**Submission Guidelines**

**• Handwritten solutions for conceptual questions must be submitted.**

**• A Word document containing:**

**o Code implementations.**

**o Execution outputs (screenshots or logs).**

**o Performance comparisons for coding exercises.**

**• Evaluation will be based on a quiz or viva, so make sure to solve the assignment yourself.**

**Question 1 Parallel Bucket-Sort Decomposition in Distributed Memory Systems:**

You have an input array A of n random integers ranging from 1 to r.

The goal: Compute array B, where B[i] gives the count of integer i in A.

**Tasks:**

1. Propose two decomposition strategies for parallelizing this problem. Compare them in terms of:

* Communication overhead
* Load balancing
* Scalability

2. Detail the input partitioning strategy (partition A) and how you would map the processes.

Describe the parallel algorithm workflow.

3. Detail the output partitioning strategy (partition B) and map it to p processes. Explain the

parallel algorithm in brief.

4. Discuss the advantages and disadvantages of both decompositions. Consider:

* p: Number of processors
* r: Range of integers
* n: Size of input array
* Communication cost between processors

**Question 2 Advanced Coding Challenges**

**Exercise 1: Matrix Multiplication with C++ Threads (std::thread or std::async)**

**Task:**

**Implement a parallel matrix-matrix multiplication program that distributes computation using C++ threads (std::thread).**

* **Divide the matrix into equal parts and process them using multiple threads.**
* **Compare execution time between normal multiplication and thread-based multiplication.**
* **Each student must generate a unique matrix size N × N using their roll number.**

**Customization Rules:**

* **Set matrix size N as:**

**int N = (roll\_number % 5) + 5; // Ensures different students get different sizes**

* **Split the work across two or more threads handling different matrix parts.**

**Example (For Roll Number: 2XF-9187)**

* **N = (9187 % 5) + 5 = 12**
* **num\_threads = (9187 % 3) + 2 = 3**
* **The matrix size will be 12 × 12, and 3 threads will be used for computation.**

**Pseudo-code outline**

**// 1. Generate a matrix of size (N × N) using roll number formula**

**// 2. Use std::thread or std::async to parallelize matrix multiplication**

**// 3. Compare execution times between normal and threaded implementations**

**Exercise 2: Merge Sort with Dynamic Task Parallelism (std::thread)**

**Task:**

**Modify Merge Sort to:**

* **Dynamically create threads for sorting subarrays using std::thread.**
* **Implement adaptive parallelism, where small subarrays are sorted sequentially.**
* **Compare execution time against a traditional recursive Merge Sort.**
* **Each student must modify their sorting algorithm based on their roll number.**

**Customization Rules:**

* **If roll number is even, use insertion sort for small subarrays.**
* **If roll number is odd, use quick sort for small subarrays.**

**Example (For Roll Number: 21F-9187)**

* **9187 is odd, so quick sort will be used for small subarrays.**
* **Sorting will be divided across multiple threads dynamically.**

**Pseudo-code outline**

**// 1. If roll number is even, use insertion sort for small arrays**

**// 2. If roll number is odd, use quicksort for small arrays**

**// 3. Implement std::thread to parallelize sorting tasks dynamically**

**// 4. Compare execution times**

**Question 3 For the task graphs given in Figure 2 (below), determine the following:**

• Maximum degree of concurrency

• Critical path length

• Maximum achievable speedup over a single processor assuming infinite processors are

available

• Minimum number of processors needed to achieve the maximum possible speedup

• Maximum achievable speedup when using 2, 4, and 8 processors

Note: Assume that each task node takes an equal amount of time to execute i.e : 4.

