Assignment 01

Instructions

1. Write your functions according to the signature provided below in a python(.py) file and submit them to the autograder for evaluation.
2. The autograder has a limited number of attempts. Hence, test it thoroughly before submitting it for evaluation.
3. Save your python file as text(.txt) file and submit it to the canvas.
4. If the assignment has theoretical questions, upload those answers as a pdf file on the canvas.
5. Submission on the canvas is mandatory and the canvas submitted text file should be the same as that of the final submission of the autograder. If not, marks will be deducted.
6. No submission on canvas or no submission on autograder then marks will be deducted.
7. Access the auto grader at https://c200.luddy.indiana.edu.

Question 1 - IUB Free Time Fun

After two weeks of starting his Master's program at Indiana University Bloomington, John realized that he has more free time than he expected. To make the most of it, he decided to watch as many movies and TV series as possible. However, he couldn't find all the series online, so he asked his friends for their collections along with their recommendations.

John bought a new hard drive with a capacity of n GB for storing these movies and series. Since he is working part-time and studying, his budget was limited, so he could only afford a hard drive with n GB of storage. Being an avid re-watcher and a person who enjoys rating the content he consumes, John knows that his preferences and ratings may change every time he watches something. However, his hard drive's storage is limited, so he needs to design an algorithm that manages the content on his hard drive efficiently.

Whenever John wants to store a new movie or series on his hard drive, he must decide which content to remove if the drive is full. He wishes to remove the content that he has accessed the least number of times. If there is a tie between multiple pieces of content, he will remove the one he accessed the longest time ago.

Additionally, whenever John wants to watch a movie or series, he first checks if it's stored on his hard drive. If it is, the access count for that content is increased. Similarly, whenever John adds or updates the rating of a movie or series, the access count for that content is also increased.

Task:

You need to implement this storage management algorithm for John. You are given the size of the hard drive (n GB), and a series of operations. Each movie or series takes up exactly 1 GB of storage.

Input:

The first line contains an integer n, the capacity of the hard drive in GB.
The next lines contain operations of two types:
`1 MovieName` – John checks if the movie/series MovieName is present on the hard drive.
`2 MovieName Rating` – John either copies or updates the rating of MovieName on his hard drive.

Output:

For each `1 MovieName` operation, output Rating if the movie/series is present on the hard drive, otherwise output -1.
For each `2 MovieName Rating` operation, update the hard drive. If the hard drive is full, remove the content that has been accessed the least number of times. If there's a tie, remove the content that was accessed the longest time ago.

Constraints

- `1 <= size(MovieName) <= 6`
- `1 <= Rating <= 10`
- `1 <= n <= 10^4`
- Total number of operations of type 1 or 2 will be at most `10^5`.

Note:

- Assume that each movie/series is stored as a single file of 1 GB in size.
- Updating the rating of an existing movie/series does not require additional space.
- Every time John accesses or updates a movie/series, the access count for that movie/series is incremented.
- MovieName is a string consisting of lower case alphabets
- Rating is a integer ranging from 1 to 10

Test Cases:

```
Input:
n = 2
operations = [
    (1, "GOT"),  # Check if "GOT" is present
    (2, ("GOT", 9)),  # Add "GOT" with rating 9
    (1, "GOT"),  # Check if "GOT" is present
    (2, ("NARUT", 10)),  # Add "NARUT" with rating 10
    (1, "NARUT"),  # Check if "NARUT" is present
    (2, ("BARUT", 6)),  # Add "BARUT" with rating 6
    (1, "GOT"),  # Check if "GOT" is present
    (1, "BARUT"),  # Check if "BARUT" is present
]

Output:
[-1, 9, 10, -1, 6]

Explanation:
1. Operation 1: Check "GOT" - Returns -1 because "GOT" is not on the hard
drive yet.
2. Operation 2: Add "GOT" with rating 9 - Adds "GOT" to the hard drive.
Access count of "GOT" is now `1`.
3. Operation 3: Check "GOT" - Returns 9 because "GOT" is present. Access
count of "GOT" increases to `2`.
4. Operation 4: Add "NARUT" with rating 10 - Adds "NARUT" to the hard drive.
Access count of "NARUT" is now `1`. The hard drive now contains "GOT" (count=`2`)
```

