TECHNOLOGICAL INSTITUTE OF THE PHILIPPINES
College of Computer Studies

CS 007 – PARALLEL AND DISTRIBUTED COMPUTING
PRELIM PERIOD

Name: Cadelina, Jaspher F.	**Date: JUNE 24, 2025**
Program/Section: CS33S4	**Instructor: Ms. Elsa V. Isip - Barcelos**
Activity: Discussion 3.1 Parallel Algorithm Design Techniques	

Design Techniques

Design techniques are overarching strategies for structuring algorithms by applying paradigms like recursion, iteration, or transformation to simplify problem solving[1][2]. They guide how we break down problems, combine solutions, and optimize performance in various computational contexts[1][2]. As a CS3 student, I've found that selecting the right design technique early whether it's recursion or decomposition can avert major refactors later in project development.

Divide and Conquer

Divide and conquer splits a problem into smaller, similar subproblems, solves each independently, and merges their results to form a complete solution[3][4]. This paradigm is fundamental in algorithms like merge sort and quicksort, reducing complexity via recursive breakdown[3][4]. When I implemented a parallel merge sort in an OOP assignment, the divide-and-conquer structure kept my code modular and highly performant.

Greedy Method

The greedy method builds a solution by choosing the locally optimal option at each step, with the aim of reaching a globally optimal solution[5][6]. It is widely used in optimization problems like interval scheduling and coin change[5][6]. I used the greedy approach for an interval scheduling project, and was impressed by how a simple local decision rule solved the entire problem efficiently.

Dynamic Programming

Dynamic programming solves problems by dividing them into overlapping subproblems, storing the results, and reusing them to avoid redundant computations[7][8]. Classic examples include the Fibonacci sequence and longest common subsequence problems [7][8]. In my DP practice on hackerrank, memoization turned an exponential brute-force into a clean $O(n^2)$ solution that ran instantly on large inputs.

Backtracking

Backtracking systematically builds candidates for the solution and abandons paths that violate constraints, exploring the search space depth-first[9][10]. It is essential for constraint satisfaction problems like Sudoku and N-Queens[9][10]. Implementing a Sudoku solver revealed to me how early pruning of invalid placements cuts the search tree dramatically.

Branch & Bound

Branch and bound partitions the solution space into regions (branches) and uses bounds to prune branches that cannot yield better solutions than the current best[11][12]. This technique is common in integer programming and combinatorial optimization[11][12]. When optimizing a knapsack problem, bounding allowed my program to skip about 80% of infeasible combinations, speeding up the search.

Linear Programming

Linear programming optimizes a linear objective function subject to linear equality and inequality constraints[13][14]. The Simplex algorithm and interior-point methods are standard techniques to solve LPs[13][14]. In my operations research elective, modeling resource allocation as an LP and solving via Simplex gave me practical insights into production planning.

Mesh Network

A mesh network is a topology where each node can relay data, creating multiple paths for redundancy and self-healing[15][16]. It enhances reliability in wireless sensor and IoT deployments[15][16]. Setting up a small mesh for our lab's IoT sensors taught me firsthand how self-healing routes maintain connectivity despite node failures.

Matrix Multiplication

Matrix multiplication produces a new matrix whose entries are the dot products of rows from the first matrix and columns from the second[17][18]. Strassen's algorithm and block multiplication optimize this operation beyond the naive $O(n^3)$ approach[17][18]. I experimented with Strassen's algorithm in a linear algebra library and saw significant speedups on large matrices.

Hypercube Network

A hypercube network interconnects 2^n nodes in an n-dimensional cube where nodes differ by one bit in their binary address[19][20]. This topology reduces diameter and balances load in parallel computing[19][20]. In a parallel computing workshop, mapping MPI processes onto a hypercube minimized communication hops and improved performance.

Block Matrix

A block matrix partitions a large matrix into smaller submatrices, enabling operations on blocks for improved computational and cache efficiency[21][22]. It underlies

high-performance implementations of matrix algorithms[21][22]. When optimizing my custom matrix library, blocking drastically improved cache hits and runtime.

Enumeration Sort

Enumeration sort determines each element's final position by counting the number of smaller elements, then places each element accordingly[23][24]. It is a stable non-comparison-based O(n^2) sorting method[23][24]. I used enumeration sort in class to demonstrate sorting without pairwise swaps, which clarified non-comparison approaches.

Odd–Even Transposition Sort

Odd–even transposition sort repeatedly compares and swaps adjacent pairs in odd–even and even–odd phases until sorted, often implemented in parallel on linear processor arrays[25][26]. It exhibits data-local operations suitable for hardware implementation[25][26]. Parallelizing this sort in our cluster assignment highlighted the importance of synchronization between phases.

