CS 215 - Project 2
Due Friday of Week 10

0/1 Knapsack or Thief Problem

Be sure to read the Programming Requirements and Grading document before turning in your assignment (hard copy only!) This is an individual assignment; you must work alone.

The 0/1 Knapsack problem (presented in class and in Chapter 16 pg. 425-426) is a classic problem in Computer Science that has applications not only in Operations Research, but in Electric Power Management, VLSI layout, Pattern Layout in Manufacturing, Process Scheduling and Computer Networking. The idea is to load a box of fixed capacity with containers of known weights and profits such that the profit is maximized and the box is not overloaded.

The 0/1 Knapsack problem is NP-Complete so we do not expect a low-cost solution to the problem. An exact solution is possible through Dynamic Programming and an approximate solution technique is presented in Chapter 16 by sorting the items in non-increasing profit/weight density. You must use your implementation of Heapsort, Mergesort, or Insertion Sort (choose and justify your choice as the best one) for this sorting algorithm with the appropriate assert statements embedded.

The solution to this problem is expressed as \( x_i : i = 1, 2, \ldots, n \) such that \( x_1 = 1 \) when the object is taken and \( x_i = 0 \) otherwise. We are given two sets value \((p_i : i = 1, 2, \ldots, n)\) and weight \((w_i : i = 1, 2, \ldots, n)\) along with a capacity \( c \), the goal is to maximize the value packed into the container or: \( \sum_{i=1}^{n} x_i p_i \) with the constraint, \( \sum_{i=1}^{n} x_i w_i \leq C \).

In this project you will implement four solutions to 0/1 Knapsack and compare the quality of those solutions and the run-time cost of their solutions. In short, this assignment requires you to implement:

- Brute force solution: Try all possible combinations of \( x_i \) and take the maximum value that fits within the given capacity.

- Greedy solution 1: Sort the items by value and always pick the most expensive item that fits into the knapsack. (Optional)

- Greedy solution 2: Form a price density array, sort it in non-ascending order, and use a greedy strategy to fit the greatest value into the knapsack.

- Dynamic Programming solution: Use the iterative method we used in class.

Your paper must address two major issues: How close does each strategy come to the optimal solution? How do the three strategies compare in terms of execution time? Explore the problem with these two issues in mind. I suggest you generate random \( p_i \) and \( w_i \) and use a constant capacity of \( c = 100 \) for testing, then generate the other capacities. The dynamic programming solution presented in class will only work for integer weights so keep that in mind when generating \( w_i \) and insure that \( \sum_{i=1}^{n} w_i \geq C \). You must explore at least the range of the number of items between 5 and 100 in some reasonable increments.

To be effective in this assignment you should read and understand the Sahni, Chapters 15, and 16 thoroughly. It is possible to download all this code from sites on the web, however, it’s much more difficult to debug and/or insert assert() statements into somebody else’s code, so it makes the project much harder! If you feel you must download the code from the web, its vital to provide the source.

Grading will follow the same rubric as Projects 1. If you do not attach your actual code as an Appendix, I cannot effectively grade your work.

\(^{1}\)20 points extra credit