Midterm – DSA 456 – February 24th , 2025

- Use this word document to provide ALL your answers (code and text)
- Exam starts at 7pm and is done at 10pm.

Section 1 – Python (20 marks)

Q1.

Write a Python function that takes a list as an input and returns the average absolute deviation of its values. The average absolute deviation is defined as the mean of the absolute differences between each element and the mean of the list.

Example:

For the list [5, 4, 8, 2, 9, 1, 3], the mean is 4.57, and the average absolute deviation is 2.24 (rounded to two decimal places).

You can use Python's built-in functions for calculations.

def avg_absolute_deviation(lst):

ANSWER:

```
def avg_absolute_deviation(lst):
    mean = sum(lst) / len(lst)
    abs_diff = [abs(x - mean) for x in lst]
    return round(sum(abs_diff) / len(lst), 2)
```

Q2.

Write a Python function that gets n and returns the nth value in the "Cool Function Sequence." The sequence is defined as:

C(n)=2∗C(n−1)+C(n−2)C(n) = 2 * C(n-1) + C(n-2)

where

C(0)=1C(0) = 1 and C(1)=1C(1) = 1.

Use your function to generate and provide the first 10 values of the "Cool Function Sequence."

def cool(n):

ANSWER:

```
def cool(n):
    if n == 0 or n == 1:
        return 1
    return 2 * cool(n - 1) + cool(n - 2)
```

Section 2 – function Analysis (30 marks)

Q6.
Analyze the function developed in Q2 and determine its time complexity T(n) and Big-O performance. How does its performance compare to quicksort? Is it faster or slower than computing the nth term of the Tribonacci sequence, defined as:
T(n)=T(n−1)+T(n−2)+T(n−3), where T(0)=0,T(1)=1,T(2)=1

ANSWER:
Step 1: Establish Variables and Functions

- Let n represent the input number.
- Let T(n) represent the number of operations required.

Step 2: Count Your Operations
Each call to cool(n) performs the following:

- Base Case Check: 1 operation (constant time).
- Recursive Calls:
  - Calls cool(n-1) ⟶ contributes T(n-1) operations.
  - Calls cool(n-2) ⟶ contributes T(n-2) operations.
- Multiplication & Addition: 1 operation.

Step 3: Establish Mathematical Expression
The number of operations follows the same pattern as the Fibonacci sequence, which expands at: $T(n) \approx 2^n$

Step 4: Simplify the Equation
$T(n) = O(2^n)$

Q7.
What is the time complexity T(n)T(n) and Big-O performance of the following function? This function takes a list of n numbers as input and finds all pairs whose sum is a prime number.

```
import math

def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
        return False
    return True

def find_prime_pairs(lst):
    count = 0
    for i in range(len(lst)):
        for j in range(i + 1, len(lst)):
            if is_prime(lst[i] + lst[j]):
                count += 1
    return count
```

Compare its performance with the time complexity of selection sort and insertion sort.

ANSWER:

Step 1: Establish Variables and Functions

- Let n be the number of elements in the list.
- Let T(n) be the total number of operations.
- The function find_prime_pairs(lst) checks all the pairs (i, j) in the list, where i < j, and counts how many pairs whose sum is a prime number.

Step 2: Count Your Operations

- Outer loop (for i in range(len(lst))) - Runs n times.
- Inner loop (for j in range(i + 1, len(lst))) - Runs approximately (n-1), (n-2), ..., 1 times, which sums to $n(n-1)/2 \approx O(n^2)$.
- Checking is_prime(lst[i] + lst[j]) - worst case is $O(\sqrt{n})$.
- Total operations: $T(n) = O(n^2 \cdot \sqrt{n})$

Step 3: Establish Mathematical Expression

$T(n) = O(n^2\sqrt{n})$

Step 4: Compare to Selection Sort and Insertion Sort

- Selection Sort - $O(n^2)$.
- Insertion Sort - $O(n^2)$ (worst case)
- Find Prime Pairs - $O(n^2\sqrt{n})$ (slower than both selection and insertion sort).

Q8.

Suppose after analyzing a recursive function, you determine that:

T(n)=3+3T(n/3)

What is the Big-O complexity of this function? Is it more efficient or less efficient than quicksort? Is it faster or slower than finding the median of an unsorted list of size n?
Hint: Use the recurrence tree method or the Master Theorem for analysis.