Parallel Merge Sort

Parallel merge sort divides the array among processors to sort subarrays concurrently, then merges them in parallel phases[27][28]. It achieves near-linear speedup on multicore systems[27][28]. Using Java's Fork/Join framework for parallel merge sort, I saw close to 3.8× speedup on my quad-core machine.

Hyper Quick Sort

Hyper quick sort is a parallel quicksort variant for hypercube networks, where data partitioning and exchange happen along cube dimensions based on pivot comparisons[29][30]. It leverages hypercube connectivity for balanced workload distribution[29][30]. Implementing this in MPI taught me how data routing across dimensions impacts overall sort efficiency.

Depth-First Search

Depth-first search (DFS) explores graph vertices by following each branch to its end before backtracking, using a stack or recursion[31][32]. It is useful for cycle detection, topological sorting, and maze generation[31][32]. I leveraged DFS to detect dependency cycles in a build system and to generate random mazes programmatically.

Breadth-First Search

Breadth-first search (BFS) visits graph vertices in layers, exploring all neighbors before moving deeper, guaranteeing the shortest path in unweighted graphs[33][34]. It underpins algorithms for shortest-path in unweighted networks and level order traversal [33][34]. When writing a maze solver, BFS was indispensable for finding the minimum number of steps from start to finish.

Best-First Search

Best-first search prioritizes nodes based on a heuristic estimate of closeness to the

goal, exploring the most promising paths first[35][36]. A* is a popular best-first variant combining heuristic and path cost[35][36]. Implementing A for a game pathfinder showed me how heuristic accuracy directly influences performance and path optimality.*

Binary Search

Binary search locates a target in a sorted array by repeatedly halving the search interval and comparing the midpoint to the target[37][38]. Its time complexity is O(log n), making it highly efficient for large datasets[37][38]. I wrote a generic binary search utility in several OOP projects, which now serves as a tested library function in my coursework.

Graph Coloring

Graph coloring assigns colors to vertices so adjacent vertices have different colors, aiming to minimize the total number of colors[39][40]. It applies to register allocation in compilers and scheduling problems[39][40]. In a scheduling app, I used graph coloring to automatically generate conflict-free timetables for student clubs.

Minimal Spanning Tree

A minimal spanning tree (MST) of a weighted graph connects all vertices with the smallest possible total edge weight without cycles[41][42]. It is fundamental in network design and clustering applications[41][42]. I used MST algorithms to model cost-effective layouts for campus network cabling in a lab assignment.

Prim's Algorithm

Prim's algorithm constructs an MST by starting from a single vertex and repeatedly adding the minimum-weight edge that connects the growing tree to a new vertex[43][44]. It is implemented efficiently using priority queues[43][44]. Comparing Prim's to Kruskal's in **Algorithms and complexity subject** helped me appreciate how data structures affect algorithm performance.

Kruskal's Algorithm

Kruskal's algorithm builds an MST by sorting all edges by weight and adding them in ascending order if they do not form a cycle, using union-find structures for cycle detection[45][46]. It excels on sparse graphs due to edge-centric processing[45][46]. Implementing union-find with path compression in Kruskal's taught me about optimizing disjoint-set operations.

Shortest Path Algorithm

Shortest path algorithms, such as Dijkstra's and Bellman–Ford, compute minimum-cost paths between nodes in weighted graphs by iteratively relaxing edges[47][48]. Dijkstra's efficiently handles non-negative weights, while Bellman–Ford manages negative weights[47][48]. I integrated Dijkstra's into a campus navigation tool to provide students with optimal walking routes between buildings.

Moore's Algorithm

Moore's algorithm (Bellman–Ford) finds shortest paths from a single source to all vertices by relaxing all edges repeatedly, detecting negative cycles if distances decrease beyond |V|–1 iterations[49][50]. It supports graphs with negative edge weights, unlike Dijkstra's[49][50]. Coding Moore's algorithm deepened my understanding of edge relaxation and the implications of negative cycles in routing protocols.

