Assignment 04

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

In the quaint town of "Apple Town," an annual competition with a unique twist captivates its residents. Those fortunate enough to own farms in this charming town participate in this event, which unfolds as follows: Each farm owner has the opportunity to cultivate one of three distinct types of apple trees on their land. There's a catch, though – each farm can only host one type of apple tree, and the town pays some price to cultivate that type of tree on the farm. You are given with the price town has to pay to plant each type of tree for each person.

What sets this competition apart is that neighboring farms must not have the same type of apple tree. The town's objective is to minimize its expenses while ensuring that the conditions of diverse apple tree cultivation are met.

The winner of this unique contest isn't determined by the usual criteria but rather by the ingenious algorithm that calculates the town's lowest expenditure. The participant who devises this algorithm and computes the least amount the town must pay emerges victorious. Can you also provide us with the minimum cost that the town needs to incur?

Examples

Example 1:

```
Input: AppleTreePrice = [[2,5,6]]
Output: 2
```

Explanation: Only one person is there, we can simply plant tree with the lowest price.

Example 2:

```
Input: AppleTreePrice = [[1,2,3],[1,2,3],[1,2,3]]
Output: 4
```

Explanation: There are 3 people in the competition, for 1st person, price to grow 1st type of tree is 1, 2nd type of tree is 2 and 3 for the 3rd type. Same for the 2nd and 3rd person.

One way to efficiently select the tree types to achieve the minimum expenditure is 1st person can plant 1st type of tree resulting in total expenditure 1.

2nd person can plant 2nd type of tree resulting in total expenditure 3 (Note that we can't select 1st type of tree for 2nd person as we are not allowed to select same type of tree in the neighbouring farms as we have already selected 1st type of tree for the first person)

3rd person can plant 1st type of tree resulting in total expenditure 4

Example 3:

```
Input: AppleTreePrice = [[4,12,7],[2,34,13]]
Output: 9
Explanation: 1st person can plant 3rd type of tree, final expenditure = 7

2nd person can plant 1st type of tree, final expenditure = 7 + 2 = 9
```

Constraints:

- Solve the problem in O(n) time complexity
- Solve the problem in O(1) space complexity. (Due to autograder issue of manipulating the same input variable, you are allowed to make a single copy of input and then change that copied array without consuming any additional space.)

Function

```
def MinimumExpenditure(AppleTreePrice):
  #return int
  return minExpenditure
```

Question 2

Alice and Bob are biology students at IU. They have two genome sequences of lengths 'A' and 'B' and want to align these sequences. A valid alignment is one where each element in sequence 'A' either matches an element in sequence 'B' or aligns with an empty space marker. Furthermore, the order of elements in sequence 'A' must be maintained in the alignment. How many valid ways can they align these two sequences?

For example, if 'A = 2' and 'B = 2', a valid alignment could be '[a1, -, a2][-, b1, b2]', where 'a1' aligns with an empty space, 'b1' aligns with an empty space, and 'a2' aligns with 'b2'. However, the alignment '[a2, -, a1][-, b1, b2]' is invalid because it doesn't preserve the order of elements in 'A'.

Examples

Example 1:

```
Input: A = 1, B = 1
Output: 3
```

Explanation: There are three ways to align two sequences of length 1: [a1][b1], [a1, -][-, b1], [-, a1][b1, -]

Example 2:

```
Input: A = 1, B = 2
Output: 5
```

Explanation: There are five ways to align two sequences of length 1 and 2: [a1, -][b1, b2], [-, a1][b1, b2], [a1, -, -][-, b1, b2], [-, a1, -][b1, -, b2], [-, -, a1][b1, b2, -]

Constraints:

Solve this problem using Dynamic Programming.

Function

```
def alignments(A, B):
  #return int
  return numAlignments
```

Question 3

While strolling down a street, you encounter a rather unusual one. Each block along this street displays a positive number. Being a math enthusiast, you can't help but be intrigued by these numbers. You notice a pattern: the numbers on the blocks are in ascending order as you proceed, but occasionally, there are gaps between the numbers on adjacent blocks. This piques your curiosity, and you decide to embark on a quest to identify the smallest missing number in this sequence. Can you find it?

Note : If you can't find the missing number in the array, return the next number

Examples:

Example1:

```
Input: [0, 1, 2, 3]
Output: 4
```

Explanation: All the blocks have perfect increasing numbers and nothing is missing. As there is nothing missing we return the next number.

Example2:

```
Input: [0, 1, 2, 5, 7, 8]
Output: 3
```

Explanation: There are 2 places where we have the missing numbers, after 2 which is 3 and 4 and after 5 which is 6. But out of these 2 options, 3 is the smallest number that is missing from the given sequence.

Constraints:

- Solve this problem using Divide and Conquer.

Function:

```
def smallestMissingNumber(streetNumbers):
    #return int
    return number
```

Question 4

Mr. Jackson, who has dementia and limited time left, wants to distribute his inheritance among his children. He possesses magical treasure boxes, each containing some number of items. These boxes are connected in a unique way: a child can receive Box 1 or Box 1 and Box 2, but not Box 1 and Box 3 simultaneously.

The challenge is to find a distribution plan that ensures fairness while minimizing the sum of items in each partition. In other words, Mr. Jackson seeks a method to allocate the boxes among his children so that the difference in the total number of items each child receives is as small as possible. Can you help Mr. Jackson implement this equitable distribution plan? If Mr.Jackson is not able to distribute his inheritance to all of the children return -1.

