CS 5720 Design and Analysis of Algorithms

Project #3

Submission requirements:

- A .zip file containing your source code. You may use any language you would like.
- A PDF (submitted separately to the Canvas assignment) containing each item below that is listed as a **Deliverable.** For each item contained in your PDF, clearly mark which deliverable it is associated with. Plots should be clearly labeled and have descriptive captions.

Dynamic Programming and Knapsack: bottom-up versus top-down. In class we discussed two versions of a DP algorithm for the Knapsack problem: **a bottom-up version** that computes the solutions to all subproblems (without repeating any solutions), and **a top-down version** that uses memoization (the book calls this the "memory functions" approach) to ensure that we solve *only* the subproblems that need to be solved. We discussed the notion that the top-down approach should be a constant factor faster than the bottom-up approach. This project explores this speedup, and looks for potential tradeoffs in this space.

1. **Implement both of these algorithms in a language of your choice** As a reference, see the book's Section 8.2. Make sure your both of your algorithms work for any problem instance. Verify that your algorithms both have worst-case complexity of $\Theta(nW)$ as analyzed in class.

   Deliverable 1: Your implementation's code and your verification of correctness.

2. **Compare the performance of your two algorithms on random inputs.** Generate inputs with random item weights and values (choose your weights to be random **integers** between 1 and capacity W). Generate plots which show run times with respect to n (for fixed W), and plots which show run times with respect to W (for fixed n). Let n and W be as large as you need them to be to see a performance difference between your two algorithms. Is the performance gap bigger when n is large or when W is large? If there is a difference, try to explain it.

   Deliverable 2: Plots showing the time performance of your algorithm as a function of n and W for each algorithm, and discussion about the reasons for any performance differences between the two algorithms.

3. **Compare the performance of your algorithms on special inputs.** Now, craft inputs where all weights are relatively low (say, select the weights to be random integers between 1 and 10). Re-run the comparison between the two algorithms. Has the performance gap changed in any interesting way?

   Deliverable 3: Plots showing the time performance of your algorithm as a function of n and W for each algorithm on special inputs, and discussion about the reasons for any performance differences between the two algorithms.

4. **Illustrate that this is a *pseudopolynomial*-time algorithm.** The time complexity $\Theta(nW)$ is pseudopolynomial: it is polynomial in the *value* of W, but not the *size of the representation of* W. Generate a plot which illustrates this characteristic.

   Deliverable 4: A plot with running time on the vertical axis and the size of the representation of W on the horizontal axis which shows that this algorithm is exponential in the size of the representation of W.