

CS375 Project 2

Due Mar 21th 10PM ET

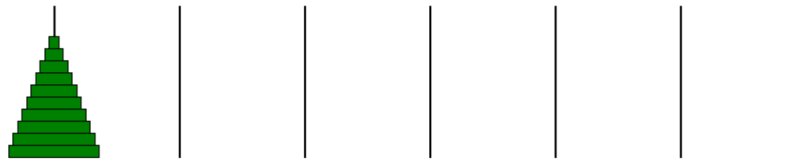
A general note for CS375: When writing things up, please write neatly and explain your answers clearly, giving all details needed to make your answers easy to understand. Graders may not award full credit to incomplete or illegible solutions.

As a reminder, projects can be submitted in small groups, optimally with 3 people. If you feel exceptionally strongly about having more than 4 people come see me.

Clear communication **is the point**, on every assignment. In general in CS375, unless explicitly specified otherwise, answers should be accompanied by explanations. Answers without explanations may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

The following project is heavily based on a series of questions from Erickson's text, specifically from the first chapter on recursion. **You will almost certainly heavily benefit from reading the first couple sections of that chapter before doing this project.**

In this project, you will be working on a variation of the game Tower of Hanoi. In this game, the board consists of k distinct pegs which will collectively hold n tiles of distinct sizes. The initial configuration of the board will look like the following:



Namely, the tiles all start on the left-most peg in sorted order (with the largest tile on the bottom). The objective of the game is to move all the tiles onto the furthest right peg. However, there are a few rules to adhere to:

- Only one tile can be moved at a time
- Tiles can only be moved to adjacent pegs
- A tile can never be placed on top of a smaller tile

The main goal of this project is to exercise your recursive problem solving techniques and to analyze the runtime of some recursive functions. This project will include both a report and video demo.

In order to help you with testing and playing around with the game, I'm including [a Python implementation](#) of the game. You'll also need this [graphics package](#) I wrote for CS151 a few years ago¹. The main class is constructed as follows: `Tower_Of_Hanoi_Game(n: int, k: int, speed:`

¹I was fed up with Zelle's graphics package not doing what I wanted.

`int = 1`): A class for the game. Here, `n` denotes the number of tiles, `k` the number of pegs, and `speed` controls the speed of the simulation - I'd recommend values less than 8 for `speed`. The class supports the following methods:

- `num_pegs()`: Returns the number of pegs in the game.
- `num_tiles()`: Returns the number of tiles (disks) in the game.
- `peg_length(peg_index: int)`: Returns the number of tiles the given peg.
- `peg_holding_tile(tile: int)`: Returns the peg on which the given tile currently rests
- `tiles_on_peg(peg: int)`: Returns the list of tiles (bottom up) currently resting on the given peg
- `move_left(peg_index: int)`: Moves the top tile on the given peg to the left
- `move_right(peg_index: int)`: Moves the top tile on the given peg to the right

The following are your explicit goals, all of which you must fully detail in your report. **For full credit, each algorithm you define below must follow recursive design.**

1. In the TowerOfHanoi Python file, I've actually given you an algorithm (`solve`) for the case $k = 3$. Determine an exact formula for the number of moves made by this algorithm on inputs of n tiles.
2. Determine an upperbound for the number of moves made by `solve` on inputs of n tiles on k pegs (this is different from part a because now the number of pegs is variable).
3. Although `solve` works, it's not very good for larger values of k . Describe and analyze a recursive algorithm for arbitrary n and k that has a better guarantee on the number of moves made than `solve`.
4. Pick an interesting function $f(n)$ and make a specialized algorithm for when $k = f(n)$ that *does asymptotically better than the algorithm you design in part 3 above*. Some interesting cases could be $k = \sqrt{n}$, $k = n + 1$, etc. For some undetermined amount of extra credit, do a second function.

Additionally, you are required to submit a video demo between 5 - 7 minutes in length demonstrating one of your algorithms from part 4. The demo should include a problem statement and a reasonable description of your algorithm, each of which should be understandable by any student in CS 231. In your demo, include an explanation of why it works and its runtime. For the runtime, I'm expecting rigorous analysis by a recursion tree and / or application of the master theorem (**if at applies**); these last two parts should be targeted for fellow CS 375 students.