and "NARUT" (count=`1`).
5. Operation 5: Check "NARUT" - Returns 10 because "NARUT" is present.
Access count of "NARUT" increases to `2`.
6. Operation 6: Add "BARUT" with rating 6 - The hard drive is full, so it must evict one item. Both "GOT" and "NARUT" have the same access count (`2`). The least recently used among them is "GOT" (because "NARUT" was accessed more recently). Therefore, "GOT" should be evicted. The hard drive now contains "NARUT" (count=`2`) and "BARUT" (count=`1`).
7. Operation 7: Check "GOT" - Returns -1 because "GOT" was evicted.
8. Operation 8: Check "BARUT" - Returns 6. Access count of "BARUT" increases to `2`.

Input:
n = 2
operations = [
    (2, ("GOT", 9)), # Add "GOT" with rating 9
    (2, ("NARUT", 10)), # Add "NARUT" with rating 8
    (1, "GOT"), # Check if "GOT" is present (increases access count)
    (2, ("BARUT", 6)), # Add "BARUT" with rating 6
    (1, "NARUT"), # Check if "NARUT" is present
    (2, ("GOT", 7)), # Update rating of "GOT" (increases access count)
    (2, ("BBAD", 9)), # Add "BBAD" with rating 9
    (1, "BARUT"), # Check if "BARUT" is present
    (1, "GOT"), # Check if "GOT" is present
    (1, "BBAD"), # Check if "BBAD" is present
]

Output:
[9, -1, -1, 7, 9]

Explanation:

1. Operation 1: Add "GOT" with rating 9 - Adds "GOT" to the hard drive.
Access count of "GOT" is `1`.
2. Operation 2: Add "NARUT" with rating 10 - Adds "NARUT" to the hard drive.
Access count of "NARUT" is `1`. The hard drive now contains "GOT" and "NARUT", both with access count `1`.
3. Operation 3: Check "GOT" - Returns 9. Access count of "GOT" increases to `2`.
4. Operation 4: Add "BARUT" with rating 6 - The hard drive is full, so it evicts "NARUT". The hard drive now contains "GOT" (count=`2`) and "BARUT" (count=`1`).
5. Operation 5: Check "NARUT" - Returns -1, because "NARUT" was evicted.
6. Operation 6: Update "GOT" with rating 7 - Access count of "GOT" increases to `3`.
7. Operation 7: Add "BBAD" with rating 9 - The hard drive is full, so it evicts "BARUT" because "BARUT" has a lower access count (`1`) compared to "GOT" (`3`).
The hard drive now contains "GOT" (count=`3`) and "BBAD" (count=`1`).
8. Operation 8: Check "BARUT" - Returns -1, because "BARUT" was evicted.
9. Operation 9: Check "GOT" - Returns 7. Access count of "GOT" increases to `4`.
10. Operation 10: Check "BBAD" - Returns 9. Access count of "BBAD" increases to `2`.

Function

```
def processOperations(n: int, operations: List[Tuple[int, Union[str, Tuple[str, int]]]])
    -> List[str]:
    # Your code goes here
```

Question 2

Find the time complexities of the following functions. Provide detailed solutions:

```
a) def function1(n):
    for i in range(0, n):
        for j in range(0, i):
            k = j
            while k < n:
                k += 1
    return

b) def function2(n):
    i = n
    while i > 0:
        i = i // 3
    return

c) def function3(n):
    i = 1
    while i * i < n:
        i += 1
    return

d) def function4(n):
    i = n
    while i > 0:
        for j in range(0, n):
            for k in range(0, j):
                x = k
        i = i // 2
    return

e) def function5(n):
    i = 2
    while i < n:
        i = i * i
    return
```

Question 3 - Combined Signals Strength

You have an array called `signals` that represents a series of network signals. You want to see how two neighboring signals can work together to make a stronger signal.

