Assignment 05

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

Once upon a time, in a bustling city named Bloomington, there was a complex network of bus routes connecting various neighborhoods and landmarks. The city was renowned for its efficient money saving schemes. However, the city's transportation authorities faced a challenge in optimizing the bus routes to save more money. To achieve the desired optimization, they needed to eliminate overlapping routes, which would save them more money. They have a list of list busStation in which each bus's route was represented as a pair of integers, where the first integer indicated the starting bus station and the second integer indicated the ending bus station for that specific bus. With this data in hand, they set out to eliminate overlapping routes. They analyzed each bus route one by one, comparing them with the other routes. If two routes had any overlap, they merged them into a single, more efficient route. This process continued until no more overlaps could be found. How many buses will remain after the buses with overlapping routes have been eliminated.

Examples

Example 1:

```
Input: busStation = [[2,8], [6, 10], [12, 14], [12, 20]]
Output: 2
```

Explanation: The buses running between bus stations [2, 8] and [6, 10] are combined into one. Similarly, the buses running between bus stations [12, 14] and [12, 20] are combined into one. So, the total number of buses running on the route are 2.

Example 2:

```
Input: busStation = [
    [1, 5],
    [3, 7],
    [6, 10],
]
Output: 1
```

Explanation: The buses running between bus stations [1, 5] and [3, 7] are combined into one. Then the bus on new route running between bus stations [1, 7] and [6, 10] are combined into one. So, the total number of buses running on the route are 1.

Constraints

- 0 <= len(busStation) <= 10000
- 0 <= busStations[i][0] <= busStations[i][1] <= 100000

Function

```
def busRemaining(busStation) :
  # your code goes here
  return remainingBuses #should return the number of remaining buses
```

Question 2

In a picturesque village nestled between rolling hills, there lived a curious mathematician named Professor Eliza. She was known far and wide for her fascination with numbers and her uncanny ability to solve the most perplexing mathematical problems.

One bright and sunny morning, as she sipped her tea and gazed out of her window at the scenic landscape, a new puzzle arrived at her doorstep. It was brought by a group of enthusiastic young students who had heard tales of her legendary problem-solving skills. The puzzle was neatly written on a piece of parchment and read as follows:

"You are given an array of positive integers. Starting with a score of zero, apply the following algorithm:

1. Choose the smallest integer in the array that is not marked. If there is a tie, choose the one with the smallest index.
2. Add the value of the chosen integer to your score.
3. Mark the chosen element and its two adjacent elements if they exist.
4. Repeat until all the array elements are marked."

What would be the score that would answer for this puzzle.

Examples

Example 1:

```
Input: numbers = [2,1,3,4,5,2]
Output: 7
Explanation: We mark the elements as follows:
'*' is used to represent marked elements
- 1 is the smallest unmarked element, so we mark it
and its two adjacent elements: [*2,*1,*3,4,5,2].
- 2 is the smallest unmarked element, so we mark it
and its left adjacent element: [*2,*1,*3,4,*5,*2].
- 4 is the only remaining unmarked element,
so we mark it: [*2,*1,*3,*4,*5,*2].
Our score is 1 + 2 + 4 = 7.
```

Example 2:

```
Input: numbers = [2,3,5,1,3,2]
Output: 5
Explanation: We mark the elements as follows:
'*' is used to represent marked elements
- 1 is the smallest unmarked element, so we mark it and
its two adjacent elements: [2,3,*5,*1,*3,2].
- 2 is the smallest unmarked element, since there are two of them,
we choose the left-most one, so we mark the one at index 0 and
its right adjacent element: [*2,*3,*5,*1,*3,2].
- 2 is the only remaining unmarked element, so we mark it: [*2,*3,*5,*1,*3,*2].
Our score is 1 + 2 + 2 = 5.
```

Constraints

- Use priority queue/heap to solve the problem
- 1 <= nums.length <= 10000
- 1 <= nums[i] <= 100000

Function

```python
def solvePuzzle(numbers):
  # your code goes here
  #should return the answer to the puzzle
  return answerToPuzzle
```

Question 3

Every summer, Sweet Scoops, a popular ice cream store was flooded with customers eager to taste their various signature flavors. To keep her customers happy, Mrs. Anderson introduced a new concept - a window of flavors.

