j MBA

RÉPUBLIQUE DU CAMEROUN
Paix – Travail – Patrie
-.-.-.-
UNIVERSITÉ DE YAOUNDÉ I
Faculté des Sciences
Département d'Informatique

REPUBLIC OF CAMEROON
Peace – Work – Fatherland
-.-.-.-
UNIVERSITY OF YAOUNDÉ I
Faculty of Science
Department of Computer Science

INFO: 5059 Search Base Software Engineering

(Dr KIMBI Xaveria)

TD 00: Advanced Algorithms and Data Structures

Exercise 1: Binary Search

Scenario: You are working on an application where you need to quickly search for a specific number within a sorted list of integers. The search should be as fast as possible.

Problem: Given a sorted list of integers, implement a Binary Search algorithm to find a target value.

Instructions:

1. Implement the Binary Search algorithm that takes a sorted list and a target value.
2. The algorithm should return the index of the target if found; otherwise, it should return `-1`.
3. Test your algorithm with different sorted lists and search values.
4. Analyze the time complexity of the binary search and compare it with a linear search.

Key Concepts:

- Searching
- Divide and conquer
- Time complexity (logarithmic time, O(log n))

Exercise 2: Graph Traversal (BFS and DFS)

Scenario: You are designing a navigation system for a small city map, where the goal is to find the shortest path or check the connectivity between locations.

Problem: Implement Breadth-First Search (BFS) and Depth-First Search (DFS) to explore a graph that represents the city's map. Nodes represent locations, and edges represent paths between them.

Instructions:

1. Implement BFS and DFS for traversing an undirected graph.
2. For BFS, implement the algorithm using a queue and explore nodes level by level.
3. For DFS, implement the algorithm using recursion or a stack and explore nodes as deep as possible before backtracking.
4. Test the traversal algorithms with a graph that has multiple connected components.
5. Implement a function that checks if two locations are connected and finds the shortest path using BFS.

Key Concepts:

- Graphs
- Traversal algorithms
- Recursion
- Time complexity (O(V + E) for BFS/DFS

Exercise 3: Dynamic Programming (Knapsack Problem)

Scenario: You are working with a packaging company that needs to optimize how products are packed into containers. The goal is to maximize the total value of packed products while staying within a weight limit.

Problem: Given a set of items, each with a weight and a value, implement the 0/1 Knapsack problem using dynamic programming to determine the maximum value that can be achieved within a weight limit.

Instructions:

1. Represent the items as a list of tuples: `(value, weight)`.
2. Implement the 0/1 Knapsack algorithm using dynamic programming:
    - Use a 2D array `dp[i][w]` where `i` represents the number of items considered and `w` represents the current weight.
    - `dp[i][w]` stores the maximum value achievable with the first `i` items and weight limit `w`.
3. Return the maximum value and the set of items included in the optimal solution.
4. Test your solution with different item sets and weight limits.

Key Concepts:

- Dynamic programming
- Optimization
- Time complexity (O(n * W) where n is the number of items and W is the maximum weight)

Exercise 4: Merge Intervals

Scenario: You are working on a calendar application that needs to merge overlapping time intervals for scheduling purposes.

Problem: Given a collection of intervals, merge all overlapping intervals and return the merged intervals.

Instructions:

1. Represent the intervals as a list of tuples, where each tuple is `(start_time, end_time)`.
2. Implement an algorithm to merge overlapping intervals:
    - Sort the intervals by start time.

    - Iterate through the sorted intervals and merge them when necessary.
3. Return the merged list of intervals.
4. Test your algorithm with a variety of intervals, including intervals with no overlaps, intervals that fully overlap, and intervals with partial overlaps.

Key Concepts:

- Sorting
- Interval merging
- Time complexity (O(n log n) for sorting)

Exercise 5: Maximum Subarray Sum (Kadane's Algorithm)

Scenario: You are developing an algorithm to find the most profitable subarray of a given array of stock prices.

Problem: Given an array of integers, implement Kadane's algorithm to find the contiguous subarray with the maximum sum.

Instructions:

1. Implement Kadane's algorithm to find the maximum sum of any contiguous subarray.
2. Track the current subarray sum and reset it if it becomes negative.
3. Test the algorithm with different input arrays, including cases where all numbers are negative, positive, or mixed.
4. Analyze the time complexity of Kadane's algorithm and compare it with a brute-force approach.

Key Concepts:

- Dynamic programming
- Subarrays
- Time complexity (O(n)

Report Structures(Max 02 pages)

Title: Algorithms for Problem Solving

Problem Representation:

- Describe the problem in detail with a simple diagram or representation.
  (e.g., for sorting, represent an unsorted array and the desired sorted output).

Solution:

- Describe the algorithm you used (e.g., Merge Sort) and provide the code snippet.

Results:

- Show sample input/output for the algorithm (e.g., input: [5, 2, 9], output: [2, 5, 9]).

Conclusion:

- Summarize the effectiveness of the algorithm and its results.

GitHub Repository:

- Upload your code and Your Report to a GitHub repository for sharing and version control.