TIP102 | Intermediate Technical Interview Prep

Intermediate Technical Interview Prep Spring 2025 (@ Section 3 | Tuesdays and Thursdays 6PM - 8PM PT)
Personal Member ID#: 117667

👋 **IMPORTANT:** This session will take place on **Thursday, March 13th at 6:00PM PDT**.

Session 1: Review

Session Overview

In this unit, we will transition from the UPI method to the full UMPIRE method. Students will review content from Units 1-3 by matching each problem to a data structure and/or strategy introduced in previous units before solving. Students will also practice evaluating the time and space complexity of each of problem. Problems will cover strings, arrays, hash tables (dictionaries), stacks, queues, and the two pointer technique.

> 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 1

▶ **Standard Problem Set Version 1**

▶ **Standard Problem Set Version 2**

▼ **Advanced Problem Set Version 1**

Problem 1: Brand Filter

You're tasked with filtering out brands that are not sustainable from a list of fashion brands. A sustainable brand is defined as one that meets a specific criterion, such as using eco-friendly materials, ethical labor practices, or being carbon-neutral.

Write the `filter_sustainable_brands()` function, which takes a list of brands and a criterion, then returns a list of brands that meet the criterion.

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 filter_sustainable_brands(brands, criterion):
    pass
```

Example Usage:

```
brands = [
    {"name": "EcoWear", "criteria": ["eco-friendly", "ethical labor"]},
    {"name": "FastFashion", "criteria": ["cheap materials", "fast production"]},
    {"name": "GreenThreads", "criteria": ["eco-friendly", "carbon-neutral"]},
    {"name": "TrendyStyle", "criteria": ["trendy designs"]}
]

brands_2 = [
    {"name": "Earthly", "criteria": ["ethical labor", "fair wages"]},
    {"name": "FastStyle", "criteria": ["mass production"]},
    {"name": "NatureWear", "criteria": ["eco-friendly"]},
    {"name": "GreenFit", "criteria": ["recycled materials", "eco-friendly"]}
]

brands_3 = [
    {"name": "OrganicThreads", "criteria": ["organic cotton", "fair trade"]},
    {"name": "GreenLife", "criteria": ["recycled materials", "carbon-neutral"]},
    {"name": "FastCloth", "criteria": ["cheap production"]}
]

print(filter_sustainable_brands(brands, "eco-friendly"))
print(filter_sustainable_brands(brands_2, "ethical labor"))
print(filter_sustainable_brands(brands_3, "carbon-neutral"))
```

Example Output:

```
['EcoWear', 'GreenThreads']
['Earthly']
['GreenLife']
```

▶ 💡 **Hint: Big O (Time & Space Complexity)**

Problem 2: Eco-Friendly Materials

Certain materials are recognized as eco-friendly due to their low environmental impact. You need to track which materials are used by various brands and count how many times each material appears across all brands. This will help identify the most commonly used eco-friendly materials.

Write the `count_material_usage()` function, which takes a list of brands (each with a list of materials) and returns the material names and the number of times each material appears across all brands.

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 count_material_usage(brands):
    pass
```

Example Usage:

```
brands = [
    {"name": "EcoWear", "materials": ["organic cotton", "recycled polyester"]},
    {"name": "GreenThreads", "materials": ["organic cotton", "bamboo"]},
    {"name": "SustainableStyle", "materials": ["bamboo", "recycled polyester"]}
]

brands_2 = [
    {"name": "NatureWear", "materials": ["hemp", "linen"]},
    {"name": "Earthly", "materials": ["organic cotton", "hemp"]},
    {"name": "GreenFit", "materials": ["linen", "recycled wool"]}
]

brands_3 = [
    {"name": "OrganicThreads", "materials": ["organic cotton"]},
    {"name": "EcoFashion", "materials": ["recycled polyester", "hemp"]},
    {"name": "GreenLife", "materials": ["recycled polyester", "bamboo"]}
]

print(count_material_usage(brands))
print(count_material_usage(brands_2))
print(count_material_usage(brands_3))
```

Example Output:

```
{'organic cotton': 2, 'recycled polyester': 2, 'bamboo': 2}
{'hemp': 2, 'linen': 2, 'organic cotton': 1, 'recycled wool': 1}
{'organic cotton': 1, 'recycled polyester': 2, 'hemp': 1, 'bamboo': 1}
```

Problem 3: Fashion Trends

In the fast-changing world of fashion, certain materials and practices become trending based on how frequently they are adopted by brands. You want to identify which materials and practices are trending. A material or practice is considered "trending" if it appears in the dataset more than once.

