<u>DESIGN AND ANALYSIS OF ALGORITHMS</u>

BACHELOR OF COMPUTER APPLICATIONS

to

K.R Mangalam University

by

ADITYA RAJ SINHA (2301201189)

<u>Lab Assignment 2</u>

<u>Algoritmic Strategies using Real World Problems</u>

<u>Faculty</u>: Dr. Aarti Sangwan

Department of Computer Science and Engineering

School of Engineering and Technology

K.R Mangalam University, Gurugram- 122001, India

April 2025

Contents

Chapter 1

Introduction

Algorithms play a fundamental role in solving complex real-world problems. Each algorithmic strategy has unique strengths depending on the problem's constraints, objective, and computational limitations. This project applies four major algorithm paradigms:

- Greedy Strategy
- Dynamic Programming
- Backtracking
- Brute-Force

Each approach is used in a real-world-inspired problem scenario, implemented in Python, profiled using time and memory measurement tools, and visualized using plots.

This report explains the algorithms, implementations, complexities, and performance plots.

Chapter 2

Problem 1: Scheduling TV Commercials Using Greedy Strategy

2.1 Real-World Context

Media companies sell commercial slots during TV shows. Each ad has revenue and a deadline by which it must be aired. Only one ad can be shown per time slot. The goal is to select the most profitable sequence of commercials.

2.2 Algorithmic Strategy: Greedy (Job Sequencing)

Greedy Algorithm sorts jobs by decreasing profit and picks the most profitable one first, placing it in the latest available slot before its deadline.

2.3 Input

Each commercial is of the form:

$$(id, deadline, profit)$$

Example:

$$(A, 2, 100), (B, 1, 19), (C, 2, 27), (D, 1, 25)$$

2.4 Algorithm Steps

1. Sort commercials in descending order of profit.
2. Create time slots from 1 to max deadline.
3. For each job, assign it to the nearest empty slot before its deadline.
4. Sum the profit of selected jobs.

2.5 Time and Space Complexity

$$O(n \log n) \text{ for sorting} + O(n) \text{ slot allocation}$$
$$\Rightarrow O(n \log n)$$

Space complexity:

$$O(n)$$

2.6 Visualization

Ads vs Revenue

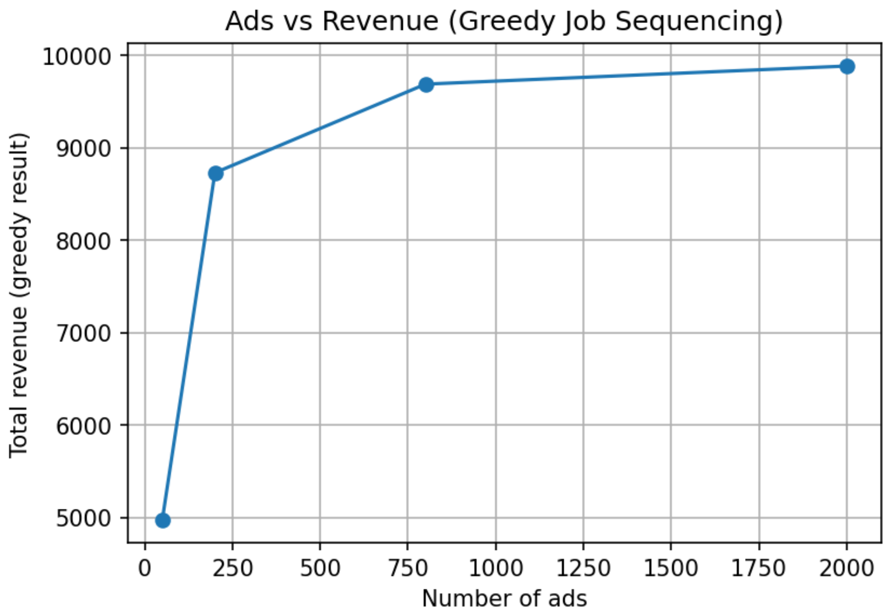

Figure 2.1: Number of Ads vs Total Revenue (Greedy)

