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

Problem 1: NFT Name Extractor

You're curating a large collection of NFTs for a digital art gallery, and your first task is to extract the names of these NFTs from a given list of dictionaries. Each dictionary in the list represents an NFT, and contains information such as the name, creator, and current value.

Write the `extract_nft_names()` function, which takes in this list and returns a list of all NFT names.

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 extract_nft_names(nft_collection):
    pass
```

Example Usage:

```
def extract_nft_names(nft_collection):
    nft_names = []
    for nft in nft_collection:
        nft_names.append(nft["name"])
    return nft_names

# Example usage:
nft_collection = [
    {"name": "Abstract Horizon", "creator": "ArtByAlex", "value": 5.4},
    {"name": "Pixel Dreams", "creator": "DreamyPixel", "value": 7.2},
    {"name": "Future City", "creator": "UrbanArt", "value": 3.8}
]

nft_collection_2 = [
    {"name": "Crypto Kitty", "creator": "CryptoPets", "value": 10.5},
    {"name": "Galactic Voyage", "creator": "SpaceArt", "value": 6.7}
]

nft_collection_3 = [
    {"name": "Golden Hour", "creator": "SunsetArtist", "value": 8.9}
]

print(extract_nft_names(nft_collection))
print(extract_nft_names(nft_collection_2))
print(extract_nft_names(nft_collection_3))
```

Example Output:

```
['Abstract Horizon', 'Pixel Dreams', 'Future City']
['Crypto Kitty', 'Galactic Voyage']
['Golden Hour']
```

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

Problem 2: NFT Collection Review

You're responsible for ensuring the quality of the NFT collection before it is displayed in the virtual gallery. One of your tasks is to review and debug the code that extracts the names of NFTs from the collection. A junior developer wrote the initial version of this function, but it contains some bugs that prevent it from working correctly.

Task:

1. Review the provided code and identify the bug(s).
2. Explain what the bug is and how it affects the output.
3. Refactor the code to fix the bug(s) and provide the correct implementation.

```
def extract_nft_names(nft_collection):
    nft_names = []
    for nft in nft_collection:
        nft_names += nft["name"]
    return nft_names
```

Example Usage:

```
nft_collection = [
    {"name": "Abstract Horizon", "creator": "ArtByAlex", "value": 5.4},
    {"name": "Pixel Dreams", "creator": "DreamyPixel", "value": 7.2}
]

nft_collection_2 = [
    {"name": "Golden Hour", "creator": "SunsetArtist", "value": 8.9}
]

nft_collection_3 = []

print(extract_nft_names(nft_collection))
print(extract_nft_names(nft_collection_2))
print(extract_nft_names(nft_collection_3))
```

Example Output:

```
['Abstract Horizon', 'Pixel Dreams']
['Golden Hour']
[]
```

Problem 3: Identify Popular Creators

You have been tasked with identifying the most popular NFT creators in your collection. A creator is considered "popular" if they have created more than one NFT in the collection.

Write the `identify_popular_creators()` function, which takes a list of NFTs and returns a list of the names of popular creators.

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 identify_popular_creators(nft_collection):
    pass
```

Example Usage:

```
nft_collection = [
    {"name": "Abstract Horizon", "creator": "ArtByAlex", "value": 5.4},
    {"name": "Pixel Dreams", "creator": "DreamyPixel", "value": 7.2},
    {"name": "Urban Jungle", "creator": "ArtByAlex", "value": 4.5}
]

nft_collection_2 = [
    {"name": "Crypto Kitty", "creator": "CryptoPets", "value": 10.5},
    {"name": "Galactic Voyage", "creator": "SpaceArt", "value": 6.7},
    {"name": "Future Galaxy", "creator": "SpaceArt", "value": 8.3}
]

nft_collection_3 = [
    {"name": "Golden Hour", "creator": "SunsetArtist", "value": 8.9}
]

print(identify_popular_creators(nft_collection))
print(identify_popular_creators(nft_collection_2))
print(identify_popular_creators(nft_collection_3))
```

Example Output:

```
['ArtByAlex']
['SpaceArt']
[]
```

Problem 4: NFT Collection Statistics

You want to provide an overview of the NFT collection to potential buyers. One key statistic is the average value of the NFTs in the collection. However, if the collection is empty, the average value should be reported as `0`.

