

# TIP102 | Intermediate Technical Interview Prep

Intermediate Technical Interview Prep Spring 2025 (@ Section 3 | Tuesdays and Thursdays 6PM - 8PM PT)

Personal Member ID#: 117667

## Session 2: Binary Search and Divide and Conquer

---

### Session Overview

This session covers binary search, recursion, and divide-and-conquer techniques to solve problems efficiently. Tasks include finding cruise lengths, locating cabins, counting checked-in passengers, and determining profitability of excursions. Other challenges involve finding the shallowest route point, conducting a treasure hunt in a grid, and solving music-related problems, all emphasizing optimal performance with logarithmic time complexity.

You can find all resources from today including session slide decks, session recordings, and more on the resources tab



### Part 1 : Instructor Led Session

---

We'll spend the first portion of the synchronous class time in large groups, where the instructor will lead class instruction for 30-45 minutes.



### Part 2: Breakout Session

---

In breakout sessions, we will explore and collaboratively solve problem sets in small groups. Here, the **collaboration, conversation, and approach** are just as important as “solving the problem” - please engage warmly, clearly, and plentifully in the process!

In breakout rooms you will:

- Screen-share the problem/s, and verbally review them together
- Screen-share an interactive coding environment, and talk through the steps of a solution approach
  - ProTip: - An Integrated Development Environment (IDE) is a fancy name for a tool you could use for shared writing of code - like Replit.com, Collabed.it, CodePen.io, or other - your staff

team will specify which tool to use for this class!

- Screen-share an implementation of your proposed solution
- Independently follow-along, or create an implementation, in your own IDE.

Your program leader/s will indicate which code sharing tool/s to use as a group, and will help break down or provide specific scaffolding with the main concepts above.

#### ► Note on Expectations

## Problem Solving Approach

To build a long-term organized approach to problem solving, we'll start with three main steps. We'll refer to them as **UPI: Understand, Plan, and Implement**.

We'll apply these three steps to most of the problems we'll see in the first half of the course.

We will learn to:

- **Understand** the problem,
- **Plan** a solution step-by-step, and
- **Implement** the solution

#### ► Comment on UPI

#### ► UPI Example

## Breakout Problems Session 2

---

### ▼ Standard Problem Set Version 1

### Problem 1: Finding the Perfect Cruise

It's vacation time! Given an integer `vacation_length` and a list of integers `cruise_lengths` sorted in ascending order, use binary search to return `True` if there is a cruise length that matches `vacation_length` and `False` otherwise.

```
def find_cruise_length(cruise_lengths, vacation_length):  
    pass
```

Example Usage:

```
print(find_cruise_length([9, 10, 11, 12, 13, 14, 15], 13))  
  
print(find_cruise_length([8, 9, 12, 13, 13, 14, 15], 11))
```

Example Output:

```
True
False
```

►  **Hint: Binary Search**

## Problem 2: Booking the Perfect Cruise Cabin

As part of your cruise planning, you have a list of available cabins sorted in ascending order by their deck level. Given the list of available cabins represented by deck level, `cabins`, and an integer `preferred_deck`, write a **recursive** function `find_cabin_index()` that returns the index of `preferred_deck`. If a cabin with your `preferred_deck` does not exist in `cabins`, return the index where it would be if it were added to the list to maintain the sorted order.

Your algorithm must have  $O(\log n)$  time complexity.

```
def find_cabin_index(cabins, preferred_deck):
    pass
```


Example Usage:

```
print(find_cabin_index([1, 3, 5, 6], 5))
print(find_cabin_index([1, 3, 5, 6], 2))
print(find_cabin_index([1, 3, 5, 6], 7))
```

Example Output:

```
2
1
4
```

►  **Hint: Recursion**

►  **Hint:**  $O(\log n)$  **Time Complexity**

## Problem 3: Count Checked In Passengers

As a cruise ship worker, you're in charge of tracking how many passengers have checked in to their rooms thus far. You are given a list of `rooms` where passengers are either checked in (represented by a `1`) or not checked in (represented by a `0`). The list is sorted, so all the `0`s appear before any `1`s.

