HW 2: Analysis of Algorithms II (CSCI 511) Spring 2025

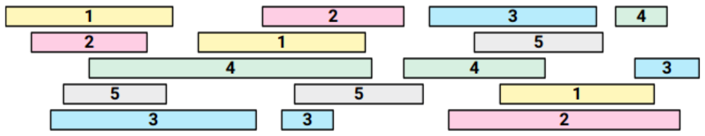
Goal: Understand greedy algorithm design and optimality proofs

Remember, you may work with your classmates but you must write up your own solutions and not copy each other. Show your work! **State your group members or that you worked alone.**

Show clear, concise solutions that are a combination of pseudocode and prose. Justify the correctness and analyze the running time of your algorithms. **Read the problem set rubric.**

Total points: 40

1. **(Interval coloring)** *[20 points]* Let X be a set of n intervals on the real line. A *proper coloring* of X assigns a color to each interval, so that any two overlapping intervals are assigned different colors. Describe and analyze an efficient algorithm to compute the minimum number of colors needed to properly color X. Assume that your input consists of two arrays `L[1:n]` and `R[1:n]`, representing the left and right endpoints of the intervals in X. Use a greedy algorithm, and prove it's correct.



A proper coloring of a set of intervals using five colors.

2. **(Double-and-add)** *[20 points]* Consider the following process. At all times you have a single positive integer x, which is initially equal to 1. In each step, you can either increment x by 1 or double x. Your goal is to produce a target value n. For example, you can produce the integer 10 in four steps since $10 = (((1 + 1) \times 2) + 1) \times 2$.

Obviously you can produce any integer n using exactly $n - 1$ increments, but for almost all values of n this is horribly inefficient. Describe and analyze an algorithm to compute the minimum number of steps required to produce any integer n. Please also give a tight upper bound on the number of increment/double steps required.

Hint: Formulate a dynamic programming solution, then show that you can make a greedy choice. I found myself using induction for the greedy choice property.

Both problems 1 & 2 are from Jeff Erickson's textbook.

Rubric for greedy problems

- *[6 points]* Pseudocode: Present compact and readable pseudocode for your solutions.
- *[2 points]* Runtime analysis: Give a tight worst-case bound on your algorithms.
- *[12 points]* Proof of correctness: Greedy algorithms are often intuitive, but intuition can be fickle. Just because the algorithm "feels" right doesn't mean it is, so prove it to me.