Department of Computer Science and Engineering
Chandpur Science and Technology University

Course Code: CSE 2201	Credits: 1.50
Course Name: Algorithm Design and Analysis	Semester: 2-2

Lab 03

Divide and Conquer: Algorithms for Sorting and Searching

I. Learning Objectives

By the end of this lab, students should be able to:

- Understand the Divide and Conquer paradigm.
- Implement and compare key algorithms: Merge Sort, Quick Sort, Binary Search.
- Analyze recursive vs. iterative implementations.
- Explore advanced problems: Closest Pair of Points.
-

II. Lesson Fit:

Prerequisite: C/C++, Data Structure

III. Theory Recap:

What is Divide and Conquer?

Divide and Conquer is a **problem-solving paradigm** in computer science and algorithm design that:

- **Divides** the problem into smaller subproblems of the same type.
- **Conquers** the subproblems recursively.
- **Combines** the results of subproblems to form the solution of the original problem.

Three Core Steps

1. **Divide**:
   Split the original problem into smaller subproblems.
   Example: In merge sort, divide an array of n elements into two halves.
2. **Conquer**:
   Solve each subproblem recursively.
   If the subproblem size is small enough (base case), solve directly.
3. **Combine**:
   Merge the subproblem results to form the final solution.
   Example: In merge sort, merging the sorted halves into a fully sorted array.

Why Use Divide and Conquer?

- Efficient on large datasets.
- Often reduces time complexity.
- Promotes recursive thinking.
- Easily parallelizable in many cases.
- Helps break down complex problems into manageable parts.

Classic Examples

Algorithm	**Problem Type**	**Time Complexity**
Merge Sort	Sorting	O(n log n)
Quick Sort	Sorting	O(n log n) avg
Binary Search	Searching	O(log n)
Closest Pair of Points	Geometry	O(n log n)
Strassen's Multiplication	Matrix	~O(n^2.81)
Karatsuba's Algorithm	Large Integer Multiplication	O(n^1.58)

Lab 2 Activity List

Experiment # 1: ***Implement Quick Sort using Iterative approach and Recursive approach and compare the time complexities.***

📊 Time Complexity:

- Best Case:
- Worst Case:
- Average Case:

🧮 Space Complexity:

📊 Comparison Table (Empirical)

n (array size)	Iterative Approach	Recursive Approach

You can plot this in a graph (X: Input size n, Y: Steps taken) to **visually compare** the growth rate.

Report:

The report should cover the following

Name of the Experiment

1. Objective
2. Algorithm
3. Theoretical Solution of given problem
4. Practical Work:
    a. Pseudocode
    b. Source Code in C/CPP/Python
5. Analysis Table

Algorithm	Best Case	Worst Case	Avg Case	Space

6. Observations
7. Challenges
8. Conclusion

📷 Attachments:

- Screenshot of program output.

- Manual step count snapshots.

- Complexity graph (drawn or plotted).