There are many different flavors of ice cream displayed on a long counter. However, only a small window on the counter is open at any given time. Customers can only choose from the items within this window. The staff periodically shifts the window to the right to allow customers to see and order different flavors.

Here's how the concept worked: - Mrs. Anderson had a list of all her 'n' ice cream flavors, ordered in a specific sequence. - She moved the window of size k(k<=n) from the leftmost flavor to the rightmost flavor. - Every 15 minutes, she shifted the window one position to the right, revealing a new set of k flavors.

Given the prices of n ice cream flavors and size of the window k, your task is to write a program to efficiently calculate the median of the prices in each window of size k.

Examples

Example 1:

```
Input: prices = [2,3,1,5,9,6], k =3
Output: [2,3,5,6]
```

```
Given the prices of 6 flavors, and size of the window = 3;
Initial 3 values in the window : [2,3,1] - Median = 2
Next 3 values in the window : [3,1,5] - Median = 3
Next 3 values in the window : [1,5,9] - Median = 5
Next 3 values in the window : [5,9,6] - Median = 6
Hence, output is [2,3,5,6]
```

Example 2:

```
Input: prices = [2,3,1,5,9,6], k =4
Output: [2.5, 4.0, 5.5]

Given the prices of 6 flavors, and size of the window = 3;
Initial the values in the window : [2,3,1,5] - Median = 2+3/2 = 2.5
Next 3 values in the window : [3,1,5,9] - Median = 3+5/2 = 4.0
Next 3 values in the window : [1,5,9,6] - Median = 5+6/2 = 5.5
Hence, output is [2.5, 4.0, 5.5]
```

Constraints

- k<=n
- Use priority queue/heap to solve the problem

Function

```
def findMedianPrice(prices, k):
  # your code goes here
  return list of medians
```

Question 4

In the bustling city of Stars Hollow, skyscrapers pierced the clouds. The city was renowned for its ever-evolving skyline, a testament to human innovation and architectural prowess. But with such towering structures came an intriguing puzzle: how could one determine, for each building, the number of buildings shorter than itself to its right?

Given the heights of Stars Hollow's skyscrapers, determine, for each building, how many other buildings to its right have a lower height.

Examples

Example 1:

```
Input: heights = [5, 4, 3, 2, 6, 3]
Output: [4, 3, 1, 0, 1, 0]
Explanation:
To the right of 5 there are 4 buildings of lesser height(4, 3, 2, 3).
To the right of 4 there are 3 buildings of lesser height (3, 2, 3).
To the right of 3 there is 1 building of lesser height (2).
To the right of 2 there are 0 buildings of lesser height ().
To the right of 6 there is 1 building of lesser height (3).
To the right of 3 there are 0 buildings of lesser height ().
```

Constraints

- Solve the problem in O(nlogn)
- Do not use any inbuilt sorting funtions

Function

```
def shorterBuildings(heights):
  # your code goes here
  return your answer as a list
```

Question 5

In a quaint village nestled in a snowy valley, winter is fast approaching. The villagers are preparing for the chilly season by ensuring that every house stays warm and cozy. As a talented engineer, your task is to design a standard heater with a fixed warm radius that can efficiently heat all the houses.

The village is laid out along a straight road, with houses and heater locations marked at different positions. Each house can be warmed as long as it falls within one heater's warm radius range. Your challenge is to determine the minimum standard radius required for the heaters to cover all the houses while maintaining an efficient and cost-effective solution.

Examples

Example 1:

```
Input: houses = [3,1,2], heaters = [2]
Output: 1
```

Explanation:

```
Road:    -----1-------2-------3----
Houses:  ---House---House---House--
Heaters: -----------Heater---------
```

The only heater was placed in the position 2, and if we use the radius of 1, then all the houses can be warmed.

Example 2:

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

Explanation: The two heaters were placed at positions 1 and 4. We need to use a radius 1 standard, then all the houses can be warmed.

Constraints

- Time Complexity should be no more than `O(nlog(n) + mlog(m))`, where n is number of houses and m is number of heaters.

Function

```
def determineStandardRadius(houses, heaters):
  # add your code here
  #the standard radius that would keep all the houses warm
  return result
```

Question 6

In the enchanted forest of Literaria, mystical creatures named "magicals" roam freely. Each "magicals" represents a letter from the alphabet. As time has passed, some "magicals" have developed unique powers, so powerful that when two of the same kind come close to each other, their combined energies create chaotic magic bursts.

Elder Scroll, the wise old sage of Literaria, has a vision: to create a Grand Festival where all "magicals" can gather. But he's aware of the volatile energies. To ensure the safety of the event, he decides that "magicals" of the same kind must be at least k steps away from each other in the parade line.

You, being a trusted "magicals" Whisperer, are approached by Elder Scroll for this task. You are given a sequence that represents the initial parade lineup of the "magicals". Your task is to rearrange this lineup so that the same "magicals" are at least k steps apart. If it isn't feasible, you must gently inform Elder Scroll it is not possible to have the given "magicals" together without bursting. The output should be True, if the sequence of "magicals" can be rearranged according to the given k value. The output is False, if it's not possible to rearrange the "magicals" as per the given constraints.

Examples

Example 1:

```
Input: s= 'aabbcc', k=3
Output: True
```

Explanation: The given letters("magicals") can be arranged as 'abcabc' such that same letters("magicals") are at least a distance of 3 steps from each other

Example 2:

```
Input: houses = s='aaabc' , k=3
Output: False
```

Explanation: It is not possible to rearrange the given "magicals" with atleast 3 steps apart from each other for same type of "magicals".

Constraints

- string consists of only lowercase English letters i.e "magicals" are only lowercase English letters.
- 0 <= k <= s.length
- Use maxheap to solve the problem.
- Do not use any inbuilt heap functions

Function

```
def isRearrangePossible(s,k):
  # add your code here
  return
```

Question 7

In Assignment 3, Santa aimed to transmit an encoded message to Banta. Now, Santa has visited the GHC and intends to share the details with Banta. This time around, they've opted for Huffman coding to encode their text. Your assistance is required to help them implement this plan.

Both Santa and Banta have agreed upon a common class structure for this task.

```
class Huffman():
    def __init__(self):
        self.huffman_codes = {}
        self.source_string = ""

    def set_source_string(self, src_str):
        self.source_string = src_str

    def generate_codes(self):
        huffman_codes = {}

        # Write your code here

        self.huffman_codes =  huffman_codes

    def encode_message(self, message_to_encode):

        encoded_msg = ""

        # Write you code here

        return encoded_msg

    def decode_message(self, encoded_msg):

        decoded_msg = ""
        return decoded_msg
```

First, they will compute the frequencies of characters in the source_string, which is stored in the `source_string` attribute (Note: The autograder sets this attribute). Subsequently, they will invoke the `generate_codes` function to create Huffman codes, which will be stored in the `huffman_codes` attribute within the Huffman class. To encode or decode any message, they will utilize the `encode_message` and `decode_message` functions, respectively, leveraging the `huffman_codes` attribute.

Your help is pivotal in making their Huffman coding implementation a success.