Write the `average_nft_value` function, which calculates and returns the average value of the NFTs in the collection.

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 average_nft_value(nft_collection):
    pass
```

Example Usage:

```
nft_collection = [
    {"name": "Abstract Horizon", "creator": "ArtByAlex", "value": 5.4},
    {"name": "Pixel Dreams", "creator": "DreamyPixel", "value": 7.2},
    {"name": "Urban Jungle", "creator": "ArtByAlex", "value": 4.5}
]
print(average_nft_value(nft_collection))

nft_collection_2 = [
    {"name": "Golden Hour", "creator": "SunsetArtist", "value": 8.9},
    {"name": "Sunset Serenade", "creator": "SunsetArtist", "value": 9.4}
]
print(average_nft_value(nft_collection_2))

nft_collection_3 = []
print(average_nft_value(nft_collection_3))
```

Example Output:

```
5.7
9.15
0
```

Problem 5: NFT Tag Search

Some NFTs are grouped into collections, and each collection might contain multiple NFTs. Additionally, each NFT can have a list of tags describing its style or theme (e.g., `"abstract"`, `"landscape"`, `"modern"`). You need to search through these nested collections to find all NFTs that contain a specific tag.

Write the `search_nft_by_tag()` function, which takes in a nested list of NFT collections and a tag to search for. The function should return a list of NFT names that have the specified tag.

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 search_nft_by_tag(nft_collections, tag):
    pass
```

Example Usage:

```
nft_collections = [
    [
        {"name": "Abstract Horizon", "tags": ["abstract", "modern"]},
        {"name": "Pixel Dreams", "tags": ["pixel", "retro"]}
    ],
    [
        {"name": "Urban Jungle", "tags": ["urban", "landscape"]},
        {"name": "City Lights", "tags": ["modern", "landscape"]}
    ]
]

nft_collections_2 = [
    [
        {"name": "Golden Hour", "tags": ["sunset", "landscape"]},
        {"name": "Sunset Serenade", "tags": ["sunset", "serene"]}
    ],
    [
        {"name": "Pixel Odyssey", "tags": ["pixel", "adventure"]}
    ]
]

nft_collections_3 = [
    [
        {"name": "The Last Piece", "tags": ["finale", "abstract"]}
    ],
    [
        {"name": "Ocean Waves", "tags": ["seascape", "calm"]},
        {"name": "Mountain Peak", "tags": ["landscape", "adventure"]}
    ]
]

print(search_nft_by_tag(nft_collections, "landscape"))
print(search_nft_by_tag(nft_collections_2, "sunset"))
print(search_nft_by_tag(nft_collections_3, "modern"))
```

Example Output:

```
['Urban Jungle', 'City Lights']
['Golden Hour', 'Sunset Serenade']
[]
```

Problem 6: NFT Queue Processing

NFTs are added to a processing queue before they are displayed. The queue processes NFTs in a First-In, First-Out (FIFO) manner. Each NFT has a processing time, and you need to determine the order in which NFTs should be processed based on their initial position in the queue.

Write the `process_nft_queue()` function, which takes a list of NFTs. The function should return a list of NFT names in the order they were processed.

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_nft_queue(nft_queue):
    pass
```

Example Usage:

```
nft_queue = [
    {"name": "Abstract Horizon", "processing_time": 2},
    {"name": "Pixel Dreams", "processing_time": 3},
    {"name": "Urban Jungle", "processing_time": 1}
]
print(process_nft_queue(nft_queue))

nft_queue_2 = [
    {"name": "Golden Hour", "processing_time": 4},
    {"name": "Sunset Serenade", "processing_time": 2},
    {"name": "Ocean Waves", "processing_time": 3}
]
print(process_nft_queue(nft_queue_2))

nft_queue_3 = [
    {"name": "Crypto Kitty", "processing_time": 5},
    {"name": "Galactic Voyage", "processing_time": 6}
]
print(process_nft_queue(nft_queue_3))
```

Example Output:

```
['Abstract Horizon', 'Pixel Dreams', 'Urban Jungle']
['Golden Hour', 'Sunset Serenade', 'Ocean Waves']
['Crypto Kitty', 'Galactic Voyage']
```

Problem 7: Validate NFT Addition

You want to ensure that NFTs are added in a balanced way. For example, every `"add"` action must be properly closed by a corresponding `"remove"` action.

Write the `validate_nft_actions()` function, which takes a list of actions (either `"add"` or `"remove"`) and returns `True` if the actions are balanced, and `False` otherwise.

A sequence of actions is considered balanced if every `"add"` has a corresponding `"remove"` and no `"remove"` occurs before an `"add"`.

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 validate_nft_actions(actions):
    pass
```