Write the `find_trending_materials()` function, which takes a list of brands (each with a list of materials or practices) and returns a list of materials or practices that are trending (i.e., those that appear more than once across all brands).

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_trending_materials(brands):
    pass
```

Example Usage:

```
brands = [
    {"name": "EcoWear", "materials": ["organic cotton", "recycled polyester"]},
    {"name": "GreenThreads", "materials": ["organic cotton", "bamboo"]},
    {"name": "SustainableStyle", "materials": ["bamboo", "recycled polyester"]}
]

brands_2 = [
    {"name": "NatureWear", "materials": ["hemp", "linen"]},
    {"name": "Earthly", "materials": ["organic cotton", "hemp"]},
    {"name": "GreenFit", "materials": ["linen", "recycled wool"]}
]

brands_3 = [
    {"name": "OrganicThreads", "materials": ["organic cotton"]},
    {"name": "EcoFashion", "materials": ["recycled polyester", "hemp"]},
    {"name": "GreenLife", "materials": ["recycled polyester", "bamboo"]}
]

print(find_trending_materials(brands))
print(find_trending_materials(brands_2))
print(find_trending_materials(brands_3))
```

Example Output:

```
['organic cotton', 'recycled polyester', 'bamboo']
['hemp', 'linen']
['recycled polyester']
```

Problem 4: Fabric Pairing

You want to find pairs of fabrics that, when combined, maximize eco-friendliness while staying within a budget. Each fabric has a cost associated with it, and your goal is to identify the pair of fabrics whose combined cost is the highest possible without exceeding the budget.

Write the `find_best_fabric_pair()` function, which takes a list of fabrics (each with a name and cost) and a budget. The function should return the names of the two fabrics whose combined cost is the closest to the budget without exceeding it.

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_best_fabric_pair(fabrics, budget):
    pass
```

Example Usage:

```
fabrics = [("Organic Cotton", 30), ("Recycled Polyester", 20), ("Bamboo", 25), ("Hem
fabrics_2 = [("Linen", 50), ("Recycled Wool", 40), ("Tencel", 30), ("Organic Cotton"
fabrics_3 = [("Linen", 40), ("Hemp", 35), ("Recycled Polyester", 25), ("Bamboo", 20)

print(find_best_fabric_pair(fabrics, 45))
print(find_best_fabric_pair(fabrics_2, 70))
print(find_best_fabric_pair(fabrics_3, 60))
```

Example Output:

```
('Hemp', 'Organic Cotton')
('Tencel', 'Recycled Wool')
('Bamboo', 'Linen')
```

Problem 5: Fabric Stacks

You need to organize rolls of fabric in such a way that you can efficiently retrieve them based on their eco-friendliness rating. Fabrics are stacked one on top of the other, and you can only retrieve the top fabric in the stack.

Write the `organize_fabrics()` function, which takes a list of fabrics (each with a name and an eco-friendliness rating) and returns a list of fabric names in the order they would be retrieved from the stack, starting with the least eco-friendly fabric.

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 organize_fabrics(fabrics):
    pass
```

Example Usage:

```
fabrics = [("Organic Cotton", 8), ("Recycled Polyester", 6), ("Bamboo", 7), ("Hemp",
fabrics_2 = [("Linen", 5), ("Recycled Wool", 9), ("Tencel", 7), ("Organic Cotton", 6
fabrics_3 = [("Linen", 4), ("Hemp", 8), ("Recycled Polyester", 5), ("Bamboo", 7)]

print(organize_fabrics(fabrics))
print(organize_fabrics(fabrics_2))
print(organize_fabrics(fabrics_3))
```

Example Output:

```
['Hemp', 'Organic Cotton', 'Bamboo', 'Recycled Polyester']
['Recycled Wool', 'Tencel', 'Organic Cotton', 'Linen']
['Hemp', 'Bamboo', 'Recycled Polyester', 'Linen']
```

Problem 6: Supply Chain

In the sustainable fashion industry, managing the supply chain efficiently is crucial. Supplies arrive in a sequence, and you need to process them in the order they arrive. However, some supplies may be of higher priority due to their eco-friendliness or scarcity.

Write the `process_supplies()` function, which takes a list of supplies (each with a name and a priority level) and returns a list of supply names in the order they would be processed, with higher priority supplies processed first.

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 process_supplies(supplies):
    pass
```

