

# Week 2 Report: Algorithmic Complexity and Dynamic Programming

## Introduction

This report summarizes the work completed in Week 2 of Algorithm Design (DTE-3611). The main focus was on exploring computational complexity—particularly NP-hard and NP-complete problems—and applying dynamic programming (DP) as a practical technique to address such challenges. Implementations of the Subset Sum, 0/1 Knapsack, and Traveling Salesman Problem (TSP) were developed in C++, and their performance was benchmarked to compare theoretical complexity with real execution results.

## Algorithms Implemented

Three main algorithms were implemented and tested: Subset Sum, 0/1 Knapsack, and TSP using the Held–Karp method. Each demonstrates the power of dynamic programming in reducing exponential complexity to manageable computational time for small to medium input sizes.

### Subset Sum Problem

**Problem:** Given a set of integers and a target sum, determine whether any subset of the numbers sums exactly to that target.

**Approach:** A bottom-up DP table  $dp[i][j]$  was constructed representing whether a subset of the first  $i$  elements can form a sum of  $j$ . Tested using  $\{3, 34, 4, 12, 5, 2\}$  with target = 9.

### Knapsack Problem

**Problem:** Given a set of items with weights and values, determine the maximum value achievable within a knapsack of capacity  $W$ .

**Approach:** DP table storing the optimal value for each sub-capacity and sub-item combination.

**Benchmark:** Tested with  $values = \{60, 100, 120\}$ ,  $weights = \{10, 20, 30\}$ , and  $W = 50$ .

### Traveling Salesman Problem (TSP)

**Problem:** Find the shortest route visiting all cities exactly once and returning to the start.

**Approach:** Implemented the Held–Karp dynamic programming algorithm using bitmasking. Tested on a 5-city distance matrix.

## NP-Hardness and NP-Completeness

Subset Sum is NP-complete (via reduction from 3-SAT). Knapsack is NP-hard in optimization form and NP-complete in decision form. TSP is NP-hard, with its decision variant NP-complete. All algorithms exemplify classical NP-hard problem-solving using DP.

Algorithm	Input Description	Approach	Time Complexity	Observed Time ( $\mu$ s)	Output
Subset Sum	Nums = {3, 34, 4, 12, 5, 2}, target = 9	DP	$O(n \times S)$	13 $\mu$ s	YES
0/1 Knapsack	Values = {60, 100, 120}, weights = {10, 20, 30}, capacity = 50	DP	$O(n \times W)$	13 $\mu$ s	Max = 220
TSP (Held–Karp)	5-city matrix	DP (bitmasking)	$O(n^2 \times 2^n)$	23 $\mu$ s	Cost = 95

## Conclusion

Dynamic programming proved highly effective for Subset Sum and Knapsack, drastically reducing runtime compared to brute-force. The Held–Karp TSP algorithm, while optimal, revealed DP’s exponential scaling limitations. Overall, the work bridged theoretical and practical understanding of algorithmic complexity and NP-hardness.

## Repository

All source code and benchmark scripts are available at:

<https://github.com/Mahid-D/DTE-3611-Week2.git>