Example Usage:

```
actions = ["add", "add", "remove", "remove"]
actions_2 = ["add", "remove", "add", "remove"]
actions_3 = ["add", "remove", "remove", "add"]

print(validate_nft_actions(actions))
print(validate_nft_actions(actions_2))
print(validate_nft_actions(actions_3))
```

Example Output:

```
True
True
False
```

Problem 8: Find Closest NFT Values

Buyers often look for NFTs that are closest in value to their budget. Given a sorted list of NFT values and a budget, you need to find the two NFT values that are closest to the given budget: one that is just below or equal to the budget and one that is just above or equal to the budget. If an exact match exists, it should be included as one of the values.

Write the `find_closest_nft_values()` function, which takes a sorted list of NFT values and a budget, and returns the pair of the two closest NFT values.

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_closest_nft_values(nft_values, budget):
    pass
```

Example Usage:

```
nft_values = [3.5, 5.4, 7.2, 9.0, 10.5]
nft_values_2 = [2.0, 4.5, 6.3, 7.8, 12.1]
nft_values_3 = [1.0, 2.5, 4.0, 6.0, 9.0]

print(find_closest_nft_values(nft_values, 8.0))
print(find_closest_nft_values(nft_values_2, 6.5))
print(find_closest_nft_values(nft_values_3, 3.0))
```

Example Output:

```
(7.2, 9.0)
(6.3, 7.8)
(2.5, 4.0)
```

Close Section

▾ Standard Problem Set Version 2

Problem 1: Meme Length Filter

You need to filter out memes that are too long from your dataset. Memes that exceed a certain length are less likely to go viral.

Write the `filter_meme_lengths()` function, which filters out memes whose lengths exceed a given limit. The function should return a list of meme texts that are within the acceptable length.

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_meme_lengths(memes, max_length):
    pass
```

Example Usage:

```
memes = ["This is hilarious!", "A very long meme that goes on and on and on...", "Sh
memes_2 = ["Just right", "This one's too long though, sadly", "Perfect length", "A b
memes_3 = ["Short", "Tiny meme", "Small but impactful", "Extremely lengthy meme that

print(filter_meme_lengths(memes, 20))
print(filter_meme_lengths(memes_2, 15))
print(filter_meme_lengths(memes_3, 10))
```

Example Output:

```
['This is hilarious!', 'Short and sweet']
['Just right', 'Perfect length']
['Short', 'Tiny meme']
```

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

Problem 2: Top Meme Creators

You want to identify the top meme creators based on the number of memes they have created.

Write the `count_meme_creators()` function, which takes a list of meme dictionaries and returns the creators' names and the number of memes they have created.

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_meme_creators(memes):
    pass
```

Example Usage:

```
memes = [
    {"creator": "Alex", "text": "Meme 1"},
    {"creator": "Jordan", "text": "Meme 2"},
    {"creator": "Alex", "text": "Meme 3"},
    {"creator": "Chris", "text": "Meme 4"},
    {"creator": "Jordan", "text": "Meme 5"}
]

memes_2 = [
    {"creator": "Sam", "text": "Meme 1"},
    {"creator": "Sam", "text": "Meme 2"},
    {"creator": "Sam", "text": "Meme 3"},
    {"creator": "Taylor", "text": "Meme 4"}
]

memes_3 = [
    {"creator": "Blake", "text": "Meme 1"},
    {"creator": "Blake", "text": "Meme 2"}
]

print(count_meme_creators(memes))
print(count_meme_creators(memes_2))
print(count_meme_creators(memes_3))
```

Example Output:

```
{'Alex': 2, 'Jordan': 2, 'Chris': 1}
{'Sam': 3, 'Taylor': 1}
{'Blake': 2}
```

Problem 3: Meme Trend Identification

You're tasked with identifying trending memes. A meme is considered "trending" if it appears in the dataset multiple times.

Write the `find_trending_memes()` function, which takes a list of meme texts and returns a list of trending memes, where a trending meme is defined as a meme that appears more than once in the 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 find_trending_memes(memes):
    pass
```