Example

```
 if source_string = "cbadeeee"
 generate_codes() will create a huffman codes.
 the huffman tree will look like
                                     [abcde: 8]
                                     /        \
                                 [abcd: 4]    [e: 4]
                                  /      \
                             [ab: 2]     [cd: 2]
                            /     \        /    \
                         [a: 1] [b: 1]   [c: 1] [d: 1]


As the rule says left childs append 0s and right child append 1s
the huffman_codes attribute will be
{
    a: '000',
    b: '001',
    c: '010',
    d: '011',
    e '1'
}

now if encode_message('ae') is called it will return '0001'
and if decode_message('0100011011') is called it will return 'cbed'
```

Constraints

- All the strings will only contain lowercase English alphabets
- `encoded_msg` will only contain `'0'`s and `'1'`s

Rules and things to note

- While generating the Huffman codes, if two or more symbols have same frequency choose the one that is lexicographically smaller. (This is to make sure you have same codes as that of autograder)
- When you move to a left child, append '0' to the current Huffman code. (This is to make sure you have same codes as that of autograder)
- When you move to a right child, append '1' to the current Huffman code. (This is to make sure you have same codes as that of autograder)
- Do not touch the `__init__` and `set_source_string` functions
- `generate_codes` function will generate huffman codes for all the alphabets and store them in the `huffman_codes` attribute
- `encode_message` will encode the input strring using the codes from the `huffman_codes` attribute which was set by `generate_codes`
- `decode_message` will decode the input strring using the codes from the `huffman_codes` attribute which was set by `generate_codes`

Question 8

Let A =[a1, a2, a3, ...an] be a given sequence of integers. Implement the wavelet tree data structure to support efficient rank and select queries.

Implement the following functions

- `get_wavelet_level_order` returns all the levels using level order traversal of the Wavelet tree as list of lists.
- `rank` returns the count of the desired element in the given range.

Reference

1. Refer Lecture 14 - Page 20
2. You can also refer the following paper to understand how to implement https://arxiv.org/pdf/0903.4726.pdf

Note

1. If the node is a leaf node, return ‘X’ as the value, based on the count of the element. Look at the examples for better understanding.
2. The rank query returns the count of the given character in the given range of the array. If such a character is not present in the given range, return 0.

Example

Example 1:

```
wv_tree = Wavelet_Tree([6, 2, 0, 7, 9, 3, 1, 8, 5, 4])
```

```
wv_tree.get_wavelet_level_order()
```

[[‘1001100110’], [‘00101’, ‘00110’], [‘100’, ‘01’, ‘010’, ‘10’], [‘01’, ‘X’, ‘X’, ‘X’, ‘10’, ‘X’, ‘X’, ‘X’],[‘X’, ‘X’, ‘X’, ‘X’]]

```
wv_tree.rank(7, 3) # 0
```

Explanation: There is no 7 in the first 3 elements.

Example 2:

```
wv_tree = Wavelet_Tree([6, 2, 0, 7, 7, 9, 3, 1, 8, 5, 4])
```

```
wv_tree.get_wavelet_level_order()
```

[[‘10011100110’], [‘00101’, ‘000110’], [‘100’, ‘01’, ‘0110’, ‘10’], [‘01’, ‘X’, ‘X’, ‘X’, ‘10’, ‘XX’, ‘X’, ‘X’], [‘X’, ‘X’, ‘X’, ‘X’]]

```
wv_tree.rank(7, 5) # 2
```

Explanation - There are two 7’s in the first 5 elements. The two 7’s are correctly represented by two X’s in the leaf node over 4th level.

Constraints

1. Each integer is between 0 and 9.
2. 1 <= position <= number of elements in the array
3. The rank query should only be implemented using the built wavelet tree in log(n) time complexity.
4. The leaf nodes should represent the correct number of X's denoting the count of the element.

```python
# Feel free to change the class Node as per your requirement
class Node:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right

class Wavelet_Tree:
    def __init__(self, A:list[int]=[]):
        pass

    def get_wavelet_level_order(self):
        # Return level order traversal of the tree. Except the last level
        pass

    def rank(self, character, position):
        #Return the rank of the given character in the given position range.
        pass
```