Sorting (for this section, I am assuming that you are implicitly not requiring the big O part)

1. Sort a linked list using insertion sort. Write the description and code.
    1. Conceptual Description
        a. First, get the user's input and create a linked list. Next, iterate through each node one by one to sort the list. The first node scanned will be the head of the sorted list, then, and each node scanned thereafter will be inserted appropriately into the sorted list until there are no nodes remaining.
    2. Code
        a.

```
class ListNode:
    def __init__(self, data = None):
        self.data = data
        self.nextNode = None

class LinkedList:
    def __init__(self, inputArray):
        self.headNode = None
        self.createLinkedList(inputArray)

    def insertionSort(self):
        scannedNode = tempNode = ListNode(0)
        currNode = tempNode.nextNode = self.headNode
        while currNode and currNode.nextNode:
            currDataVal = currNode.nextNode.data
            if currNode.data < currDataVal:
                currNode = currNode.nextNode
                continue
            if scannedNode.nextNode.data > currDataVal:
                scannedNode = tempNode
            while scannedNode.nextNode.data < currDataVal:
                scannedNode = scannedNode.nextNode
            newNode = currNode.nextNode
            currNode.nextNode = newNode.nextNode
            newNode.nextNode = scannedNode.nextNode
            scannedNode.nextNode = newNode
        self.headNode = tempNode.nextNode

    def createLinkedList(self, dataArray):
        prevNode = None
        for dataVal in dataArray:
            newNode = ListNode(dataVal)
            if not self.headNode:
                self.headNode = newNode
            else:
                prevNode.nextNode = newNode
            prevNode = newNode
```

2. Code (continued)
    a.

```
    def printLinkedList(self):
        currNode = self.headNode
        dataArray = []
        while currNode:
            dataArray.append(str(currNode.data))
            currNode = currNode.nextNode
        print('->'.join(dataArray))

inputArray = [1, 3, 2, 0]

userLinkedList = LinkedList(inputArray)
print("Linked list before sorting: ", end='')
userLinkedList.printLinkedList()
userLinkedList.insertionSort()
print("Linked list after sorting:  ", end='')
userLinkedList.printLinkedList()
```

    b. Linked list before sorting: [1, 3, 2, 0]
    Linked list after sorting: [0, 1, 2, 3]
3. Complexity Analysis
    a. The time complexity of insertionSort() is $O(n^2)$, where n is the length of the linked list. The outer while loop iterates over each node in the list, or n times. The inner loop runs if the node is out of order and will run n-1 times in the worst-case scenario. Thus, the overall time of insertionSort() is $O(n^2)$.

2. Given two sorted integer arrays A and B, merge B into A as one sorted array in O(n) time. Write the description and code of the algorithm.
    1. Conceptual Description
        a. First, get the user's two pre-sorted input arrays. Next, extend the first array by the length of the second array by appending 0's to it. Afterwards, beginning at the end of the now-extended first array, compare the last values of the original first and second arrays, and replace the digit at the currently selected position with the larger of the two. Depending on which array had the larger value, compare the next descending value with the other array's number, and repeat this process until the first array is filled with all values from both.
    2. Code
        a.

```
def arrayMerge(arr1, arr2):
    currIndexArr1 = len(arr1) - 1
    currIndexArr2 = len(arr2) - 1
    for i in arr2:
        arr1.append(0)
    lastIndex = len(arr1) - 1
    while lastIndex > -1:
        if currIndexArr1 > -1 and (currIndexArr2 == -1 or
arr1[currIndexArr1] >= arr2[currIndexArr2]):
            arr1[lastIndex] = arr1[currIndexArr1]
            currIndexArr1 -= 1
        else:
            arr1[lastIndex] = arr2[currIndexArr2]
            currIndexArr2 -= 1
        lastIndex -= 1

userArray1 = [1, 2, 5]
userArray2 = [3, 4]

print(f'Unsorted arrays: {userArray1}, {userArray2}')
arrayMerge(userArray1, userArray2)
print("Sorted array:\t", userArray1)
```

        b. Unsorted arrays: [1, 2, 5], [3, 4]
        Sorted array: [1, 2, 3, 4, 5]
    3. Complexity Analysis
        a. The time complexity of arrayMerge() is O(n+m), where n is the length of the first sorted array, and m is the length of the second sorted array. The while loop iterates n+m times until both arrays have been merged.

