# Project 01 README Team mchou3

Version 1 9/11/24

| | |
|---|---|
| 1 | Team Name: **mchou3** |
| 2 | Team members names and netids: **Matthew Chou - mchou3** |
| 3 | Overall project attempted, with sub-projects: **Bin Packing Problems - The "Knapsack" Problem** |
| 4 | Overall success of the project: **Completed - Solver Works** |
| 5 | Approximately total time (in hours) to complete: **15 hours (not including runtime of scripts)** |
| 6 | Link to github repository: **https://github.com/matthewvchou/mchou3_knapsack** |
| 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 | |
| **/code/testcase_generator_mchou3.py** | **Script to generate custom testcases for the knapsack problem.** |
| **/code/knapsack_solver_mchou3.py** | **Script to solve the testcases for the knapsack problem.** |
| **/code/graph_maker_mchou3.py** | **Script to create graphs of number of coins vs. time for the testcases for the knapsack problem.** |
| Test Files | |
| **/data/test_files/failure_testcases.csv** | **CSV file that holds testcases that will fail for the knapsack problem (i.e. target value cannot be reached)** |
| **/data/test_files/sucess_testcases.csv** | **CSV file that holds testcases that will succeed for the knapsack problem (i.e. a subset of coins will be found that add up to target value)** |
| Output Files | |
| **/data/output_files/failure_results.csv** | **CSV file that holds the results and data of the testcases that will fail the knapsack problem** |

| | |
|---|---|
| **/data/output_files/success_results.csv** | **CSV file that holds the results and data of the testcases that will succeed the knapsack problem** |
| Plots (as needed) | |
| **/data/graph_files/failure_graph.pdf** | **PDF file that holds the plotted points of the number of coins vs. time for the failure testcases** |

| | |
|---|---|
| 8 | Programming languages used, and associated libraries: **Language: Python** **Libraries: sys, csv, matplotlib.pyplot, numpy, time, random** |
| 9 | Key data structures (for each sub-project): **Generator: lists, CSV file** **Solver: lists, CSV file, for algorithm it follows a tree-like path using DFS** **Graph: lists, CSV file** |
| 10 | General operation of code (for each subproject) **testcase_generator_mchou3.py generates random test cases for the knapsack problem and writes them to a CSV file. It allows the user to specify parameters such as the number of test cases to generate for different ranges of coin sizes (small, medium, large), the maximum number of coins within each test case (jar), and whether the generated test cases should have a solution or not. The script creates test cases by randomly selecting coin values within specified ranges and appending a target value to the list, which could either be solvable (a subset of the coins adds up to the target) or unsolvable (greater than the sum of all coins). It then writes these test cases to a CSV file, with each line representing one test case.** **knapsack_solver_mchou3.py solves the knapsack problem using a backtracking and depth-first search (DFS) approach and writes the results to a CSV file. The script reads test cases from an input CSV file, where each row consists of a list of coin values and a target sum. For each test case, it attempts to find a combination of coins that sums to the target value using a recursive function. The solution, along with the execution time, is then written to an output CSV file. If no valid combination of coins is found, it records that as well. The script tracks and prints the execution time for each test case.** **graph_maker_mchou3.py generates a graph that visualizes the execution times of test cases from a CSV file for the knapsack problem. The script reads the input CSV file, where each row contains information about a test case, including the number of coins used and the time taken to solve it. It processes the results and plots the execution time against the number of coins for multiple test case categories (e.g., small, medium, and large value coins). The script uses different colors and markers for each category to make the plot visually distinct, then saves the generated graph as an image file.** |
| 11 | What test cases you used/added, why you used them, what did they tell you about the |

| | |
|---|---|
| | correctness of your code.<br><br>**For this project, I used a total of 96 failing testcases and 150 successful testcases to thoroughly test the knapsack solver. The 96 failing testcases were divided into 3 subsets, each with 32 test cases. In these cases, the number of coins ranged from 1 to 32 per test, and these failing testcases were primarily used to demonstrate the worst-case time complexity of the solver, which is O(2^n). Since no solution existed for these cases, the solver had to exhaustively search through all possible subsets, allowing me to test how it handled cases where no valid combination of coins could meet the target.**<br><br>**The 150 successful testcases were split into 3 subsets of 50 testcases each, with the number of coins ranging from 1 to 50. These testcases reflected different ranges of coin values: small (1-99), medium (100-999), and large (1000-10,000). The successful testcases were used to verify the solver's correctness in identifying valid combinations of coins that met the target sum. These testcases also allowed me to see how the execution time scaled as the number of coins increased in each subset.**<br><br>**By using both the failing and successful testcases, I was able to evaluate both the correctness and efficiency of the solver. The variation in coin values and the number of coins in each testcase helped confirm that the solver was working as expected across a wide range of inputs, while also providing insight into how the solver's performance changes depending on whether or not a solution exists (worst case scenario/bounding curve) and the complexity of the input set.** |
| 12 | How you managed the code development<br>**Developed solver first, tested with basic cases and edge cases manually. Next developed a generator for randomized testcases. Ran solver on randomized testcases and then developed a graph maker to plot results.** |
| 13 | Detailed discussion of results:<br><br>**For the failing testcases, it was really easy to see the worst-case time complexity of the solver - O(2^n), where 'n' is the number of coins in the jar. As expected, the execution time grew exponentially as the number of coins increased. Everytime a coin was added to the jar, the overall execution time doubled. One thing I noticed from the graph is that there seemed to be no relationship between the value of the coins and the execution time. Whether the coins were small (1-99), medium (100-999), or large (1000-10,000), the execution time stayed pretty consistent across all testcases. Surprisingly, the small coin set actually had the slowest execution time overall, but the difference wasn't that big, so it didn't seem like the value of the coins really made a difference in how fast the solver worked.**<br><br>**I decided to only include the graphs for the failing testcases because my graph maker wasn't able to handle the outliers in the successful testcases very well. For the success cases, the execution times still followed the expected 2^n growth pattern, but because of the outliers, the graph ended up with a really distorted scale. In some of the successful testcases, the solver finished way quicker than** |

| | |
|---|---|
| | **expected or took longer, which threw off the graph. Even though the O(2^n) pattern was still there in the data for the successful testcases, it just didn't show up clearly on the graph because the outliers messed with the scale.**<br><br>**Overall, the failing testcases really helped show how the solver behaves in the worst case, and it was interesting to see that the value of the coins didn't really impact performance, only the number of coins did.** |
| 14 | How team was organized<br>**No team, did by myself** |
| 15 | What you might do differently if you did the project again<br>**If I were to do the project again, I would probably allocate more time for running the scripts in order to see the different relationships between variables. Additionally, I would explore more testcases to collect more data in order to see the relationships clearer.** |
| 16 | Any additional material:<br>**NA** |