**DEPARTMENT OF ARTIFICIAL INTELLIGENCE**  
**QUAID E AWAM UNIVERSITY OF ENGINEERING SCIENCE AND TECHNOLOGY**

**DATA STRUCTURES**

Dated: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Roll No(s): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **Lab 02 – Complexity Analysis**

## **Lab Objectives**

- Understand time and space complexity of algorithms.

- Measure execution time of searching and sorting operations in Python.

- Compare algorithm performance for O(1), O(log n), and O(n) operations.

## **Background**

In data structures and algorithm design, complexity analysis helps determine how efficiently an algorithm performs as the input size increases. Python provides built-in tools like the 'time' and 'bisect' modules to measure and compare performance.  
  
Common complexity classes include:  
• O(1) → Constant Time (e.g., accessing an element in a list)  
• O(log n) → Logarithmic Time (e.g., binary search)  
• O(n) → Linear Time (e.g., traversing a list or linear search)

## **Lab Tasks**

### **Task 1: Measure Execution Time of Searching Algorithms**

Write a Python program to measure and compare the execution time of:  
- Linear Search (O(n))  
- Binary Search (O(log n))  
  
For arrays/lists of increasing sizes (e.g., 1,000; 10,000; 100,000). Print the time taken by each search method and compare performance.

import time, bisect  
  
def linear\_search(arr, x):  
 for i in arr:  
 if i == x:  
 return True  
 return False  
  
def binary\_search(arr, x):  
 index = bisect.bisect\_left(arr, x)  
 return index < len(arr) and arr[index] == x  
  
sizes = [1000, 10000, 100000]  
for n in sizes:  
 data = list(range(n))  
 key = n - 1  
  
 start = time.time()  
 linear\_search(data, key)  
 print(f"Linear Search ({n}):", time.time() - start, "sec")  
  
 start = time.time()  
 binary\_search(data, key)  
 print(f"Binary Search ({n}):", time.time() - start, "sec")

### **Task 2: Compare O(1), O(log n), and O(n) Operations**

import time, bisect  
  
data = list(range(1\_000\_000))  
  
# O(1)  
start = time.time()  
x = data[500\_000]  
print("O(1):", time.time() - start)  
  
# O(log n)  
start = time.time()  
bisect.bisect\_left(data, 500\_000)  
print("O(log n):", time.time() - start)  
  
# O(n)  
start = time.time()  
for x in data:  
 if x == 500\_000:  
 break  
print("O(n):", time.time() - start)

Observe that:  
- O(1) is nearly instantaneous.  
- O(log n) is very fast even for large lists.  
- O(n) grows significantly with data size.

### Task 3 (Optional): Sorting Algorithms Comparison

Measure the execution time of:  
- Bubble Sort (O(n²))  
- Built-in sorted() (Timsort, O(n log n))  
  
Compare results for arrays of size 1000, 5000, and 10000.

## Questions

1. What does algorithm complexity describe?

2. Why does binary search outperform linear search?

3. Which factors can affect timing results other than algorithm efficiency?

4. Write a Python snippet that measures the time of list reversal using slicing ([::-1]).

5. Compare the growth rate of O(1), O(log n), and O(n) in your own words.