- num_items: signifies the number of items in each box (list)
- num_boxes: total number of boxes (int)
- children: total number of Mr.Jacksons children (int)

Examples:

Example 1:

```
input: num_items = [1,2,3,4,5,6,7,8], num_boxes = 8 and children = 5
output: 9

Explanation:  Child 1 gets [1,2,3]
              Child 2 gets [4,5]
              Child 3 gets [6]
              Child 4 gets [7]
              Child 5 gets [8]
              Note: Highest sum amongst partition is 9
```

Example 2:

```
input: num_items = [1,2,3,4,5,6,7,8], num_boxes = 8 and children = 9
output: -1
```

Explanation: One child will be left out as num_boxes $<$ children.

Constraints:

- children > 0, num_boxes > 0.
- Required time complexity of `O(len(num_items) * log(sum(num_items) - max(num_items)))`.
- Return type integer.
- Return -1 if theres no solution.

Signature:

```
def solution_inheritance(num_items, num_boxes, children):
    #Add your code here
    return answer
```

Question 5

"The Abyss Valley," a notorious location with treacherous winding roads. This valley serves as a crucial route for travelers attempting to traverse the city. However, a sinister curse plagues the valley, claiming the lives of reckless drivers who speed through it, causing them to plummet into an abyss from which they never return.

The government has entrusted you with the responsibility of strategically installing speed bumps along the Abyss Valley. The government has specified three potential intervals for placing these speed bumps, and there's a crucial requirement: a speed bump must be placed at the very end of the valley to ensure maximum safety.

Your objective is to determine the maximum number of speed bumps that can be placed in this perilous valley while adhering to the government's guidelines and safeguarding the lives of the travelers who navigate this ominous route.

If you are not able to place a speed bump at the very end of the valley return 0.

Examples:

Example 1:

```
Input: len_road = 7, bump_int1 = 5, bump_int2 = 3 and bump_int3 = 2
Output: 3
```

```
Explanation:  Possible solution:
              Placed a speedbump at 2 (interval3, i.e 2)
              Placed a speedbump at 4 (interval3, i.e 2)
              Placed the *last* speedbump at 7 (interval2, i.e 3)
              Note: Total number of speedbumps placed = 3
```

Example 2:

```
Input: len_road = 13, bump_int1 = 5, bump_int2 = 7 and bump_int3 = 7
Output: 0
```

```
Explanation:
Cannot place speedbump at the end of the valley
Lets say,
    Placed a speedbump at 5 (interval1, i.e 5)
    Placed a speedbump at 10 (interval1, i.e 5)
    Now cannot place a speedbump at the end because
    the interval is either 5 or 7
```

Constraints:

- len_road > 0, bump_int1 > 0, bump_int2 > 0, and bump_int3 > 0.

- Required time complexity of `O(len_road)` and Space Complexity `O(len_road)`
- Return type integer.
- Return 0 if theres no solution.

Signature:

```
def place_max_speedbump(len_road, bump_int1, bump_int2, bump_int3):
    #Add your code here
    return answer
```

Question 6

Indiana Jones, while hunting a treaure in an ancient temple, comes across an abyss having several floating stones with numbers inscribed on them. When he steps on a stone having number *num*, it can take him any of the stones between the one he stepped on and the one *num* stones ahead (both inclusive). There is a pit of lava and he has to get to the other side in the shortest time possible in order to get the treasure, and...well, stay alive. You need to help him figure out how to find the fastest (the least number of stones used) path to cross the abyss. You can consider the last stone as the end of the abyss to make things easier. There will always be a path to the end of the abyss.

Examples

Example 1:

```
Input: [1, 1, 1, 4, 1, 1, 3]
Output: 4
```

Explanation: Stone 1 takes Indie to stone 2, stone 2 to stone 3, and stone 3 to stone 4. Stone 4 can take him to either stone 5 or stone 6 or stone 7. Since stone 7 is the end of the path, it takes him exactly 4 stones to reach there, hence 4 is the answer.

Example 2:

```
Input: [1, 1, 4, 6, 1, 1, 1, 1, 1, 1]
Output: 4
```

Explanation: Stone 1 takes Indie to stone 2, stone 2 to stone 3. Stone 3 can take him to one of stone 4, stone 5, stone 6, stone 7. In order to use the least number of stones, he must take stone 3 to stone 4, and then stone 4 to stone 10 to reach the end of the path. There exist longer routes but Indie would get burnt by the rising lava if he chooses those.

Constraints

- Time Complexity - O(N2)
- Space Complexity - O(N)

Signature

```
def find_path(stone_inscription_list):
  # Your code here
  return path_length
```

Question 7

In the heart of a tranquil village, Samuel arrived bearing ancient scrolls, each containing numbers meticulously arranged in ascending order. Whispers of a hidden secret had always lingered, and Samuel believed that by ingeniously arranging these scrolls, he could unveil the long-guarded mysteries of the village. The challenge was clear: to create one ultimate parchment by aligning these scrolls in such a way that all the numbers are seen in an ascending sequence.

Examples

Example 1:

```
Input: [[1, 4, 5], [1, 6, 7], [3, 3]]
Output: [1, 1, 3, 3, 4, 5, 6, 7]
```

Example 2:

```
Input: [[1, 2, 8], [3, 4, 9], [1, 2], [5, 7], [2, 3, 5, 7, 9]]
Output: [1, 1, 2, 2, 2, 3, 3, 4, 5, 5, 7, 7, 8, 9, 9]
```

Constraints

- Time Complexity - N log(K), where K is the total number of scrolls and N is total count of all numbers present across the K scrolls

Signature:

```python
def decode_cryptic_message(lists):
  # Your code here
  return arranged_numbers
```