problem 1 part 2:

1.Why is there no data for the recursive BacktrackingSearch (BKT) beyond sum = 45?

---> There is no data for the recursive BacktrackingSearch (BKT) beyond sum = 45 because it reaches the python recursive limit. As the program crashes when the jugs exceed their maximum size because the search paths become excessively deep.

2.What could explain the significant dip in execution time at sum=70 across all algorithms?

--->The significant dip at sum = 70 occurs because that particular test case shoots up in a very few moves. While the jugs are huge, the search algorithms find them almost instantly at the start of the search tree.

Problem 1 part 3:

3.Generate two plots, one for BFS and one for DFS. Use the Sum of Capacities on the x-axis. Plot the branching factor and depth (d for BFS and D for DFS) on the left y-axis, and add execution time on a secondary y-axis (right). Figure 2 provides a template for reference.

--->The N-Jugs problem exhibits exponential complexity because its constant branching factor of 8 creates an unbounded increase of states when solution depth rises. The "dip" at Sum 70 occurs because the algorithm solved the problem at a shallow depth of d=11 which resulted in it examining only a small portion of the state space needed for deeper problems.

4.Do the plots confirm your answer to Part 2 question 2? If your answer was different, keep it — I'd love to see it. Use these plots to explain the dip in execution time at sum = 70.

--->The graphs demonstrate that the time required to complete a task depends on the number of steps needed to reach the goal distance of (d) instead of the jug sizes. At sum 70, the time drops because the answer is only 11 steps away, so the computer does much less work than on deeper problems.

problem 2 part 2:

1.What happens as you increase the number of coins?

--->As the number of coins increases, there is a significant delay or lag before the AI moves. After your turn the program will display a period of unresponsiveness which lasts several seconds before it continues to function normally again.

2.Why do you think that is?

--->I think this happens because the tree structure of the game expands at an exponential rate with each new coin addition. The computer processor requires extra time to perform calculations because the tree structure has developed into an enormous state which contains multiple operational paths.

3.Briefly describe how you would fix this problem. (No implementation is required for this question)

--->To fix this, I would use Alpha-Beta Pruning, allowing the AI to skip "prune" branches of the game tree that it already knows are worse than a move it has already found. The AI achieves faster decision-making through the elimination of nonessential paths while maintaining its ultimate plan of action.