nums[j], nums[k]] s.t. i != j, i != k, and j != k.
• **Tricks:**	
125. Valid Palindrome	Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.
• `s_new = ".join(char.lower() for char in s if char.isalnum())` to remove non-alphanumeric and lowercase.	
• Use front and back pointers. If they not equal, return False. If equal move both pointers.	
167. Two Sum II - Input array is sorted	Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a target.
• Use front and back pointers. If > target, move back pointer left. If < target, move front pointer right.	

4.3 Front and Back Pointers

Algorithm:

1. Initialize two pointers, one at the front and one at the back of the array.

4.3.1 Common Problems

Summary:

Problem	Description:
15. 3Sum	Given an array of integers, return all the triplets [nums[i], nums[j], nums[k]] s.t. i != j, i != k, and j != k.
• **Tricks:**	
125. Valid Palindrome	Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.
• `s_new = ".join(char.lower() for char in s if char.isalnum())` to remove non-alphanumeric and lowercase.	
• Use front and back pointers. If they not equal, return False. If equal move both pointers.	
167. Two Sum II - Input array is sorted	Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a target.
• Use front and back pointers. If > target, move back pointer left. If < target, move front pointer right.	

5 Sliding Window

5.1 Fixed Sliding Window

Summary:
- Find a subarray/substring of a fixed size that satisfies a condition.
- Find the maximum or minimum of a subarray of a fixed size.

Algorithm:

```
initialize window_sum = 0
initialize max_result (or other required value)

# Set up initial window
for i in range(0, k):
    window_sum += arr[i]

max_result = window_sum # Initialize result

# Slide the window
for i in range(k, n):
    window_sum += arr[i] - arr[i - k] # Add new element and remove 1st element of prev window
    max_result = max(max_result, window_sum) (or other computation)

return max_result (or other required value)

```

5.1.1 Common Problems

Summary:

Problem	Description:
643. Maximum Average Subarray I • Follow template.	Given an integer array nums and an integer k, return the maximum average value of a subarray of length k.

5.2 Dynamic Sliding Window

Summary:

- Find longest or shortest subarray/substring that satisfies a condition.

Algorithm:

```
initialize left = 0
initialize window_state (sum, count, frequency map, etc.)
initialize min_or_max_result

for right in range(n):
    update window_state to include arr[right] # Expand the window

    while window_state violates the condition:
        update min_or_max_result (if needed)
        update window_state to exclude arr[left] # Shrink the window
        move left pointer forward

return min_or_max_result
```

5.2.1 Common Problems

Summary:

Problem	Description:
3. Longest Substring Without Repeating Characters	Given a string s, find the length of the longest substring without repeating characters.

- Use a frequency map to track characters in the current window.
- If a character is repeated, move the left pointer to the right of the last occurrence.

6 Binary Search

6.1 When to Use?

Summary:
- If array is sorted.

7 Linked List

Summary: Data structure for storing objects in linear order.
- **Object:** Data and a pointer to the next object.

7.1 When to Use?

Summary:
- Implement other DS: stacks, queues, hash tables.
- Dynamic memory allocation.

7.2 Operations

Summary:

Operation	Time Complexity
Search	$O(n)$
Insert	$O(1)$
Delete	$O(1)$
Access	$O(n)$

7.3 Singly Linked List

Algorithm:

7.4 Doubly Linked List

Algorithm:

7.5 Circular Linked List

Algorithm:

7.6 Common Problems

Summary:

Problem	Description:
206. Reverse Linked List	Given the head of a singly linked list, reverse the list and return the reversed list.

- **Iterative:**
  - Initialize prev, curr = None, head.
  - While loop until curr is None. curr will point to prev, then curr will get updated to a temp that has curr.next and prev will be updated to curr.
  - prev will be the new head.
  - $\underbrace{\text{None}}_{\text{prev}} \rightarrow \underbrace{0}_{\text{curr}} \rightarrow 1 \rightarrow 2$
  - $\underbrace{\text{None}}_{\text{prev}} \leftarrow \underbrace{0}_{\text{curr}} \rightarrow \underbrace{1}_{\text{temp}} \rightarrow 2$
  - $\text{None} \leftarrow \underbrace{0}_{\text{prev=curr}} \rightarrow \underbrace{1}_{\text{curr=temp}} \rightarrow 2$