CPSC-50300 Algorithms and Data Structures

Spring 2022

Midterm Exam

**Question-1 (10 pts)**: write a Python function that takes in a list of integers and returns maximum and minimum values in the list as a tuple. Hint (can be done in one pass, you are not allowed to use built-in min and max functions.). What is the time complexity of your functions?

max, min = find\_max\_min(my\_list):

**Answer:**

**Python Function:**

def find\_max\_min(my\_List):

maximum\_number = minimum\_number = my\_List[0]

for number in my\_List:

if number > maximum\_number:

maximum\_number = number

elif number < minimum\_number:

minimum\_number = number

return (maximum\_number, minimum\_number)

my\_List= []

for i in range(5):

number = int(input("Enter the numbers for the list: "))

my\_List.append(number)

print(find\_max\_min(my\_List))

**Time Complexity:** Time complexity of my function is **O(n)** because we are using a for loop traversing the complete list.

**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?

**Answer:**

**Python Function:**

def Rerverse\_a\_List(my\_List):

reversed\_List = []

reverse\_index = len(my\_List)

for i in my\_List:

reverse\_index = reverse\_index - 1

reversed\_List.append(my\_List[reverse\_index])

return reversed\_List

my\_List= []

for i in range(5):

number = int(input("Enter the numbers for the list: "))

my\_List.append(number)

print("Original List is: ",my\_List)

print("Reversed List is: ", Rerverse\_a\_List(my\_List))

**Time Complexity:** Time complexity of my function is **O(n)** because we are using a for loop traversing the complete list to reverse each element of index in the list.

**Question-3 (10 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.

**def** two\_sum(elements: List[int], sum: int):  
  
 **for** i **in** range(len(elements)):  
 **for** j **in** range(i + 1):  
 **if** elements[i] + elements[j] == num:  
 **return True  
  
 return False**

Study above function and answer below questions:

1. What is time complexity of two\_sum?
2. 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.

**Answer:**

**A:**

Overall time complexity of above function is **O(n2).**

**Time Complexity:**

|  |  |
| --- | --- |
| **Scenario** | **Time Complexity** |
| Average Case | O(nlog(n)) |
| Best Case | O(1) |
| Worst Case | O(n)2 |

**B:**

According to my point of view we can improve this above algorithm.

Optimized Algorithm:

1. First of all we have to sort the array in ascending order.
2. We have to take 2 variables one should be l=0 and J=n-1 where n is total elements, I is the first element and J is the last element.
3. Make a loop while(I<J):
4. Then we have to apply a condition within the loop that if(elements[l]+elements[J]==num) then return True.
5. Then if the above condition does not meet the requirement then we use else if condition that (elements[l]+elements[J]
6. Otherwise we have to decrement in the J by one time.
7. At the end if we did not get en element then we have to return False.
8. The time complexity of above algorithm is depends on sorting algorithm we use in case if we use merge sort algorithm then Time complexity is o(nlogn) and space complexity is o(n).

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

1. In one statement, describe what does the function do.

**def** fun(nums: List[int]):  
 n = nums[0]  
 **for** v **in** nums:  
 if v > n:

n = v  
 **return** n

1. Write an equivalent function using recursion.
2. Discuss advantages and disadvantages of both implementations in terms of time and space complexity.

**Answer:**

**A:**

This function is used to find the maximum value from a list and return the maximum value.

**B:**

**Python Function using recursion:**

def fun(my\_List):

lenght = len(my\_List)

if lenght == 1:

return my\_List[0]

else:

Maximum\_number = fun(my\_List[1:])

if Maximum\_number > my\_List[0]:

return Maximum\_number

else:

return my\_List[0]

my\_List = [12,34,54,87,2,8,112,0,456]

print("The maximum number in the list is: ", fun(my\_List))

**C:**

**Advantages and disadvantages of both implementations:**

1. Recursion is when a function calls itself within its own code, the instructions it contains are executed again. While Iteration occurs when a loop, such as a "for" loop or a "while" loop, performs the same set of instructions over and over again.
2. We employ function calls to execute statements repeatedly inside the function body in recursion. On the other hand, we utilize loops like "for" and "while" to perform the same thing in iteration.
3. Recursive code are smaller and easier to understand. On the other hand, It is easier to optimize iterative codes, and they generally have polynomial time complexity.
4. The simplicity of recursion comes at the cost of time and space efficiency.
5. It’s difficult to traverse trees/graphs using loops.
6. Iteration is faster and more efficient than recursion. It is difficult to optimize a recursive code, and they generally have higher time complexity than iterative codes due to overlapping sub problems.

**Question-5 (15 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.

**Answer:**

**Python Function:**

class SinglyList:

class \_Node:

def \_\_init\_\_(self, e, next):

self.\_element = e

self.\_next = next

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

self.\_head = self.\_tail = None

def add\_element(self, p):

new\_Node = self.\_Node(p,None)

new\_Node.\_next = self.\_head

self.\_head = new\_Node

def find\_median(self):

count = 0

median = self.\_head

while median is not None:

count = count + 1

median = median.\_next

median = self.\_head

for i in range(count // 2):

median = median.\_next

return median.\_element

Singly\_Linked\_List = SinglyList()

Singly\_Linked\_List.add\_element(12)

Singly\_Linked\_List.add\_element(15)

Singly\_Linked\_List.add\_element(6)

print("The median of Linked List is: ", Singly\_Linked\_List.find\_median())

**Time Complexity:** The time complexity of find\_median function is O(n+n) complexity which is O(2n) but it becomes **O(n)** because we are using two loops.