Example Usage:

```
supplies = [("Organic Cotton", 3), ("Recycled Polyester", 2), ("Bamboo", 4), ("Hemp"
supplies_2 = [("Linen", 2), ("Recycled Wool", 5), ("Tencel", 3), ("Organic Cotton", 
supplies_3 = [("Linen", 3), ("Hemp", 2), ("Recycled Polyester", 5), ("Bamboo", 1)]

print(process_supplies(supplies))
print(process_supplies(supplies_2))
print(process_supplies(supplies_3))
```

Example Output:

```
['Bamboo', 'Organic Cotton', 'Recycled Polyester', 'Hemp']
['Recycled Wool', 'Organic Cotton', 'Tencel', 'Linen']
['Recycled Polyester', 'Linen', 'Hemp', 'Bamboo']
```

Problem 7: Calculate Fabric Waste

In the sustainable fashion industry, minimizing waste is crucial. After cutting out patterns for clothing items, there are often leftover pieces of fabric that cannot be used. Your task is to calculate the total amount of fabric waste generated after producing a collection of clothing items. Each clothing item requires a certain amount of fabric, and the available fabric rolls come in fixed lengths.

Write the `calculate_fabric_waste()` function, which takes a list of clothing items (each with a required fabric length) and a list of fabric rolls (each with a specific length). The function should return the total fabric waste after producing all the items.

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 calculate_fabric_waste(items, fabric_rolls):
    pass
```

Example Usage:

```
items = [("T-Shirt", 2), ("Pants", 3), ("Jacket", 5)]
fabric_rolls = [5, 5, 5]

items_2 = [("Dress", 4), ("Skirt", 3), ("Blouse", 2)]
fabric_rolls = [4, 4, 4]

items_3 = [("Jacket", 6), ("Shirt", 2), ("Shorts", 3)]
fabric_rolls = [7, 5, 5]

print(calculate_fabric_waste(items, fabric_rolls))
print(calculate_fabric_waste(items_2, fabric_rolls))
print(calculate_fabric_waste(items_3, fabric_rolls))
```

Example Output:

```
5
3
6
```

Problem 8: Fabric Roll Organizer

You need to organize fabric rolls for optimal usage. Each fabric roll has a specific length, and you want to group them into pairs so that the difference between the lengths of the rolls in each pair is minimized. If there's an odd number of rolls, one roll will be left out.

Write the `organize_fabric_rolls()` function, which takes a list of fabric roll lengths and returns a pair of fabric roll lengths, where the difference in lengths between the rolls is minimized. If there's an odd number of rolls, the last roll should be returned separately.

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 organize_fabric_rolls(fabric_rolls):
    pass
```

Example Usage:

```
fabric_rolls = [15, 10, 25, 30, 22]
fabric_rolls_2 = [5, 8, 10, 7, 12, 14]
fabric_rolls_3 = [40, 10, 25, 15, 30]

print(organize_fabric_rolls(fabric_rolls))
print(organize_fabric_rolls(fabric_rolls_2))
print(organize_fabric_rolls(fabric_rolls_3))
```

Example Output:

```
[(10, 15), (22, 25), 30]
[(5, 7), (8, 10), (12, 14)]
[(10, 15), (25, 30), 40]
```

▾ Advanced Problem Set Version 2

Problem 1: Track Screen Time Usage

In the digital age, managing screen time is crucial for maintaining a healthy balance between online and offline activities. You need to track how much time users spend on different apps throughout the day.

Write the `track_screen_time()` function, which takes a list of logs, where each log contains an app name and the number of minutes spent on that app during a specific hour. The function should return the app names and the total time spent on each app.

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 track_screen_time(logs):
    pass
```

Example Usage:

```
logs = [("Instagram", 30), ("YouTube", 20), ("Instagram", 25), ("Snapchat", 15), ("Y
logs_2 = [("Twitter", 10), ("Reddit", 20), ("Twitter", 15), ("Instagram", 35)]
logs_3 = [("TikTok", 40), ("TikTok", 50), ("YouTube", 60), ("Snapchat", 25)]

print(track_screen_time(logs))
print(track_screen_time(logs_2))
print(track_screen_time(logs_3))
```

Example Output:

