Executive Summary

The knapsack problem is implemented using the Brute Force, Dynamic Programming, and Greedy Algorithm for this project. I conducted my experiment by making sure my three algorithms mentioned above were running. Then I made a user interface with a menu so that they could select which algorithm they wanted to use for solving the knapsack problem. The answers to each of the test cases I used is in the table answering question number 3. I learned many lessons such as implementing the three algorithms and learning how they're similar and different from each other. The most time and space efficient is the Greedy Algorithm because it followed a logical, straightforward way of solving the problem and has the least number of steps and loops to complete the problem. The next time and space efficient algorithm is dynamic programming because it uses a table to keep track of all the optimal values and then returns the biggest value located in the last cell. It solves the sub-problems only once. The Brute Force, though, solves them repeatedly and tries all the combinations so it takes the most time. It also stores items constantly and repeatedly goes through the loop and if statements.

1. **What is the time/space efficiency of each of your algorithms?**

For Brute Force, I got O(n*2^n) for time efficiency because it goes through all the possible combinations which is 2^n and there's a for loop that runs through the elements which is n. For space efficiency, I got O(2^n) because it goes through list of all combinations that are possible in which the size depends on what the input is. For Dynamic Programming, I got O(n*Capacity) for time and space efficiency because there is a for loop that goes through the number of values inputted and a for loop that goes through the capacity. When you use the summation formula, you get O(n* Capacity). For the Greedy Algorithm, I got O(nlogn) for the time efficiency because the sorting takes O(nlogn) time and there is a for loop that goes through the values of the ratios which is O(n). When you compare these two, you get the most significant to be O(nlogn). The space efficiency is O(n) because it grows linearly based on the inputs.

Efficiencies/Algorithms	Brute Force	Dynamic Programming	Greedy Algorithm
Time Efficiency	O(n*2^n)	O(n*Capacity)	O(nlogn)
Space Efficiency	O(2^n)	O(n*Capacity)	O(n)

2. **Do all three of your solutions provide an optimal solution? Why or Why not?**

The Brute Force and Dynamic Programming algorithms provide an optimal solution. This is because with Brute Force, the optimal solution will be produced because it looks through all the possible combinations. Dynamic Programming will also give the optimal solution because it keeps track of the values in a table and will return the highest stored value in the table. However, the Greedy Algorithm does not give the most optimal output. This is because the Greedy Algorithm makes decisions based only on the information it has at the step it is on without looking at the overall problem. In other words, it makes a locally-optimal choice in the hope that the choice will lead to a globally-optimal solution, which is not always the case. In the case of the Knapsack example, the Greedy Algorithm will take the first item with the highest ratio that can fit in the knapsack without trying to see what is optimal.

3. **Create and supply two other knapsack examples to test your 3 functions.**

Knapsack Examples	Brute Force	Dynamic Programming	Greedy Algorithm

C= 6
W= [3, 2, 1, 4, 5]
V= [25, 20, 15, 40, 50]
	Answer: (65, [15, 50])
	Time: 0.000107
		Answer: (65, [15, 50])
		Time: 0.000084
			Answer: (65, [15, 50])
			Time: 0.000040

C= 100
W= [30, 42. 53, 15, 23]
V= [40, 25, 74, 4, 120]
	Answer: (198, [74,4,120])
	Time: 0.000164
		Answer: (198, [74,4,120])
		Time:0.000891
			Answer: (198, [74,4,120])
			Time: 0.000027

C= 8
W= [1, 5, 3, 4]
V= [15, 10, 9 , 5]
	Answer: (29, [15, 9, 5])
	Time: 0.000089
		Answer: (29, [15, 9, 5])
		Time: 0.000108
			Answer: (29, [15, 9, 5])
			Time:0.000032

*The answer is in the form of the most optimal solution and the values that produced the optimal solution

4. **Which is the better knapsack solution and why? Is this true for all knapsack examples?**

The better knapsack solution out of the 3 algorithms implemented is Dynamic Programming. This is because dynamic programming solves each of the smaller sub-problems only once and records the results in a table instead of repeatedly solving overlapping sub-problems. The Brute Force method will produce all the possible combinations which is not time efficient. Furthermore, the Greedy Algorithm does not return the optimal solution as explained in question number 2. This is not true for all knapsack examples because the Knapsack problem is NP-complete, meaning that no such algorithm has been found to be solved in polynomial time.