Your task is to return a new array called `combined_signal_strength` that contains the combined strength of each pair of neighboring signals. Use the `bitwise OR` operation to combine each pair of signals.

Constraints

- 2 <= signals.length <= 100
- 0 <= signals[i] <= 100

Test Cases:

```
Input: signals = [5,4,9,11]
Output: [5,13,11]
Explaination:
- 5 | 4 = 5
- 4 | 9 = 13
- 9 | 11 = 11
```

```
Input:signals = [8,4,2]
Output: [12, 6]
Explaination:
- 8 | 4 = 12
- 4 | 2 = 6
```

Function

```
def combine_signals(signals):
    return combined_signal_strength
```

Question 4 - Elite Teams

Once upon a time, there was a kingdom where soldiers were known for their unique skills and abilities. Each soldier in the kingdom had a distinct strength level, represented by a unique rating. The king decided to form elite teams of three soldiers to protect the realm from an impending threat. To ensure that the teams were perfectly balanced, the king issued a challenge: A team of three soldiers would only be deemed worthy if their skills showed a clear progression or regression. In other words, the team would be valid if either the first soldier in the lineup was weaker than the second, and the second was weaker than the third (indicating a rising progression), or the first soldier was stronger than the second, and the second was stronger than the third (indicating a descending regression). Now, your task as the king's advisor is to figure out how many such elite teams can be formed from the soldiers. Remember, a single soldier can be part of multiple teams, but the teams must be in the correct order for the kingdom to remain safe.

Can you help the king form his elite teams?

Constraints

- n == rating.length
- 3 <= n <= 1000
- 1 <= rating[i] <= 105

Note: A soldier can be a part of multiple teams. Think how this can help.

Test Cases:

```
Input: [2,5,3,4,1]
Output:3
Explanation: We can form three teams given the conditions. (2,3,4), (5,4,1), (5,3,1).
```

```
Input: [2,1,3]
Output:0
Explanation: We can't form any team given the conditions.
```

Function

```
def numberOfTeams(ratings:List[int])->int:
    #Your code goes here.
```

Question 5 - Pirate Crew Task

Luffy and his crew have found a treasure map with mysterious numbers on it, representing the power levels of different crew members. The map has an integer array called `powerLevels`, where each element represents the power of a specific crew member.

Your task is to help Luffy calculate a new array called `newPower`, where `newPower[i]` represents the combined power of all the crew members except the one at position `i`.

However, there's a catch! The calculation must be done without using any division techniques (because the treasure map forbids it), and it must be done in O(n) time (because the crew is in a hurry).

The combined power of any group of crew members is guaranteed to fit in a 32-bit integer.

Constraints

- 2 <= powerLevels.length <= 10^5
- -30 <= powerLevels[i] <= 30
- The product of any prefix or suffix of `powerLevels` is guaranteed to fit in a 32-bit integer.

Follow up: Can you solve the problem in `O(1)` extra space complexity? (The output array **does not** count as extra space for space complexity analysis.)

Test Cases:

```
Input:powerLevels = [1,2,3,4]
Output: [24,12,8,6]
Explaination:
- For the Zeroth index 2*3*4 = 24
- For the First index 1*3*4 = 12
- For the Second index = 1*2*4 = 8
- For the Third index = 1*2*3 = 6
```

```
Input: powerLevels = [-1,1,0,-3,3]
Output: [0,0,9,0,0]
Explaination:
- For the Zeroth index 1*0*(-3)*3 = 0
- For the First index -1*0*(-3)*3= 0
- For the Second index = -1*1*(-3)*3 = 9
- For the Third index = -1*1*0*3 = 0
- For the Fourth index = -1*1*0*(-3) = 0
```

Function

```
def findPowerLevels(powerLevels: list[int]) -> list[int]:
    return powerLevels
```

Question 6 - Anagram Half Match

Indiana Jones, the famed archaeologist, has stumbled upon an ancient puzzle while exploring the ruins of a long-lost civilization. The most prized artifact in the treasury there is a mystical tablet inscribed with a series of enigmatic symbols. Upon further inspection, Indiana discovered that the symbols on the tablet were actually two halves of a string. To unlock the hidden message, Indiana must decipher whether these two halves can be rearranged to form identical sequences. Each pair of symbols is a key that opens a unique chamber of secrets. Your task is to write a function 'decrypt(s)' such that it will help Indiana Jones solve this puzzle. The function should take a single string 's' that represents the symbols on the tablet. Your goal is to return the number of character changes needed in the first half of the string to make it an anagram of the second half.

Constraints

- 1<=n<=100
- 1<= |s| <10^4
- s consists of only characters in the range ascii [a-z]

Note: To solve the puzzle, think about how many characters from the first half need to be swapped with characters from the second half to make the two halves anagrams of each other.

Test Cases:

```
Input: aaabbb
Output:3
Explanation: We split it into two strings 'aaa' and 'bbb' so we have to replace all
the characters from the second string to make identical strings
```

```
Input: xyyx
Output:0
Explanation: When we split them into two halves they are already identical to each
other so there is no need for any changes or substitutions.
```

Function

```
def decrypt(s:str)->int:
    #Your code goes here.
```

Question 7 - Count and Say

In a distant kingdom, there is a magical scribe who writes down sequences of numbers in a peculiar way. The first sequence is simply "1". Each subsequent sequence is created by describing the previous sequence using a special rule: the scribe counts the number of consecutive digits and then writes down that count followed by the digit itself.

For instance, if the scribe starts with the sequence "11122411", they would describe it as follows:

- Three 1s become "31"
- Two 2s become "22"
- One 4 becomes "14"
- Two 1s become "21"
- So, the next sequence would be "31221421".

Your task is to help the magical scribe by determining the nth sequence in this mysterious pattern. Given a positive number n, can you figure out what the scribe will write as the nth sequence?

Test Cases:

```
Input: n = 1
Output: "1"
Explanation: magicalScribe(1) = "1". This is the base case.
```

```
Input: n = 5
Output: "111221"
Explanation: magicalScribe(1) = "1".
Now, we have one "1". So, magicalScribe(2) = "11".
Now, we have two 1s. So, magicalScribe(3) = "21".
Now, we have one "2" and one "1". So, magicalScribe(4) = "1211".
Now, we have one "1", one "2", and two 1s. So, magicalScribe(5) = "111221".
```

Constraints

- 1 <= n <= 30

Function Signature

```
def magicalScribe(n: int) -> str:
    pass
```

Question 8 - Vowel Sort Decode

Batman, while on a mission in Gotham, receives a coded message from his trusted ally, Inspector Gordon. The message is hidden in a string of letters, but there's a catch! The consonants of the message are fixed in place, acting as a kind of lock, and they cannot be moved. However, the vowels

(which serve as the key) have been scrambled and must be sorted by their strength—measured by their ASCII value—so that the message can be decoded properly.

To ensure that the message is deciphered correctly, Batman must follow these rules:

- All consonants must stay exactly where they are; they can't be moved or rearranged.
- The vowels, which include 'a', 'e', 'i', 'o', and 'u' (both lowercase and uppercase), need to be sorted in ascending order of their ASCII values.
- Once the vowels are sorted, Batman will unlock the true message to proceed with his mission.

Constraints

- 1 <= message.length <= 10,000
- `The message will only contain uppercase and lowercase English letters.`

Test Cases:

```
Input: "BAtmAnCavE"
Output: "BAtmAnCEva"
Explanation: the vowels uEae are rearranged to A A E a
```

```
Input: "bRucEwaYne"
Output: "bREcaweYnu"
Explanation: the vowels uEae are rearranged to E a e u
```

Function

```
def sortVowels(code: str) -> str:
    return result
```

Question 9 - Help Elara

`Why use one space when you can use twenty? Make every line as perfect as your life plans.`
`--Rancho`