ANSWER:
Using the Master Theorem $T(n) = aT(n/b) + f(n)$ where:

- a = 3 (recursive calls)
- b = 3 (problem size is reduced by)
- f(n) = 2 (additional work)

Since $log_3 3 = 1$ and **f(n)** grows slower than n, the complexity follows $O(n)$.

Therefore Quicksort is O(nlogn) which is slower and finding the median is the same efficiency at O(n).

Section 3 - Linked Lists (30 marks)

Q6.
What are advantages and disadvantages of using singly linked lists vs. doubly linked lists? Please elaborate

ANSWER:
A singly linked list is more memory-efficient because each node only has one pointer to the next node, however it can only be traversed in one direction, and deleting a node in the middle requires finding the previous node first, making some operations slower.

A doubly linked list, unlike the singly linked list, has two pointers (next and previous), allowing for easier backward traversal and faster deletion, but it uses more memory and requires extra pointer updates, making it more complex.

A singly linked list is better for simple, memory-efficient tasks, while a doubly linked list is more useful when frequent insertions and deletions are needed in both directions.

Q7.
Assume that a linked list is declared as follows:

```
class LList:
  class Node:
    def __init__(self, data, next=None):
      self.data = data
      self.next = next

  def __init__(self):
    self.front = None
    self.back = None
```

Develop a two cloning function:

```
def clone_range_index_reverse(self, myLList, i , j):
def clone_range (self, myLList, min , max):
```

that gets a linked list myLList and builds the linked list based on the given myLList by:
clone_range_index_reverse: copying over the ith, (i+1)th, (i+2)th , …, jth element of myLList into the new list in reverse (starting at index ith). If i=len(myLList), and j=1 the new linked list will be identical to myLList.
clone_range : copying over all elements of myLlist that are larger or equal to min and smaller or equal to max

Hint: You can first implement a simple insert function for your linked list that gets a node (or data) and insets it in the list. Then in the clone_range function, iterate the given myLList and one by one insert the qualified data into

ANSWER

```python
class LList:
    class Node:
        def __init__(self, data, next=None):
            self.data = data
            self.next = next

    def __init__(self):
        self.front = None
        self.back = None

    def insert(self, data):
        new_node = self.Node(data)
        if self.front is None:
            self.front = self.back = new_node
        else:
            self.back.next = new_node
            self.back = new_node

    def clone_range_index_reverse(self, myLList, i, j):
        new_list = LList()
        stack = []
        current = myLList.front
        index = 0
        while current:
            if i <= index <= j:
                stack.append(current.data)
            current = current.next
            index += 1
        while stack:
            new_list.insert(stack.pop())
        return new_list

    def clone_range(self, myLList, min_val, max_val):
        new_list = LList()
        current = myLList.front
        while current:
            if min_val <= current.data <= max_val:
                new_list.insert(current.data)
            current = current.next
        return new_list
```

Section 4 - Tables (20 marks)

Q8. Implementing a double hashed table

Consider the following double hash function:
hash1(key) = key % 13
hash2(key) = 9 - (key % 9)
Assume that the table of length 11 is already partially populated as

Index	Key	Value
0	11	Apple
1		
2	13	Banana
3	25	Cherry
4		
5		
6	22	Date
7		
8		
9	39	Elderberry
10	21	Fig

Using the double hash functions above, do the following two insertions
Insert(Key = 0, value = Plum)
Insert(Key = 9, value = Strawberry)
Just provide a simple explanation and populate the values in the table above. No coding required.

GOOD LUCK!

ANSWER:

Index	Key	Value
0	11	Apple
1		
2	13	Banana
3	25	Cherry
4		
5	9	Strawberry
6	22	Date
7	0	Plum
8		
9	39	Elderberry
10	21	Fig

(Key = 0, value = Plum) was first assigned to index 0, but since it was occupied, double hashing moved it to index 9 (also occupied), then to index 7, where it was inserted.
(Key = 9, value = Strawberry) was initially assigned to index 9, which was occupied, so it was then moved to index 7 (also occupied), and finally placed at index 5, which was empty.