Kamil Gumienny
Project 3 - CS2223

Executive Summary

In this project, we tackled the knapsack problem using three different algorithms.

- Brute Force

  Using a brute force algorithm, you compute all possible sets of items in a knapsack including knapsacks with no items to knapsacks with all the possible items. Then, all those sets are checked against each other to see which one holds the most value and is under the weight limit for the knapsack.

  - Time efficiency: $n2^n$
    - Every element is accessed $2^n$ times when building every possible set from the number of elements given and then every set is accessed to compute its weight and value when finding the optimal one.
  - Space efficiency: $2^n$
    - A list of n items generates $2^n$ sets when making a subset of every possible combination of items
  - Optimal: Yes, all possible solutions are compared, however it comes with a cost to time and space efficiency.

- Dynamic Programming

  Using Dynamic programming, a table is constructed that saves space in exchange for time. A computation happens one time that computes subsets that contain certain items and a weight. It is a bottom-up computation that creates the optimal items to add given x amount of weight left and is then traversed across rows and columns looking for items to add.

  - Time efficiency: O(n*W)
    - Where n is # items and W is max weight of sack. The table grows vertically as the number of items goes up and and grows horizontally as capacity is greater, which requires more computations.
  - Space efficiency: O(n*w)
    - The table grows as the variables increase which takes up more space, obviously.
  - Optimal: Yes, the table is sorted with the best solutions so the best solution is always chosen.

- Method of Choice (Ratio method):

  My method of choice was a ratio method that took the list of items and sorted the list and all of its items by the ratio of each of the item's weight to its value. Then, items from the front of the list are prioritized and added to the knapsack until the knapsack cannot hold any items or there are no more items to choose from.

  - Time efficiency: O(nlogn)
    - We only have to do a computation O(1) and sort using any of the sorting algorithms we learned which is at best O(nlogn)
  - Space efficiency: O(1)
    - You work with the list you're given.
  - Optimal: No.
    - This generates an optimal solution sometimes but not always because it's greedy and doesn't look at the most optimal solution.

The best knapsack solution with regards to time and space efficiency is the ratio method which is a greedy algorithm. This method does not produce the optimal solution every time so it's not always correct.

| Input | Output Brute | Output Dynamic | Output Ratio | comments |
|---|---|---|---|---|
| 100<br>30,42,53,15,23<br>40,25,74,4,120 | 198 | 198 | 198 | |
| 250<br>30,42,53,15,23,15,34,54,63,23<br>40,25,74,4,120,43,21,34,50,320 | 672 | 672 | 682 | Greedy algorithm fails |
| 100<br>23,15,34,54,63,23<br>10,4,21,34,50,30 | 80 | 80 | 80 | |