Time and Memory Usage

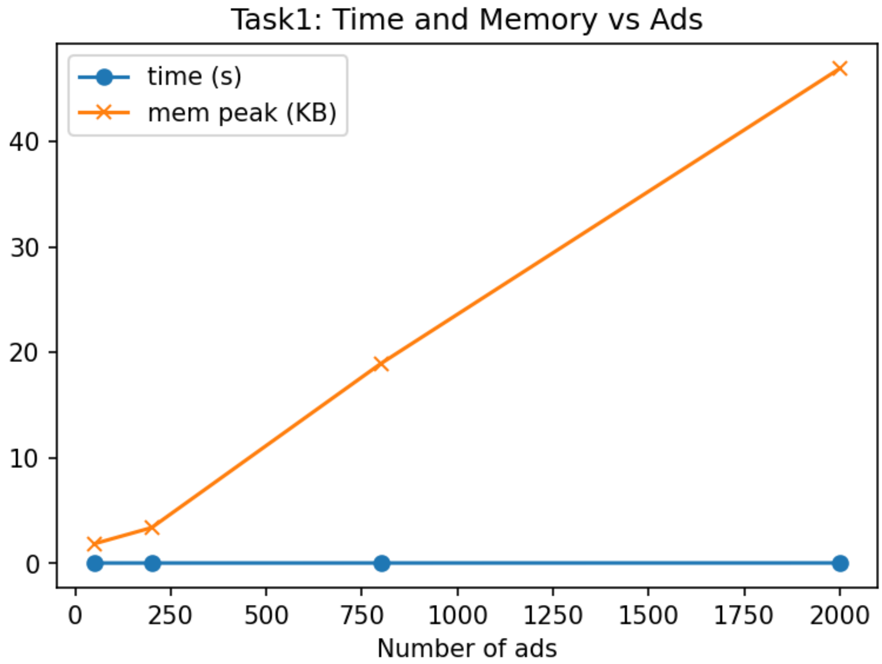

Figure 2.2: Time and Memory Usage for Job Sequencing

Chapter 3

Problem 2: Maximizing Profit with Limited Budget Using Dynamic Programming

3.1 Real-World Context

Investment decisions often require choosing a subset of profitable projects under a fixed budget constraint. Dynamic Programming is suited for problems involving trade-offs between cost and value.

3.2 Algorithmic Strategy: 0/1 Knapsack

Given:

$$\text{weights (costs)}, \quad \text{values (profits)}, \quad \text{capacity (budget)}$$

DP table:

$$dp[i][w] = \text{maximum profit using first } i \text{ items and budget } w$$

3.3 Time and Space Complexity

$$O(n \times W)$$

Where:

$$n = \text{number of items}, \; W = \text{budget}$$

Space complexity:

$$O(nW)$$

3.4 Visualization

Execution Time vs Number of Items

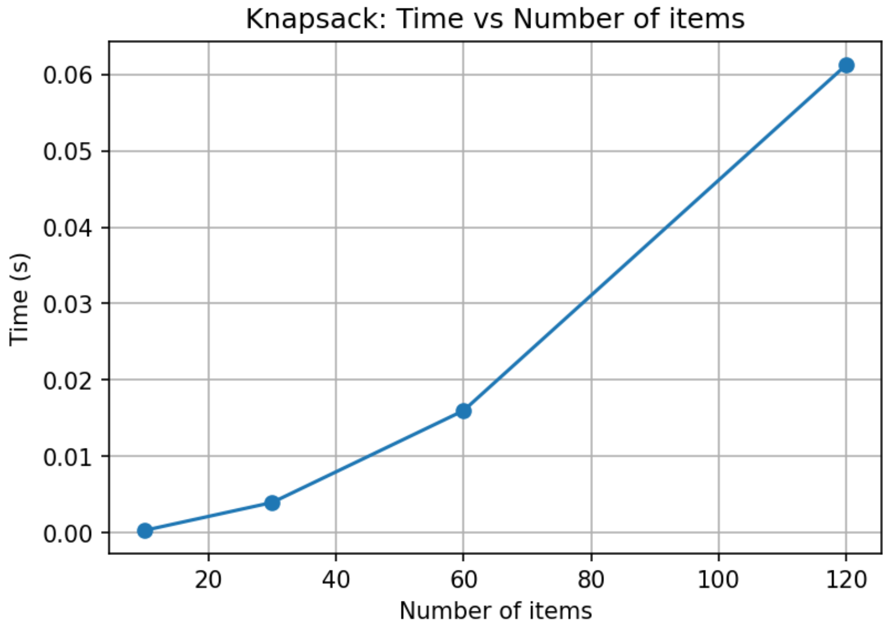

Figure 3.1: Knapsack Time Complexity Growth

Memory Usage

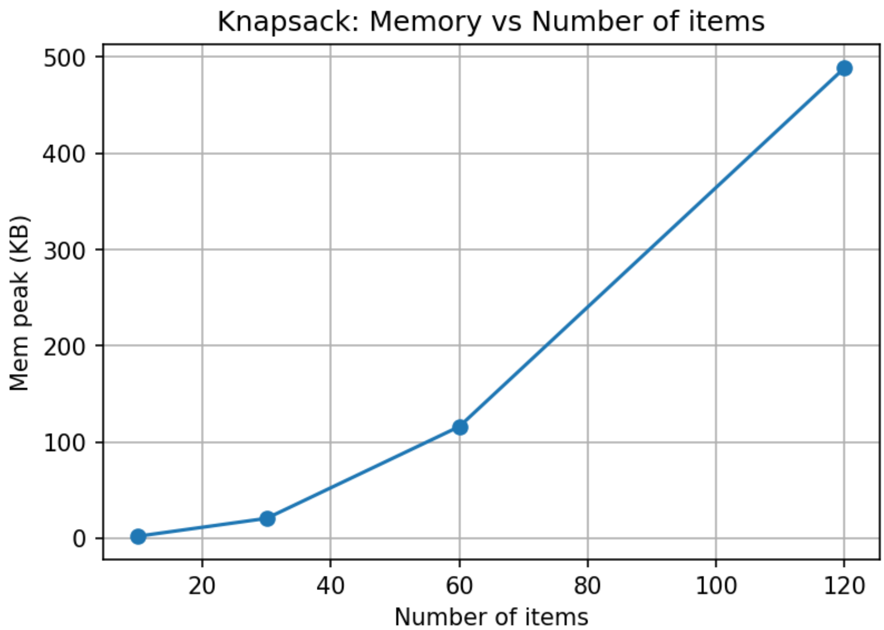

Figure 3.2: Knapsack Memory Consumption

Chapter 4

Problem 3: Solving Sudoku Using Backtracking

4.1 Real-World Context

Sudoku puzzles require placing digits from 1 to 9 such that each row, column, and 3×3 box contains all digits exactly once. This is a classic Constraint Satisfaction Problem (CSP).

4.2 Algorithmic Strategy: Backtracking

Backtracking tries a value, explores further, and undoes if invalid.

4.3 Characteristics

- Exponential worst-case time
- Prunes paths early using constraint checks

4.4 Time Complexity

Worst-case:

$$O(9^n)$$

Where n is number of empty cells.