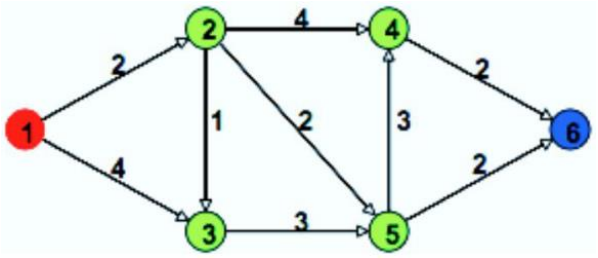
3. Implement Counting Sort. Write the description and code.
    1. Conceptual Description
        a. First, get the user's array containing only positive integers (> 0). Next, create two new arrays, the first of which will be the output array and will be the same length as the user's array, and the second of which will be the counting array and will have a length equal to that of the user's array's maximum. Next, for all integers in the user's array, add one to its corresponding position in the counting array (Ex: in the user's array, '6' will result in 1 being added to position 6 in the counting array). Next, for all numbers in the counting array, beginning with the number in the second position, add the previous number in the list to the current number selected. Finally, iterating from the end of the user's array, the selected number will be inserted into the output array at the index found by first, retrieving the number at the user array's selected number's position in the counting array, then using that number to insert the user array's selected number in that position in the output array, repeating this process for all numbers in the user's array.
    2. Code
        a.

```
def countingSort(inputArray):
    if len(inputArray) > 0 and min(inputArray) > 0:
        outputArray = [0] * len(inputArray)
        countArray = [0] * max(inputArray)
        for num in inputArray:
            countArray[num-1] += 1
        for index in range(1, len(countArray)):
            countArray[index] += countArray[index-1]
        for num in reversed(inputArray):
            outputArray[countArray[num-1]-1] = num
            countArray[num-1] -= 1
        for index in range(len(outputArray)):
            inputArray[index] = outputArray[index]

userArray = [3, 3, 1, 4, 4]

print("Unsorted array:", userArray)
countingSort(userArray)
print("Sorted array:  ", userArray)
```

        b. Unsorted array: [3, 3, 1, 4, 4]
        Sorted array: [1, 3, 3, 4, 4]
    3. Complexity Analysis
        a. The time complexity of countingSort() is O(n+m), where n is the length of the input array, and m is the range of integers in the input array. The max function takes O(n) time, the countArray initialization takes O(m) time, the first for loop takes O(n) time, the second for loop takes O(m) time, the third for loop takes O(n) time, and the last for loop takes O(n) time. Overall, the time complexity is O(n+m).

Graph Algorithms (all HAND-CALCULATION questions)

4. Find the shortest path on the graph below using Dijkstra's algorithm (Start on node "1").
   (1) Show the result table as discussed in class.
   (2) What is the shortest path from 1 to 6? What is its distance?

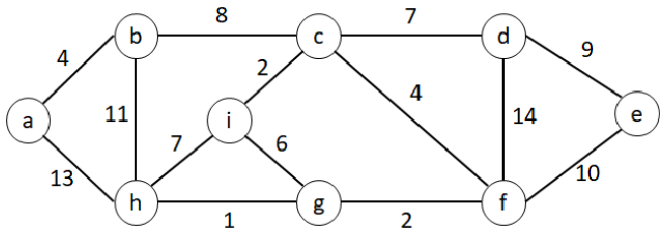


1. Result Table

Vertex	*Shortest Distance from (1)*	*Previous Vertex*
(1)	0	None, beginning node
(2)	~~∞~~ 2	(1)
(3)	~~∞~~ ~~4~~ 3	~~(1)~~ (2)
(4)	~~∞~~ 6	(2)
(5)	~~∞~~ 4	(2)
(6)	~~∞~~ 6	(5)

2. Shortest Path
   a. The shortest path from 1 to 6 is 1->2->5->6, and its distance is 6.

5. Construct the minimum spanning tree (MST) for the given graph using Kruskal's Algorithm. Show the resulting MST.



Edge Weight	*Source Vertex*	*Destination Vertex*	*Forms Cycle?*
1	(h)	(g)	N, used
2	(g)	(f)	N, used
2	(i)	(c)	N, used
4	(a)	(b)	N, used
4	(c)	(f)	N, used
6	(i)	(g)	Y, discarded
7	(h)	(i)	Y, discarded
7	(c)	(d)	N, used
8	(b)	(c)	N, used
9	(d)	(e)	N, used

Done checking since we've reached (v-1) edges, thus, the resulting MST is: