CS 3310 Design and Analysis of Algorithms
Project #2

In this class, we have discussed several solutions for the 0-1 Knapsack problem. Implement the following algorithms in either Java, Python, or C++:

a. The Brute-Force solution that implicitly builds the entire state space tree.
b. The Backtracking solution.
c. The Branch-and-Bound solution. Make sure to use the best-first search version.

Algorithms b and c have been discussed in class. The Brute-Force solution is similar to the Backtracking solution, except we never Backtrack until we reach a leaf.

When the user starts the program, they should be asked for the following input:

- How many items are there to potentially take?
- What is the weight and profit of each item?
- What is the max weight the bag can hold?

You should then find the correct solution using each of the three algorithms. However, you want to modify the algorithms to keep track of how many nodes are visited in the state space tree. When each algorithm has found a solution, print out which items are taken and which are not, what the maximum profit is, and how many nodes were checked.

Answer the following: Does the branch-and-bound solution always perform better than the backtracking solution? If not, for what input did the backtracking solution check less nodes? Did you find any inputs for which the brute-force algorithm is not much worse than either of the other two solutions? On average, how much more efficient are the two solutions discussed in class over the brute-force solution? After running your program on several different inputs, make a table displaying the # of nodes checked for each of the three solutions.

***Please zip your submission including your source code, output of executions, and a typed report*. *Name your zip file as username_p2.zip and send it to jdjohannsen@cpp.edu*.**

This project will be graded based on the correctness and efficiency of your implementation of the three algorithms (70%) and the quality of your report (30%).

Note: Rather than use parallel arrays as the book does, I found it easier to create an Item class that contained weight, profit, and ratio variables. I then stored each Item in a single ArrayList, sorted by ratio.