```
{'Instagram': 55, 'YouTube': 30, 'Snapchat': 15}
{'Twitter': 25, 'Reddit': 20, 'Instagram': 35}
{'TikTok': 90, 'YouTube': 60, 'Snapchat': 25}
```

▶ 💡 **Hint: Big O (Time & Space Complexity)**

Problem 2: Identify Most Used Apps

You want to help users identify which apps they spend the most time on throughout the day. Based on the screen time logs, your task is to find the app with the highest total screen time.

Write the `most_used_app()` function, which takes a dictionary containing the app names and the total time spent on each app. The function should return the app with the highest screen time. If multiple apps have the same highest screen time, return any one of them.

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 most_used_app(screen_time):
    pass
```

Example Usage:

```
screen_time = {"Instagram": 55, "YouTube": 30, "Snapchat": 15}
screen_time_2 = {"Twitter": 25, "Reddit": 20, "Instagram": 35}
screen_time_3 = {"TikTok": 90, "YouTube": 90, "Snapchat": 25}

print(most_used_app(screen_time))
print(most_used_app(screen_time_2))
print(most_used_app(screen_time_3))
```

Example Output:

```
Instagram
Instagram
TikTok
```

Problem 3: Weekly App Usage

Users want to know how much time they are spending on each app over the course of a week. Your task is to summarize the total weekly usage for each app and then identify the app with the most varied usage pattern throughout the week. The varied usage pattern can be measured by the difference between the maximum and minimum daily usage for each app.

Write the `most_varied_app()` function, which takes a dictionary containing the app names and daily usage over seven days. The function should return the app with the highest difference between the maximum and minimum usage over the week. If multiple apps have the same difference, return any one of them.

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 most_varied_app(app_usage):
    pass
```

Example Usage:

```
app_usage = {
    "Instagram": [60, 55, 65, 60, 70, 55, 60],
    "YouTube": [100, 120, 110, 100, 115, 105, 120],
    "Snapchat": [30, 35, 25, 30, 40, 35, 30]
}

app_usage_2 = {
    "Twitter": [15, 15, 15, 15, 15, 15, 15],
    "Reddit": [45, 50, 55, 50, 45, 50, 55],
    "Facebook": [80, 85, 80, 85, 80, 85, 80]
}

app_usage_3 = {
    "TikTok": [80, 100, 90, 85, 95, 105, 90],
    "Spotify": [40, 45, 50, 45, 40, 45, 50],
    "WhatsApp": [60, 60, 60, 60, 60, 60, 60]
}

print(most_varied_app(app_usage))
print(most_varied_app(app_usage_2))
print(most_varied_app(app_usage_3))
```

Example Output:

```
YouTube
Reddit
TikTok
```

Problem 4: Daily App Usage Peaks

You want to help users identify the peak hours of their app usage during the day. Users log their app usage every hour, and your task is to determine the highest total screen time recorded during any three consecutive hours.

Write the `peak_usage_hours()` function that takes a list of 24 integers, where each integer represents the number of minutes spent on apps during a specific hour (from hour 0 to hour 23). The function should return the start hour and the total screen time for the three-hour period with the highest total usage. If multiple periods have the same total, return the earliest one.

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 peak_usage_hours(screen_time):
  pass
```

Example Usage:

```
screen_time = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160
screen_time_2 = [5, 15, 10, 20, 30, 25, 50, 40, 35, 45, 60, 55, 65, 75, 70, 85, 95, 
screen_time_3 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

print(peak_usage_hours(screen_time))
print(peak_usage_hours(screen_time_2))
print(peak_usage_hours(screen_time_3))
```

Example Output:

```
(21, 690)
(21, 360)
(0, 0)
```

Problem 5: App Usage Pattern Recognition

Users want to identify patterns in their app usage over the course of a day. Specifically, they are interested in finding out if they have periods of repetitive behavior, where they switch between the same set of apps in a recurring pattern. Your task is to detect the longest repeating pattern of app usage within a 24-hour period.

Write the `find_longest_repeating_pattern()` function, which takes a list of app usage logs, where each element in the list represents the app used in a particular hour (from hour 0 to hour 23). The function should return the longest repeating pattern of apps and how many times the pattern repeats. If there are multiple patterns of the same length, return the first one found.

