CS577
Homework #4, Problem #4
Sreeram Danda

a) Dynamic Program

```
int[] max_profit(int[] values, int[] sizes, int capacity){
	//EMPTY TABLE AND RETURN STRING
	int[][] table = new int[ sizes.length+1 ][ capacity+1 ];
	int[] picked_items;
	//FILL THE TABLE
	for(int row = 0; row <= sizes.length; row++){
		for(int col = 0; col <= capacity.length; col++){
			//IF CAPACITY IS 0 OR NO ITEM IS PICKED = NO PROFIT
			if(row == 0 || col == 0){
				table[row][col] = 0;
			//IF CURRENT ITEM WEIGHT IS LESS THAN CAPACITY
			} else if (size[row-1] <= capacity){
				/*PICK MAX OF EITHER 1) PICKING THE ITEM + VALUE OF
				REMAINING SPACE FILLED WITH PREVIOUS ITEMS OR 2) DON'T
				PICK ITEM AND GET VALUE OF PREVIOUS PICKS*/
				table[row][col] = max(table[row-1][col],
				table[row-1][capacity - col] + value[row]);
			/*IF CURRENT ITEM WEIGHT IS GREATER THAN CAPACITY DONT PICK
			IT*/
			} else {
				table[row][col] = table[row-1][col];
		}
	}
	int col = capacity;
	//OBTAIN WHICH ITEMS WERE PICKED FROM TABLE
	for(int row = sizes.length; row >= 0; row--){
		if(table[row][col] > table[row-1][col]){
			picked_items.add(row);
			capacity -= size[col];
		}
	}
	return picked_items;
}
```

b) Runtime and Program Correctness

Runtime Analysis: The runtime of this algorithm is O(n*m) where n is the number of items the bandit can pick from and m is the m is the capacity of his car. This is because it takes n*m time to fill the table with the value of making certain choices. Since it only take O(n) time to obtain the result and O(n*m) is larger, the algorithm takes O(n*m) time.

Program Correctness: The first thing the algorithm does is make a table and fill it with values. It first fills the initial column and initial row with 0s. This is correct as the table is built with the columns representing possible capacities from 0 - capacity and the rows represent the items. Column 0 means a weight capacity of 0 so no item can be picked thus the column is filled with 0s. Similarly, row 0 represents no items when “no items” is picked, no weight is taken and thus the row is filled with 0s. Now the algorithm checks if the current item's weight is less than the capacity as this would be the only way to add the item. Then the bandit can make two decisions represented by the two lines following the condition. The bandit can not pick the current item or pick the current item. If the bandit chooses not to pick the current item, then at that column capacity we take the previous row’s value. This is correct because not picking the current item is equivalent to choosing only from the prior items. Otherwise, the bandit can pick the item, in which case the value of the item is added and the value of picking from the items prior for the new adjusted capacity can also be added. Selecting the greatest of these values allows for maximum profit. The last condition of the double loop accounts for not being able to pick the item since it is greater than the capacity that the bandit can carry. In this case, the algorithm correctly takes the previous value on that column, which contains the optimized value for picking from all prior items and not picking the current item. In order to obtain a result from the table the algorithm starts at the bottom right corner of the table. At this point, we can conditionally check to see whether or not including an item adds to the maximum profit which is located at this point in the table. If that point is distinct from the value above it, then it means the item was picked, not discarded based on the reasoning presented for filling the table. The algorithm then adds the value of the row which represents the item to the return array. Then the algorithm iterates over the rest of the rows going from bottom to top, checking with an updated column value depending on whether or not an item was picked. If an item is picked the left over space is the maximum capacity minus the size of that item. At that point the algorithm makes the same conditional check as the very first value and adds it based on the result. The algorithm then correctly returns the array which contains the row values that represent each item.