ASSIGNMENT - 2

Q1. Determine the time complexity of the following algorithm and provide a detailed explanation of your analysis

```
def sum_elements(lst):
    total = 0
    for num in lst:
        total += num
    return total
```

Solution:

A) Initialization (total = 0):
This is a constant time operation, so it takes O(1).

B) Loop Over List (for num in lst):

- The loop iterates over each element of the input list lst.
- If the list contains n elements, the loop will run n times.
- Inside the loop, there is a constant time operation total += num, which adds the value of num to the total. This operation takes O(1) time for each iteration.

C) Return the Total (return total):
Returning the value of total is a constant time operation, so it takesO(1).

Time Complexity:

The loop runs n times, where n is the length of the list lst. Each iteration of the loop involves a constant-time operation (total += num), so the total time spent inside the loop is O(n).

The rest of the operations (initialization and returning the result) are constant-time operations, so they do not affect the overall time complexity.

Thus, the time complexity of this algorithm is O(n), where n is the number of elements in the list lst

Q2. Determine the time complexity of the following algorithm and provide a detailed explanation of your analysis

```
def bubble_sort(lst):
    n = len(lst)
    for i in range(n):
        for j in range(0, n-i-1):
            if lst[j] > lst[j+1]:
                lst[j], lst[j+1] = lst[j+1], lst[j]
    return lst
```

Solution:

1. **Getting the Length of the List (n = len(lst))**:
    - This takes O(1)O(1)O(1), as len(lst) is a constant-time operation.
2. **Outer Loop (for i in range(n))**:
    - This loop runs **n times**, where n is the length of the list. In each iteration of this outer loop, the size of the inner loop decreases by 1.
3. **Inner Loop (for j in range(0, n-i-1))**:
    - The inner loop runs **n-i-1** times for each iteration of the outer loop.
    - In the first pass, the inner loop runs n−1n-1n−1 times; in the second pass, it runs n−2n-2n−2 times, and so on, until it runs only once in the last pass.
    - So, the number of iterations of the inner loop decreases linearly with each pass of the outer loop.
4. **Comparison and Swap (if lst[j] > lst[j+1])**:
    - This is a constant-time operation, O(1)O(1)O(1), for each comparison and swap.

Total Time Complexity:

- For each iteration of the outer loop, the inner loop runs approximately n−1n-1n−1, n−2n-2n−2, ..., 1 times. The total number of comparisons and swaps across all iterations is:

(n−1)+(n−2)+⋯+1=n(n−1)2(n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2}(n−1)+(n−2)+⋯+1=2n(n−1)

- This is equivalent to O(n2)O(n^2)O(n2), as the constant factors and lower-order terms can be ignored in Big-O notation.

Thus, the **time complexity of bubble sort is O(n2)O(n^2)O(n2)** in the worst and average cases.

Best Case Time Complexity:

- The best case occurs when the list is already sorted. In that case, bubble sort still runs through all the iterations but doesn't perform any swaps.
- Even in the best case, without any optimization, the algorithm will still run in O(n2)O(n^2)O(n2) time due to the nested loops.