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_longest_repeating_pattern(app_logs):
  pass
```

Example Usage:

```
app_logs = ["Instagram", "YouTube", "Snapchat", "Instagram", "YouTube", "Snapchat", 
app_logs_2 = ["Facebook", "Instagram", "Facebook", "Instagram", "Facebook", "Instagr
app_logs_3 = ["WhatsApp", "TikTok", "Instagram", "YouTube", "Snapchat", "Twitter", "

print(find_longest_repeating_pattern(app_logs))
print(find_longest_repeating_pattern(app_logs_2))
print(find_longest_repeating_pattern(app_logs_3))
```

Example Output:

```
(['Instagram', 'YouTube', 'Snapchat'], 3)
(['Facebook', 'Instagram'], 3)
(['WhatsApp', 'TikTok', 'Instagram', 'YouTube', 'Snapchat', 'Twitter', 'Facebook'],
```

Problem 6: Screen Time Session Management

As part of a digital wellbeing initiative, you're designing a system to manage screen time sessions on a device. The device tracks various apps being opened and closed throughout the day. Each app opening starts a new session, and each closing ends that session. The system should ensure that for every app opened, there is a corresponding closure.

Write the `manage_screen_time_sessions()` function, which takes a list of actions representing app openings and closures throughout the day. Each action is either `"OPEN <app>"` or `"CLOSE <app>"`. The function should return `True` if all the app sessions are properly managed (i.e., every opened app has a corresponding close in the correct order), and `False` otherwise.

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 manage_screen_time_sessions(actions):
  pass
```

Example Usage:

```
actions = ["OPEN Instagram", "OPEN Facebook", "CLOSE Facebook", "CLOSE Instagram"]
actions_2 = ["OPEN Instagram", "CLOSE Instagram", "CLOSE Facebook"]
actions_3 = ["OPEN Instagram", "OPEN Facebook", "CLOSE Instagram", "CLOSE Facebook"]

print(manage_screen_time_sessions(actions))
print(manage_screen_time_sessions(actions_2))
print(manage_screen_time_sessions(actions_3))
```

Example Output:

```
True
False
False
```

Problem 7: Digital Wellbeing Dashboard Analysis

You're building a digital wellbeing dashboard that tracks users' daily app usage and helps them identify patterns and areas for improvement. Each user has a log of their daily app usage, which includes various activities like Social Media, Entertainment, Productivity, and so on. The goal is to analyze this data to provide insights into their usage patterns.

Write the `analyze_weekly_usage()` function, which takes a dictionary where each key is a day of the week (e.g., `"Monday"`, `"Tuesday"`) and the value is another dictionary. This nested dictionary's keys represent app categories (e.g., `"Social Media"`, `"Entertainment"`) and its values represent the time spent (in minutes) on that category during that day.

Your function should return:

1. The total time spent on each category across the entire week.

2. The day with the highest total usage.
3. The most-used category of the week.

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 analyze_weekly_usage(weekly_usage):
  pass
```

Example Usage:

```
weekly_usage = {
    "Monday": {"Social Media": 120, "Entertainment": 60, "Productivity": 90},
    "Tuesday": {"Social Media": 100, "Entertainment": 80, "Productivity": 70},
    "Wednesday": {"Social Media": 130, "Entertainment": 70, "Productivity": 60},
    "Thursday": {"Social Media": 90, "Entertainment": 60, "Productivity": 80},
    "Friday": {"Social Media": 110, "Entertainment": 100, "Productivity": 50},
    "Saturday": {"Social Media": 180, "Entertainment": 120, "Productivity": 40},
    "Sunday": {"Social Media": 160, "Entertainment": 140, "Productivity": 30}
}
print(analyze_weekly_usage(weekly_usage))
```

Example Output:

```
{'total_category_usage': {'Social Media': 890, 'Entertainment': 630, 'Productivity':
```

Problem 8: Optimizing Break Times

As part of a digital wellbeing initiative, your goal is to help users optimize their break times throughout the day. Users have a list of activities they perform during breaks, each with a specified duration in minutes. You want to find two breaks that have the total duration closest to a target time.

Write the `find_best_break_pair()` function, which takes a list of integers representing the duration of each break in minutes and a target time in minutes. The function should return the pair of break durations that sum closest to the target time. If there are multiple pairs with the same closest sum, return the pair with the smallest break durations. If the list has fewer than two breaks, return an empty tuple.

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_best_break_pair(break_times, target):
  pass
```

Example Usage:

```
break_times = [10, 20, 35, 40, 50]
break_times_2 = [5, 10, 25, 30, 45]
break_times_3 = [15, 25, 35, 45]
break_times_4 = [30]

print(find_best_break_pair(break_times, 60))
print(find_best_break_pair(break_times_2, 50))
print(find_best_break_pair(break_times_3, 70))
print(find_best_break_pair(break_times_4, 60))
```

Example Output:

```
(20, 40)
(5, 45)
(25, 45)
()
```

Close Section