Write a function `count_checked_in_passengers()` that efficiently counts and returns the total number of checked-in passengers ( $O(1)$ s) in the list in  $O(\log n)$  time.

```
def count_checked_in_passengers(rooms):  
    pass
```

Example Usage:

```
rooms1 = [0, 0, 0, 1, 1, 1, 1]  
rooms2 = [0, 0, 0, 0, 0, 0, 1]  
rooms3 = [0, 0, 0, 0, 0, 0, 0]  
  
print(count_checked_in_passengers(rooms1))  
print(count_checked_in_passengers(rooms2))  
print(count_checked_in_passengers(rooms3))
```

Example Output:

```
4  
1  
0
```

## Problem 4: Determining Profitability of Excursions

As the activities director on a cruise ship, you're organizing excursions for the passengers. You have a sorted list of non-negative integers `excursion_counts`, where each number represents the number of passengers who have signed up for various excursions at your next cruise destination. The list is considered **profitable** if there exists a number `x` such that there are **exactly** `x` excursions that have **at least** `x` passengers signed up.

Write a function that determines whether `excursion_counts` is profitable. If it is profitable, return the value of `x`. If it is not profitable, return `-1`. It can be proven that if `excursion_counts` is profitable, the value for `x` is unique.

Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

```
def is_profitable(excursion_counts):  
    pass
```

Example Usage:

```
print(is_profitable([3, 5]))  
print(is_profitable([0, 0]))
```

Example Output:

2

Example 1 Explanation: There are 2 values (3 and 5) that are greater than or equal to

-1

Example 2 Explanation: No numbers fit the criteria for x.

- If  $x = 0$ , there should be 0 numbers  $\geq x$ , but there are 2.
- If  $x = 1$ , there should be 1 number  $\geq x$ , but there are 0.
- If  $x = 2$ , there should be 2 numbers  $\geq x$ , but there are 0.
- $x$  cannot be greater since there are only 2 numbers in nums.

## Problem 5: Finding the Shallowest Point

As the captain of the cruise ship, you need to take a detour to steer clear of an incoming storm.

Given an array of integers `depths` representing the varying water depths along your potential new route, write a function `find_shallowest_point()` to help you decide whether the new route is deep enough for your ship. The function should use a divide-and-conquer approach to return the shallowest point (minimum value) in `depths`. You may not use the built-in `min()` function.

Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

```
def find_shallowest_point(arr):  
    pass
```

Example Usage:

```
print(find_shallowest_point([5, 7, 2, 8, 3]))  
print(find_shallowest_point([12, 15, 10, 21]))
```

Example Output:

```
2  
10
```

►  **Hint: Divide and Conquer**

## Problem 6: Cruise Ship Treasure Hunt

As a fun game, the cruise ship director has organized a treasure hunt for the kids on board and hidden a chest of candy in one of the rooms on board. The rooms are organized in a `m x n` grid, where each row and each column are sorted in ascending order by room number. Given an integer representing the room number where the prize is hidden `treasure`, use a divide and conquer approach to return a tuple in the form `(row, col)` representing the row and column indices where `treasure` was found. If `treasure` is not in the matrix, return `(-1, -1)`.

Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

```
def find_treasure(matrix, treasure):  
    pass
```

Example Usage:

```
rooms = [  
    [1, 4, 7, 11],  
    [8, 9, 10, 20],  
    [11, 12, 17, 30],  
    [18, 21, 23, 40]  
]  
  
print(find_treasure(rooms, 17))  
print(find_treasure(rooms, 5))
```

Example Output:

```
(2, 2)  
(-1, -1)
```

[Close Section](#)

## ▼ Standard Problem Set Version 2

### Problem 1: Finding the Perfect Song

Abby Lee of Dance Moms is looking for the perfect song to choreograph a group routine to and needs a song of a specified length. Given a specific song length `length` and a list of song lengths `playlist` sorted in ascending order, use the binary search algorithm to return the index of the song in `playlist` with `length`. If no song with the target `length` exists, return `-1`.

```
def find_perfect_song(playlist, length):  
    pass
```

Example Usage:

```
print(find_perfect_song([101, 102, 103, 104, 105], 103))  
print(find_perfect_song([201, 202, 203, 204, 205], 206))
```

```
2  
-1
```

► 💡 **Hint: Binary Search**

## Problem 2: Finding Tour Dates

Your favorite artist is doing a short residency in your city and you're hoping to attend one of their concerts! But because of school, you're only free one day this month 😞. Given a sorted list of integers `tour_dates` representing the days this month your favorite artist is playing, and an integer `available` representing the day you are available, write a **recursive** function `can_attend()` that returns `True` if you will be able to attend one of their concerts (some date in `tour_dates` matches `available`) and `False` otherwise.