Once upon a time, there was a wise scribe named Elara. She was tasked with writing letters to the king, but the king had a very specific request: every line of the letter had to be exactly the same length. If it wasn't, the king would get upset and send the letter back. Elara had a box of `words` she wanted to use, and she needed to fit these words into each line so that every line had exactly the `width` number of characters. To do this, Elara decided to be clever. She would fit as many words as possible into a line, but if there were any leftover spaces, she would spread them out evenly between the words. However, sometimes the leftover spaces couldn't be perfectly divided, so she gave more spaces to the earlier gaps and fewer to the later ones. This made the lines look beautiful. But Elara knew that the last line should look special. It shouldn't be stretched out, just neatly aligned to the left, with any remaining space added at the end. By following this careful pattern, Elara's letters were always perfectly formatted, and the king was very pleased with her work.

Constraints

- 1 <= words.length <= 300

- 1 <= width <= 100
- words[i].length <= width

Test Cases:

```
Input: words = ["To","be","or","no","question","answer","is","question"], width = 13
Output: [
            "To  be  or no",
            "question     ",
            "answer     is",
            "question     "
        ]

Explanation: In the first line, spaces cannot be divided equally among gaps, so
the left gaps receive more spaces than the right ones.\
The second line is left-aligned because it contains only one word.\
The last line is also left-aligned, as specified in the requirements.

Input: words = ["Hello", "world", "this", "is", "a", "test", "of", "help", "Elara"],
width = 15
Output: [
            "Hello     world",
            "this  is a test",
            "of help Elara  "
        ]
```

Function

```python
def helpElara(words: List[str], width: int) -> List[str]:
    pass
```

Question 10 - Cyclonia's Quest

In the small town of Cyclonia, there's a unique roundabout that loops back to where a random place in the town. Locals say that if you keep walking on this path, you'll end up at the same spot again and again. This has led to a local legend about a treasure that can only be found by breaking the loop. Many curious visitors come to test the myth, wondering if they'll endlessly circle or if there's a way to find the treasure by stepping off the usual path.

Given `head`, the head of a linked list, return `true` if the treasure can be found, else return `false` (boolean values). Internally, `pos` is used to denote the index of the spot that tail's next pointer is connected to. Note that `pos` is not passed as a parameter.

Constraints

1. The number of the nodes in the list is in the range [0, 10^4].
2. -10^5 <= Node.val <= 10^5
3. `pos` is -1 or a valid index in the linked-list.
4. Solve in O(1) (i.e. constant) space complexity

Test Cases:

```
Input: head = [2, 1, 0, 10], pos = 2
Output: false
Explanation: They looped back to a previous spot i.e. 2nd node (0-indexed), so the
treasure was not found.

Input: head = [20, -1], pos = 0
Output: false
Explanation: They looped back to a previous spot i.e. 0th node (0-indexed), so the
treasure was not found.

Input: head = [12], pos = -1
Output: true
Explanation: They broke out of the loop, so the treasure was found.
```

Function

```python
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

def findTreasure(head):
    return res
```

Question 11

Prove or disprove the following questions on asymptotic notation. Provide detailed explanations to convey your ideas clearly. You can either solve them over paper, scan and upload the pdf file over canvas. Or Type the solutions using latex and upload the compiled pdf file over canvas.

a. Prove or disprove that $f(n) \in O(g(n))$ where $f(n) = 3n + 4$ and $g(n) = n$.

b. Prove or disprove that $f(n) \in \Theta(g(n))$ where $f(n) = 5n^2 + 2n \log n$ and $g(n) = n^2$.

c. Prove or disprove that $f(n) \in \Omega(g(n))$ where $f(n) = n \log n + 100$ and $g(n) = n$.

d. Prove or disprove that $f(n) \in \Theta(g(n))$ where $f(n) = n^{\log n}$ and $g(n) = 2^n$.

e. Prove or disprove that $f(n) \in o(g(n))$ where $f(n) = n^{1.5} + \log^2 n$ and $g(n) = n^2$.