Example Usage:

```
memes = ["Dogecoin to the moon!", "One does not simply walk into Mordor", "Dogecoin
memes_2 = ["Surprised Pikachu", "Expanding brain", "This is fine", "Surprised Pikachu
memes_3 = ["Y U No?", "First world problems", "Philosoraptor", "Bad Luck Brian"]

print(find_trending_memes(memes))
print(find_trending_memes(memes_2))
print(find_trending_memes(memes_3))
```

Example Output:

```
['Dogecoin to the moon!', 'One does not simply walk into Mordor']
['Surprised Pikachu']
[]
```

Problem 4: Reverse Meme Order

You want to see how memes would trend if they were posted in reverse order.

Write the `reverse_memes()` function, which takes a list of memes (representing the order they were posted) and returns a new list with the memes in reverse order.

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 reverse_memes(memes):
    pass
```

Example Usage:

```
memes = ["Dogecoin to the moon!", "Distracted boyfriend", "One does not simply walk
memes_2 = ["Surprised Pikachu", "Expanding brain", "This is fine"]
memes_3 = ["Y U No?", "First world problems", "Philosoraptor", "Bad Luck Brian"]

print(reverse_memes(memes))
print(reverse_memes(memes_2))
print(reverse_memes(memes_3))
```

Example Output:

```
['One does not simply walk into Mordor', 'Distracted boyfriend', 'Dogecoin to the mo
['This is fine', 'Expanding brain', 'Surprised Pikachu']
['Bad Luck Brian', 'Philosoraptor', 'First world problems', 'Y U No?']
```

Problem 5: Trending Meme Pairs

You've been given partially completed code to identify pairs of memes that frequently appear together in posts. However, before you can complete the implementation, you need to ensure the plan is correct and then review the provided code to identify and fix any potential issues.

Your task is to:

1. **Plan:**

   Write a detailed plan (pseudocode or step-by-step instructions) on how you would approach solving this problem. Consider how you would:

   - Iterate through each post.
   - Generate pairs of memes.
   - Count the frequency of each pair.
   - Identify pairs that appear more than once.
   - Ensure the final result is accurate and efficient.

2. **Review:**

   Examine the provided code and answer the following questions:

   - Are there any logical errors in the code? If so, what are they, and how would you fix them?
   - Are there any inefficiencies in the code that could be improved? If so, how would you optimize it?
   - Does the code correctly handle edge cases, such as an empty list of posts or posts with only one meme?

```
def find_trending_meme_pairs(meme_posts):
    pair_count = {}

    for post in meme_posts:
        for i in range(len(post)):
            for j in range(len(post)):
                if i != j:
                    meme1 = post[i]
                    meme2 = post[j]

                    if meme1 < meme2:
                        meme1, meme2 = meme2, meme1
                    pair = (meme1, meme2)
                    if pair in pair_count:
                        pair_count[pair] += 1
                    else:
                        pair_count[pair] = 1

    trending_pairs = []
    for pair in pair_count:
        if pair_count[pair] >= 2:
            trending_pairs.append(pair)

    return trending_pairs
```

Example Usage:

```
meme_posts_1 = [
    ["Dogecoin to the moon!", "Distracted boyfriend"],
    ["One does not simply walk into Mordor", "Dogecoin to the moon!"],
    ["Dogecoin to the moon!", "Distracted boyfriend", "One does not simply walk into
    ["Distracted boyfriend", "One does not simply walk into Mordor"]
]

meme_posts_2 = [
    ["Surprised Pikachu", "This is fine"],
    ["Expanding brain", "Surprised Pikachu"],
    ["This is fine", "Expanding brain"],
    ["Surprised Pikachu", "This is fine"]
]

meme_posts_3 = [
    ["Y U No?", "First world problems"],
    ["Philosoraptor", "Bad Luck Brian"],
    ["First world problems", "Philosoraptor"],
    ["Y U No?", "First world problems"]
]

print(find_trending_meme_pairs(meme_posts))
print(find_trending_meme_pairs(meme_posts_2))
print(find_trending_meme_pairs(meme_posts_3))
```

Example Output:

```
[('Distracted boyfriend', 'Dogecoin to the moon!'), ('Dogecoin to the moon!', 'One d
[('Surprised Pikachu', 'This is fine')]
[('First world problems', 'Y U No?')]
```

Problem 6: Meme Popularity Queue

You're tasked with analyzing the order in which memes gain popularity. Memes are posted in a sequence, and their popularity grows as they are reposted.

Write the `simulate_meme_reposts()` function, which takes a list of memes (representing their initial posting order) and simulate their reposting by processing each meme in the queue. Each meme can be reposted multiple times, and for each repost, it should be added back to the queue. The function should return the final order in which all reposts are processed.

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 simulate_meme_reposts(memes, reposts):
    pass
```

Example Usage:

```
memes = ["Distracted boyfriend", "Dogecoin to the moon!", "One does not simply walk
reposts = [2, 1, 3]

memes_2 = ["Surprised Pikachu", "This is fine", "Expanding brain"]
reposts = [1, 2, 2]

memes_3 = ["Y U No?", "Philosoraptor"]
reposts = [3, 1]

print(simulate_meme_reposts(memes, reposts))
print(simulate_meme_reposts(memes_2, reposts))
print(simulate_meme_reposts(memes_3, reposts))
```

Example Output:

```
['Distracted boyfriend', 'Dogecoin to the moon!', 'One does not simply walk into Mor
['Surprised Pikachu', 'This is fine', 'Expanding brain', 'This is fine', 'Expanding
['Y U No?', 'Philosoraptor', 'Y U No?', 'Y U No?']
```

Problem 7: Search for Viral Meme Groups

You're interested in identifying groups of memes that, when combined, have a total popularity score closest to a target value. Each meme has an associated popularity score, and you want to find the two memes whose combined popularity score is closest to the target value. The list of memes is already sorted by their popularity scores.

Write the `find_closest_meme_pair()` function, which takes a sorted list of memes (each with a name and a popularity score) and a target popularity score. The function should return the names of the two memes whose combined popularity score is closest to the target.

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_closest_meme_pair(memes, target):
    pass
```

Example Usage:

```
memes_1 = [("Distracted boyfriend", 5), ("Dogecoin to the moon!", 7), ("One does not
memes_2 = [("Surprised Pikachu", 2), ("This is fine", 6), ("Expanding brain", 9), ("
memes_3 = [("Philosoraptor", 1), ("Bad Luck Brian", 4), ("First world problems", 8),

print(find_closest_meme_pair(memes_1, 13))
print(find_closest_meme_pair(memes_2, 10))
print(find_closest_meme_pair(memes_3, 12))
```

Example Output:

```
('Distracted boyfriend', 'Dogecoin to the moon!')
('Surprised Pikachu', 'Expanding brain')
('Bad Luck Brian', 'First world problems')
```

Problem 8: Analyze Meme Trends

You need to analyze the trends of various memes over time. You have a dataset where each meme has a name, a list of daily popularity scores (number of reposts each day), and other metadata.

Write the `find_trending_meme()` function, which takes in a list of memes (each with a name and a list of daily repost counts) and a time range (represented by a start and end day, inclusive). The function should return the name of the meme with the highest average reposts over the specified period. If there is a tie, return the meme that appears first in the 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 find_trending_meme(memes, start_day, end_day):
    pass
```

Example Usage:

```
memes = [
    {"name": "Distracted boyfriend", "reposts": [5, 3, 2, 7, 6]},
    {"name": "Dogecoin to the moon!", "reposts": [2, 4, 6, 8, 10]},
    {"name": "One does not simply walk into Mordor", "reposts": [3, 3, 5, 4, 2]}
]

memes_2 = [
    {"name": "Surprised Pikachu", "reposts": [2, 1, 4, 5, 3]},
    {"name": "This is fine", "reposts": [3, 5, 2, 6, 4]},
    {"name": "Expanding brain", "reposts": [4, 2, 1, 4, 2]}
]

memes_3 = [
    {"name": "Y U No?", "reposts": [1, 2, 1, 2, 1]},
    {"name": "Philosoraptor", "reposts": [3, 1, 3, 1, 3]}
]
print(find_trending_meme(memes, 1, 3))
print(find_trending_meme(memes_2, 0, 2))
print(find_trending_meme(memes_3, 2, 4))
```

Example Output:

```
Dogecoin to the moon!
This is fine
Philosoraptor
```

Close Section

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