Your solution must have `O(log n)` time complexity.

```
def can_attend(tour_dates, available):  
    pass
```

Example Usage:

```
print(can_attend([1, 3, 7, 10, 12], 12))  
print(can_attend([1, 3, 7, 10, 12], 5))
```

Example Output:

```
True  
False
```

► 💡 **Hint: Recursive Helpers**

► 💡 **Hint: `O(log n)` Time Complexity**

## Problem 3: Sqrt(x)

Given a non-negative integer `x`, use binary search to return the square root of `x` rounded down to the nearest integer. The returned integer should be non-negative as well.

You may not use any built-in exponent function or operator. You may not use any external libraries like `math`.

- For example, do not use `pow(x, 0.5)` or `x ** 0.5`.

Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

```
def my_sqrt(x):  
    pass
```

Example Usage:

```
print(my_sqrt(4))
print(my_sqrt(8))
```

Example Output:

```
2
4
```

Example 2 Explanation: The square root of 8 is 2.82842..., and since we round it down to the nearest integer, the answer is 2.

## Problem 4: Granting Backstage Access

You're helping manage a music tour, and you have an array of integers `group_sizes` where each element represents a group of friends attending tonight's concert together. The artist has time to meet two sets of fans backstage before the show. You want to choose two groups such that the combined number of people is the highest possible while still strictly below a threshold `room_capacity`. Given the list `group_sizes` and integer `room_capacity`, use binary search to return the maximum sum of two distinct groups in `group_sizes` where the sum is less than `room_capacity`. If no such pair exists, return `-1`.

Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

```
def get_group_sum(group_sizes, room_capacity):
    pass
```

Example Usage:

```
print(get_group_sum([1,20,10,14,3,5,4,2], 12))
print(get_group_sum([10,20,30], 15))
```

Example Output:

```
11
```

Example 1 Explanation: We can use 1 and 10 to sum 11 which is less than 12

```
-1
```

Example 2 Explanation: In this case it is not possible to get a pair sum less than 15

## Problem 5: Harmonizing Two Musical Tracks

You're working as a music producer and have two tracks `track1` and `track2`, each represented by a sorted list of pitch values. Using the divide-and-conquer approach, merge the pitch values into a single, sorted sequence and return the resulting list.

Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.



```
def merged_tracks(track1, track2):  
    pass
```

Example Usage:

```
track1 = [1, 3, 5]  
track2 = [2, 4, 6]  
track3 = [10, 20]  
track4 = [15, 30]  
  
print(merged_tracks(track1, track2))  
print(merged_tracks(track3, track4))
```

Example Output:

```
[1, 2, 3, 4, 5, 6]  
[10, 15, 20, 30]
```

► 💡 **Hint: Divide and Conquer**

## Problem 6: Merge Sort Playlist

Given a list of strings `playlist`, use merge sort to write a recursive `merge_sort_playlist()` function that accepts that returns the list of songs sorted in alphabetical order.

Pseudocode has been provided for you

```
def merge_sort_helper(left_arr, right_arr):  
    # Create an empty list to store merged result list  
    # Use pointers to iterate through left_arr and right_arr  
        # Compare their elements, and add the smaller element to result list  
        # Increment pointer of list with smaller element  
    # Add any remaining elements from the left half  
    # Add any remaining elements from the right half  
    # Return the merged list  
    pass  
  
def merge_sort_playlist(playlist):  
    # Base Case:  
    # If the list has 1 or 0 elements, it's already sorted  
  
    # Recursive Cases:  
    # Divide the list into two halves  
    # Merge sort first half  
    # Merge sort second half  
    # Use the recursive helper to merge the sorted halves (pass in sorted left half,  
    # Return the merged list  
    pass
```

Example Usage:

```
print(merge_sort_playlist(["Formation", "Crazy in Love", "Halo"]))  
print(merge_sort_playlist(["Single Ladies", "Love on Top", "Irreplaceable"]))
```

Example Output:

```
['Crazy in Love', 'Formation', 'Halo']  
['Irreplaceable', 'Love on Top', 'Single Ladies']
```

►  **Hint: Merge Sort**

Close Section

- **Advanced Problem Set Version 1**
- **Advanced Problem Set Version 2**