4.5 Visualization: Time vs Number of Blanks

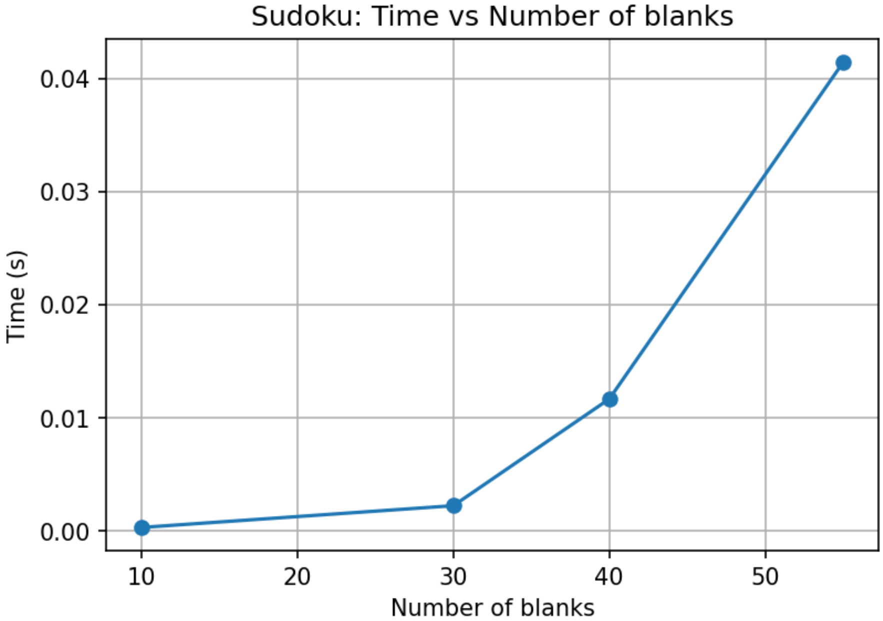

Figure 4.1: Backtracking Time vs Number of Blanks

Chapter 5

Problem 4: Password Cracking Using Brute-Force

5.1 Real-World Context

Brute-force password cracking tries all character combinations until the password is found.

5.2 Algorithmic Strategy: Brute-Force Enumeration

Using:

$$\text{itertools.product(charset, repeat} = \text{length)}$$

5.3 Time Complexity

$$O(|charset|^L)$$

For length L and charset size C, combinations explode exponentially.

5.4 Visualization: Time vs Password Length

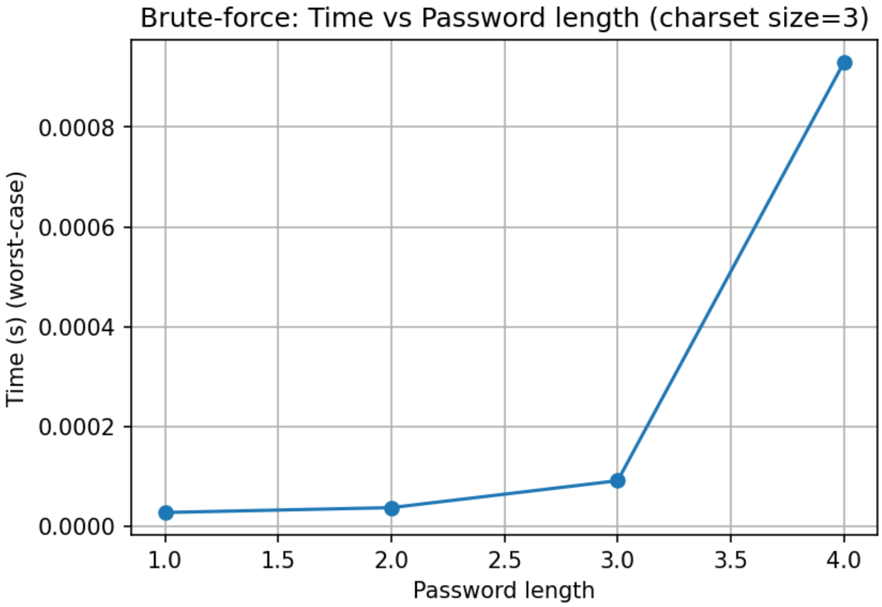

Figure 5.1: Password Length vs Time to Crack

Chapter 6

Comparative Summary

Problem	Strategy	Time Complexity	Domain
TV Commercial Scheduling	Greedy	$O(n \log n)$	Media & Advertisemen
Knapsack Profit Maximization	Dynamic Programming	$O(nW)$	Budget Planning
Sudoku Puzzle Solving	Backtracking	Exponential	Gaming
Password Cracking	Brute-Force	Exponential	Cybersecurity

Table 6.1: Comparison of All Algorithmic Strategies

Insights

- Greedy was the fastest, with linear memory and simple visuals.
- DP was significantly slower for large budgets due to 2D table.
- Backtracking showed steep time increases with harder puzzles.
- Brute-force grows exponentially and quickly becomes infeasible.

Chapter 7

Conclusion

This project demonstrates how algorithmic theory bridges into real-world systems. Each problem aligns naturally with a different algorithmic strategy:

- Greedy works well for optimization with local decisions.
- Dynamic Programming handles structured subproblems with reuse.
- Backtracking excels at constraint-driven search.
- Brute-force enumerates all possibilities when no shortcuts exist.

Practical profiling results showed how theoretical complexity reflects real execution time and memory usage.