Grant McCord

CS300 : Analysis and Design

October 27, 2024

## Contents

Introduction ........ 1
Evaluation ........ 1
Analysis ........ 1

## Introduction

This section of Project One for ABC University (ABCU) contains an analysis of the run-time and memory for vectors, hash tables, and binary search trees. The project focuses on creating pseudo code and Big O analysis for a course management program that utilizes these data structures.

## Evaluation

The following is an analysis of the runtime and memory efficiency of data structures utilized for managing course data. This evaluation specifically focuses on the process of opening a file, reading its contents, parsing each line, and creating course objects while checking for formatting errors. The worst-case running time for these operations, are expressed in Big O notation.

**Summary**

| | Open | Read | Parse | Validate | Insert | Print (sorted) |
|---|---|---|---|---|---|---|
| ***Vector*** | *O(1)* | *O(n)* | *O(n)* | *O(n)* | *O(n)* | *O(n)* |
| ***Hash Table*** | *O(1)* | *O(n)* | *O(n)* | *O(n)* | *O(n)* | *O(n log n)* |
| ***Binary Tree*** | *O(1)* | *O(n)* | *O(n)* | *O(n)* | *O(log n)* | *O(n)* |

## Analysis

The analysis of the most suitable data structure for storing course information depends significantly on the nature of the data being handled. In this case, we are dealing with a relatively small dataset, likely comprising fewer than 5,000 records. Given this context, the choice of data structure must balance efficiency, ease of implementation, and the specific operations required by the application.

For this dataset, vectors can provide a straightforward implementation. While vectors do allow for direct access to elements via indexing, finding a specific course requires iterating through the entire vector, resulting in *O(n)* time complexity for search operations. Inserting new records in a sorted order can also incur overhead, as elements may need to be shifted, leading to O(n) complexity.

Hash tables provide an average-case time complexity of *O(1)* for both insertions and lookups, making them highly efficient for handling unique identifiers like course numbers. However, their performance hinges on the effectiveness of the hash function, which must minimize collisions, and they do not inherently maintain any order.

Binary search trees (BSTs) maintain an ordered structure for efficient searching, insertion, and deletion operations. However, the balance of a BST depends on how the data is inserted.

So which one do I pick?

For this project, I recommend using a vector for data storage. The number of classes will be relatively small (around 5,000), and with today's technology, sorting and indexing such a dataset will typically take less than a second. The primary reason for this choice is ease of implementation and maintenance. While there are technically more efficient data structures, such as binary trees or hash tables, the practical implications of maintaining a vector in this case outweigh the theoretical benefits.