Project Readme Template

Version 1 9/11/24

A single copy of this template should be filled out and submitted with each project submission, regardless of the number of students on the team. It should have the name readme_"teamname"

Also change the title of this template to "Project x Readme Team xxx"

1	Team Name: bbaron2
2	Team members names and netids Brendan Baron bbaron2
3	Overall project attempted, with sub-projects: Bin Packing Problems - The "Knapsack" Problem An equivalent of DumbSAT for a solver that returns a coin combination that works
4	Overall success of the project: High. I felt like I learned a lot about how an algorithm takes exponentially longer as the input/number of variables increases, and I think my results clearly show that.
5	Approximately total time (in hours) to complete: 9.5 hours 1 hour to understand the different prompts, read up on them, decide which one I wanted to do, and gain enough basic understanding of it to start 3 hours building the code, building the test cases, optimizing the code, and creating the graphs for said problem 2 hours after learning that the problem is not infinite coins to change it to a finite coins problem 2 hours after learning that it is not the target that should be on the x-axis but the number of coins (it took a while to get my test cases as some would run for a while, which I later shortened to save everyone time) 1.5 hours to get everything together, answer all the questions, and submit it Note: Even though I had to redo a fair part of the code, I wouldn't say that time was wasted as it helped me better understand how to approach the problem. Some of the changes that needed to be made were also small.
6	Link to github repository: https://github.com/bbaron26/Theory
7	List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary). Add more rows as necessary.

File/folder Name	File Contents and Use

Code Files	
knapsackCoins_bbaron2.py	Code to solve and hold the results of the problem, test cases, and code to plot the problem
Test Files	
checkData_bbaron2.xlsx	Shows how to get to the target values using the given coins from the test case used
Output Files	
outputData_bbaron2.txt	Contains the number of total coins, target, result (true if target reached, false if not), time, and combination to reach target (empty array if not possible) for each test case ran Note: the code just prints this out but I copied it and put it in a .txt file to make it easy to access
Plots (as needed)	
plot_bbaron2.png	Plots total coins and time for each test case. shows the exponential nature of the problem. The test cases were capped at 14 total coins because after that it grew by so much that the rest of the points became difficult to distinguish between (they were all pretty much at 0 in comparison. this made the graph look worse)

8	Programming languages used, and associated libraries: python time, matplot.pyplot as plt
9	Key data structures (for each sub-project): List/Array: coins (their different values), countsList (number of each coin available), targetList (list of target values), executionTimes (stored the executionTimes), results (stored True if a combination was found and False if one wasn't), combinations (stored the coins used to reach the target and was left empty if no combination was possible), totalCoinsList (the total number of coins available), targetValues (stored target values corresponding to the results) Recursion: function determineCombinationLimited used recursion to explore the different possible combinations of the available coins that could be used and see if any of them reached the target value. To check if it was possible with a finite number of each coin, it reduced the count of each coin. Then, it would make recursive calls to try different combinations. If none of them reached the target, it would restore its coin count. Tupe: determineCombinationLimited returned a tuple (result, combination). Result was a boolean indicating if a valid combination can be found. Combination was a list of the coins that could be used to reach a target. The combination was empty if the result was

	false.
10	General operation of code (for each subproject) function determineCombinationLimited is a recursive algorithm that iterates through all the coins trying to find a combination that reaches the target. If it finds one, it exits, returning “True” and the combination of coins that was used. If it doesn’t find one, it exits after trying all possible combinations, returning “False.” A list of coin values, 2d array of counts available for each coin, and a target list are all initialized. The code then loops through the coin counts and target list, calling the function to see if a combination can be formed to reach each target with each different coin count. The time it takes for each combination attempt is measured. The resulting execution times are then plotted against the total number coins. It saves this plot, which is an exponential, as a .png file.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. The test cases I used involved 6-14 total coins. I used these because anything larger than 14 grew by so much that all the other dots would seem to be at 0, making the exponential a bit harder to see. I remember that 16 took almost a minute when no combination was possible for example. I made the coins all even values so that I could have a couple odd target values that wouldn’t be possible. This made it easy to make sure that there were always a couple that showed the worst case. I used multiple targets to include some lower and higher ones since that would have a bit of an effect on the time. I put the combinations and targets in an excel sheet titled “checkData_bbaron2.xlsx” to double check my results. They verified the correctness of my code by showing that the combinations that the code came up with did in fact total to the target value. It is harder to prove a negative (that it is impossible to get to certain numbers), but that is why I only used even coins and had a couple odd targets because I knew it would be impossible to reach them.
12	How you managed the code development: After coding the part that would check to see if a target was possible given a set of coins (both their values and how many of each of them you had), I first checked the code by running it on really simple numbers that I knew the answer to. After I got that to work, I added the code to keep track of the coins used to get to the target and made sure that they fit the criteria of coins available and added up to the target. After that, I added the time element and looked through the data to make sure it seemed exponential, as I was expecting it to be. Then, I plotted the data on the graphs to see if I could see the exponential curve, which I was able to do. Note: I originally coded this problem thinking that it was unlimited coins. However, I didn’t really write about that part since I don’t think it is what is being looked for in this question. However, it is important to note that since that code was similar to this code, it made it easier to write the algorithm to see if the target could be reached using the coins available. I also already had test cases made that I was able to use after slight modifications. I still did follow all the steps lined out above.
13	Detailed discussion of results: The results show a clear exponential relationship between the number of coins available and the time it takes to determine if a combination to reach the target value is possible. The exponential curve grows at a crazy rate the more coins that are added. It quickly goes from under a second to minutes. Larger targets also take longer than smaller

	targets. This can be seen by how the exponential curve for target 281 is above the exponential curve for 261. The difference between the two grows more and more. It is important to note though that this growth only happens if the maximum number made from the coins available is greater than the target (once the target goes above it, it no longer increases the time as so). Having more coins of different values also increased the time as there were more possible combinations that needed to be tried. Note: whenever I say increased the time, this focuses on the worst case (where a combination isn't possible)
14	How team was organized I was the leader and I had to obey myself
15	What you might do differently if you did the project again I for some reason was confused at first on how the Knapsack Coin Problem worked. At first, I thought that the number of each coin was unlimited. Then, I thought that the target value is what increased the time exponentially. Both of those were wrong as the number of each coin is limited and the number of total coins is what increases the time exponentially. Ideally, I wouldn't have spent so much time redoing my work due to my incorrect assumptions. However, this also did lead to a greater appreciation of the problem for me, as it allowed me to see how all of the moving pieces impacted the time. I would be interested to see how graphs would differ if I played around with those other variables more.
16	Any additional material: