Lab 1: Arrays & Recursion

COSC 2436
Spring 2025
Due: Sunday, February 16, 11:59 PM

Introduction

In this lab you will find several exercises to strengthen your understanding of recursion and C++ arrays. Turn in all your work (code, spreadsheets, graphs, and short answers) into Blackboard. You may consult your fellow students and/or AI while working on the lab but ensure that your answers are your own and that you agree with the group and/or AI.

Part 1: Recursion

1) Write a recursive function named ***factorial*** that computes the factorial for a given int.
2) Write a recursive function named ***fibonacci*** that computes the value of the nth Fibonacci sequence: [1, 2, 3, 5, …]
3) Write a function named ***towers*** that counts the number of moves in Towers of Hanoi given the number of rings n. (Note, that this is different from the version implemented in class which printed the moves to the console. This time we only want the number of moves.)

Part 2: Unit Testing

Write test functions for the three functions in part 1. Name your test functions with a "test" prefix; for example, ***testFactorial***. Use the minimal test equivalence as discussed in class to ensure a minimal yet complete test. Use c-style asserts as demonstrated in class to verify and validate values or print to the console for manual verification as discussed in the Carrano text.

Part 3: Time Complexity

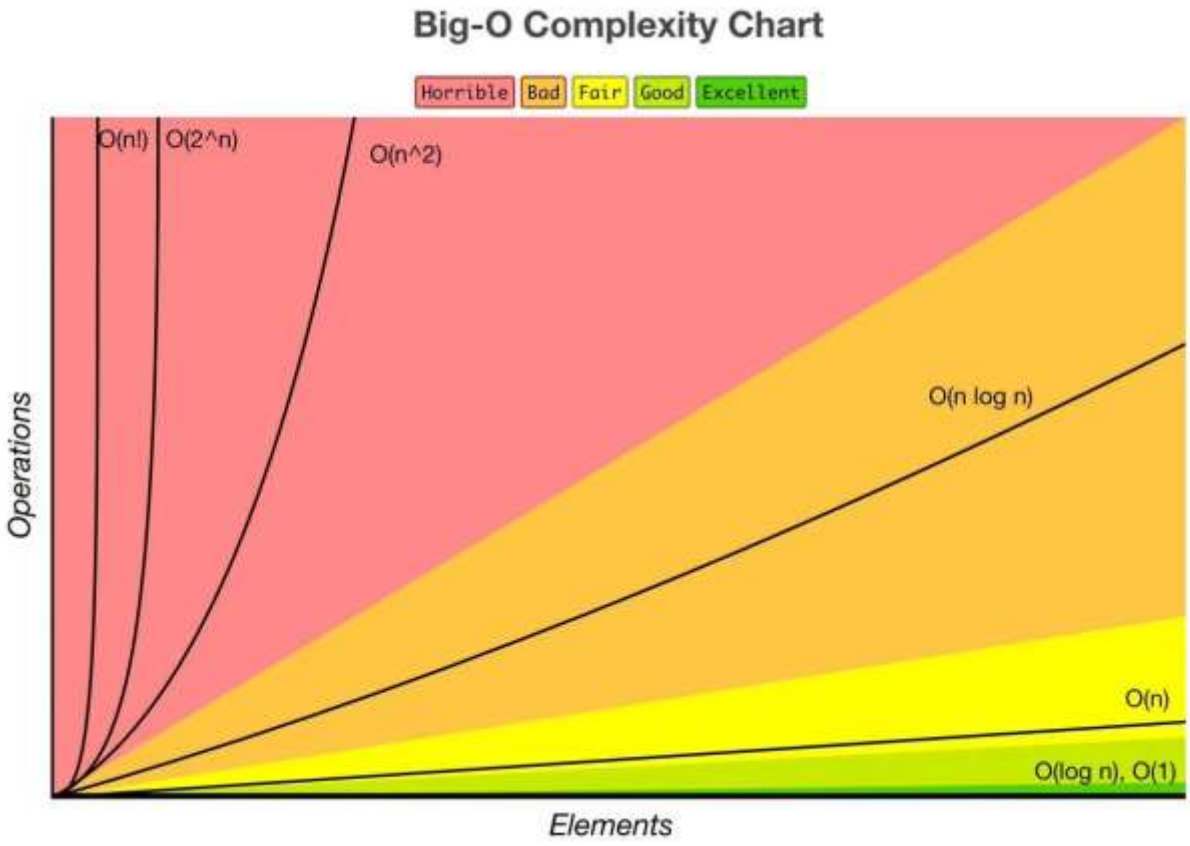
Review Big-O notation in the link provided.[1] In this exercise we'll explore the time complexity of the recursive Towers of Hanoi solution. Below you'll see a Big-O complexity chart we'll use later containing common ways algorithms can scale with user input.

The towers function you created outputs the number of moves required for the number of rings given. Use this function to gather several input and output sample points. Enter these points into a spreadsheet program such as Google Sheets or Microsoft Excel. Once you have a few sample points, create a plot like in the Big-O complexity chart. Use as many samples as necessary to make a good plot comparison. Take a screenshot of your plot to turn in.

[1] https://medium.com/@lchan217/big-o-notation-6c309d8a0fe7

Compare the chart you created to the Big-O complexity chart and answer the following questions:

1) What Big-O do you estimate the time complexity to be? O(n), O(n^2), etc. …
2) Using your chosen function from step 1, how many significant operations would 10 rings would require? 100? 1,000? 10k?
3) Is this a practical algorithm for large input sizes? Explain your reasoning.



Part 4: Array Partitioning & Test-Driven Development

As part of a larger effort implementing a quick sort algorithm, we would like to come up with a recursive solution for the k^{th} smallest value of an unordered array without sorting the array first.

Finding the kth smallest value in a sorted array is trivial; the answer is the index k of the sorted array. We can use this information to create a test to help us while we develop our solution. This is how Test-Driven development (TDD) works; we start with a test that fails and then make the test pass. TDD is especially useful for implementing data structures. It's more difficult to do with GUI applications where the user input is difficult to reproduce for the unit test.

Use the code provided which includes a completed random test and a partial recursive implementation for finding the k^{th} smallest value. Running this code will start the randomized test which is currently failing. Complete the implementation at the TODO's and ensure the test passes.