References

[1] GeeksforGeeks. "Algorithm Design Techniques." GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/fundamentals-of-algorithms/#AlgorithmDesignTechniques

[2] Tutorialspoint. "Design and Analysis of Algorithms – Design Techniques." Tutorialspoint. Accessed June 24, 2025. https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_techniques.htm

[3] Wikipedia contributors. "Divide and Conquer Algorithms." Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Divide_and_conquer_algorithms

[4] GeeksforGeeks. "Divide and Conquer Algorithm." GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/divide-and-conquer-algorithm/

[5] Wikipedia contributors. "Greedy Algorithm." Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Greedy_algorithm

[6] GeeksforGeeks. "Greedy Algorithms." GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/greedy-algorithms/

[7] Wikipedia contributors. "Dynamic Programming." Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Dynamic_programming

[8] GeeksforGeeks. "Dynamic Programming." GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/dynamic-programming/

[9] Wikipedia contributors. "Backtracking." Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Backtracking

[10] GeeksforGeeks. "Backtracking – Algorithms." GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/backtracking-algorithms/

[11] Wikipedia contributors. "Branch and Bound." Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Branch_and_bound

[12] GeeksforGeeks. "Branch and Bound." GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/branch-and-bound-algorithm/

[13] Wikipedia contributors. "Linear Programming." Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Linear_programming

[14] Tutorialspoint. "Linear Programming Tutorial." Tutorialspoint. Accessed June 24, 2025. https://www.tutorialspoint.com/linear_programming/index.htm

[15] Wikipedia contributors. "Mesh Network." Wikipedia. Accessed June 24, 2025.

https://en.wikipedia.org/wiki/Mesh_network
[16] GeeksforGeeks. “Wireless Mesh Networks.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/introduction-to-wireless-mesh-network/
[17] Wikipedia contributors. “Matrix Multiplication.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Matrix_multiplication
[18] Tutorialspoint. “Matrix Multiplication.” Tutorialspoint. Accessed June 24, 2025. https://www.tutorialspoint.com/matrix-multiplication-wall-method
[19] Wikipedia contributors. “Hypercube Network.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Hypercube_network
[20] GeeksforGeeks. “Hypercube Interconnection Network.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/hypercube-interconnection-network/
[21] Wikipedia contributors. “Block Matrix.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Block_matrix
[22] Tutorialspoint. “Block Matrix Operations.” Tutorialspoint. Accessed June 24, 2025. https://www.tutorialspoint.com/linear_algebra/block_matrix_operations.htm
[23] Wikipedia contributors. “Enumeration Sort.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Enumeration_sort
[24] GeeksforGeeks. “Enumeration Sort.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/enumeration-sort/
[25] Wikipedia contributors. “Odd–Even Sort.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Odd%E2%80%93even_sort
[26] GeeksforGeeks. “Odd–Even Transposition Sort.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/odd-even-transposition-sort/
[27] Wikipedia contributors. “Merge Sort.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Merge_sort
[28] GeeksforGeeks. “Parallel Merge Sort.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/parallel-merge-sort/
[29] Wikipedia contributors. “Quicksort.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Quicksort
[30] GeeksforGeeks. “Hyper Quick Sort.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/hyper-quick-sort/
[31] Wikipedia contributors. “Depth-First Search.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Depth-first_search
[32] GeeksforGeeks. “Depth First Search Algorithm.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
[33] Wikipedia contributors. “Breadth-First Search.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Breadth-first_search
[34] GeeksforGeeks. “Breadth First Search Algorithm.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
[35] Wikipedia contributors. “Best-First Search.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Best-first_search
[36] GeeksforGeeks. “Best First Search.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/best-first-search/

[37] Wikipedia contributors. “Binary Search Algorithm.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Binary_search_algorithm
[38] GeeksforGeeks. “Binary Search.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/binary-search/
[39] Wikipedia contributors. “Graph Coloring.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Graph_coloring
[40] GeeksforGeeks. “Graph Coloring Problem.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/graph-coloring-applications/
[41] Wikipedia contributors. “Minimum Spanning Tree.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Minimum_spanning_tree
[42] GeeksforGeeks. “Minimum Spanning Tree.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/
[43] Wikipedia contributors. “Prim’s Algorithm.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Prim%27s_algorithm
[44] GeeksforGeeks. “Prim’s Algorithm.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/
[45] Wikipedia contributors. “Kruskal’s Algorithm.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Kruskal%27s_algorithm
[46] GeeksforGeeks. “Kruskal’s Algorithm.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/
[47] Wikipedia contributors. “Shortest Path Problem.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Shortest_path_problem
[48] GeeksforGeeks. “Shortest Path Algorithms.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/shortest-path-in-a-directed-acyclic-graph/
[49] Wikipedia contributors. “Bellman–Ford Algorithm.” Wikipedia. Accessed June 24, 2025. https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm
[50] GeeksforGeeks. “Bellman–Ford Algorithm.” GeeksforGeeks. Accessed June 24, 2025. https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/

Honor Pledge

"I affirm that I have not given or received any unauthorized help on this activity, and this work is my own."

– Jaspher F. Cadelina