CPSC-50300 Algorithms and Data Structures

Summer 2022

Midterm Exam

**Question-1 (10 pts)**: write a Python function that takes in a list of integers (elements), and an integer number (num). The functions should count and return number of integers in elements greater than, less than and equal to num.

gt, lt, eq = find\_gt\_lt\_eq(my\_list, num = 3)

def find\_gt\_lt\_eq(lst, num): # -----> time complexity is O(n)

gt = 0

lt = 0

eq = 0

for i in lst:

if(i>num):

gt += 1

elif(i<num):

lt += 1

else:

eq += 1

return [gt, lt, eq]

gt, lt, eq = find\_gt\_lt\_eq([1, -2, 5, 0, 10, 8], -2)

print(gt, lt, eq)

**What is time complexity of your function?**

**O(n)**

**What is output of the following call?**

print(find\_gt\_lt\_eq([1, -2, 5, 0, 10, 8], -2))

5 0 1

**Question-2 (10 pts)**: **):** write a Python function that takes in a list of integers (nums) and reverses its elements. For example, calling function on nums: [1, 2, 3, 4, 5], changes nums to: [5, 4, 3, 2, 1]. Function must run in place. (Note: this is not a linked list, just a Python list). What is the time complexity of your function?

def reverse(lst): # ---------> Time Complexity is O(n)

return lst[::-1]

print(reverse([1,2,3,4]))

**Question-3 (15 pts)**: two\_sum is a Python function that takes in a list of integers (elements) and an integer number (sum) and returns True if there exists two values in elements that add up to sum, otherwise, function returns False.

Study above function and answer below questions:

* What is time complexity of two\_sum?
* Is it possible to improve on above algorithm performance (in terms of its asymptotic cost)? If yes, describe your algorithm, be detailed as much as possible.

def twoSum(arr, s): # -------> Time Complexity is O(n^2)

sums = []

for i in range(0,len(arr)):

for j in range(i+1, len(arr)):

if(arr[i]+arr[j] == s):

sums.append([arr[i], arr[j]])

return sums

print(twoSum([1,2,3,4,5,6],5))

-----------------------------------------------

def UpgradedTwoSum(arr, s): # --------> Time Complexity is O(n)

sums = []

hashTable = {}

for i in range(0, len(arr)):

sumMinusElement = s - arr[i]

if(str(sumMinusElement) in hashTable):

sums.append([arr[i], sumMinusElement])

hashTable[str(arr[i])] = arr[i]

return sums

print(UpgradedTwoSum([1,2,3,4,5,6],5))

**Question-4 (15 pts):** study the following Python function carefully and answer the following questions.

* In one statement, describe what does the function do.
* Write an equivalent function using recursion.
* Discuss advantages and disadvantages of both implementations in terms of time and space complexity.

In Python, a function is a group of related statements that performs a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.Furthermore, it avoids repetition and makes the code reusable.

def factorial(n): # ---------> Recursive function example

if n == 0:

return 1

else:

return n \* factorial(n-1)

Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input. Similarly, Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.

**Question-5 (10 pts):** Considering a singly linked list of integers implementation, assuming that elements of the list are always in ascending order, write a function (find\_median) that finds and returns the median value. The function cannot scan the list more than ONCE.

**class** SinglyList:  
 **class** \_Node:  
 **def** \_\_init\_\_(self, e, next):  
 self.\_element = e  
 self.\_next = next  
  
 **def** \_\_init\_\_(self):  
 self.\_head = self.\_tail = **None**

**def** find\_median(self):

class LinkedList:

# Function to initialize head

def \_\_init\_\_(self):

self.head = None

# Function to insert a new node at the beginning

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

self.head = new\_node

# Print the linked list

def printList(self):

node = self.head

while node:

print(str(node.data) + "->", end="")

node = node.next

print("NULL")

# Function that returns middle.

def find\_median(self):

# Initialize two pointers, one will go one step a time (slow), another two at a time (fast)

slow = self.head

fast = self.head

# Iterate till fast's next is null (fast reaches end)

while fast and fast.next:

slow = slow.next

fast = fast.next.next

# return the slow's data, which would be the middle element.

print("The median element is ", slow.data)

# Code execution starts here

if \_\_name\_\_=='\_\_main\_\_':

# Start with the empty list

llist = LinkedList()

for i in range(5, 0, -1):

llist.push(i)

llist.printList()

llist.find\_median()