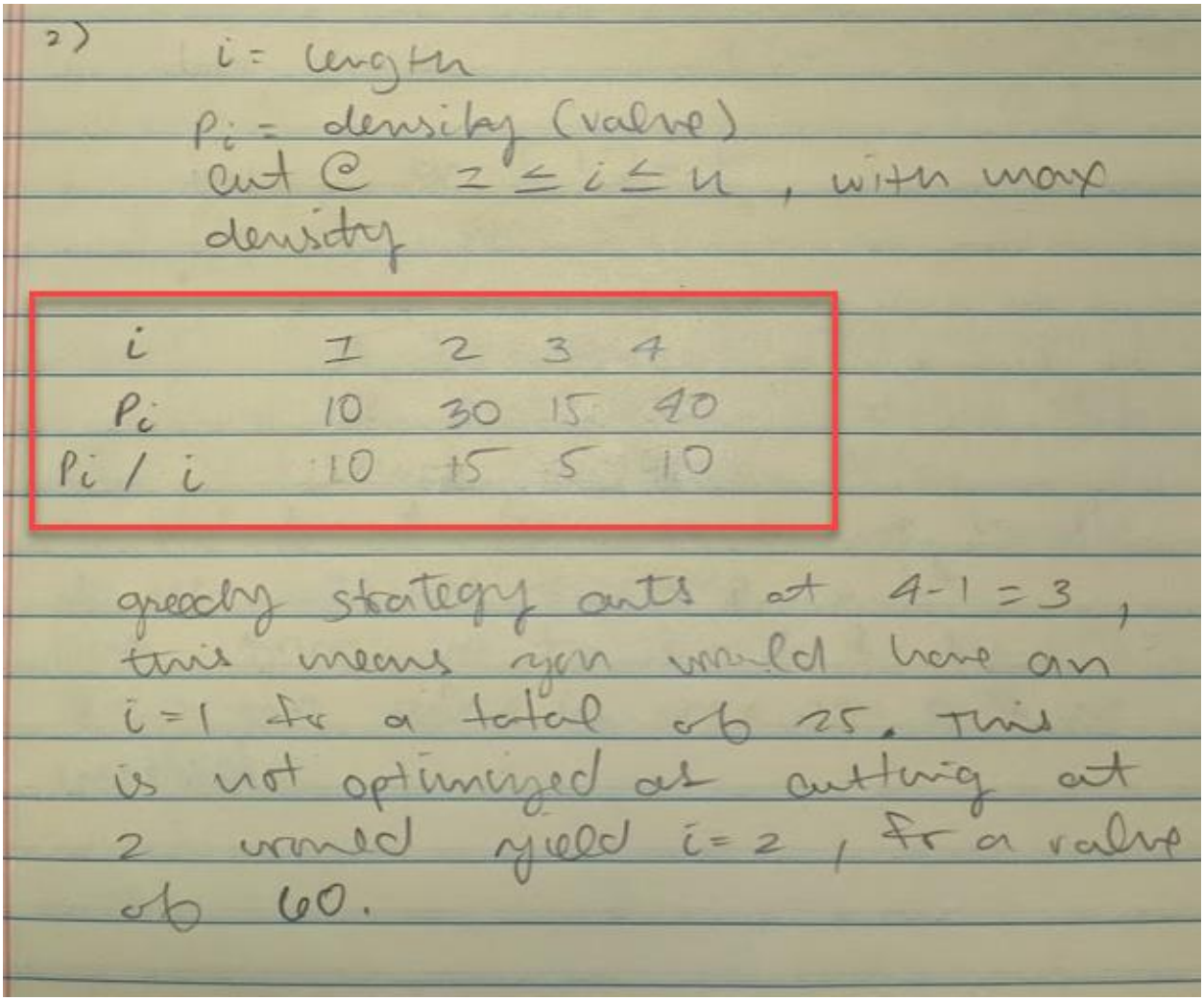
Bryan Rodriguez

CS 325 F2020

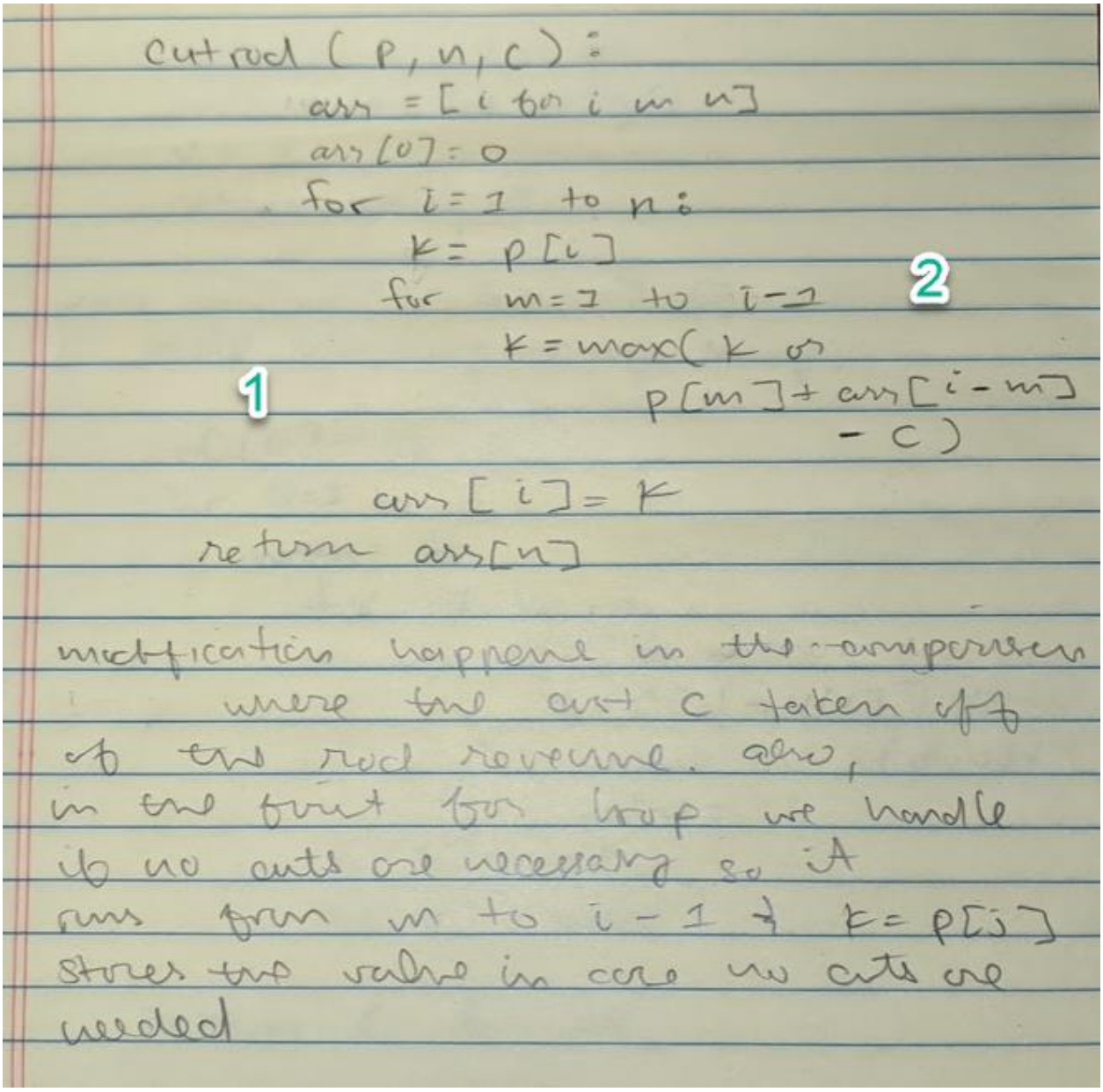
Homework 3

1)



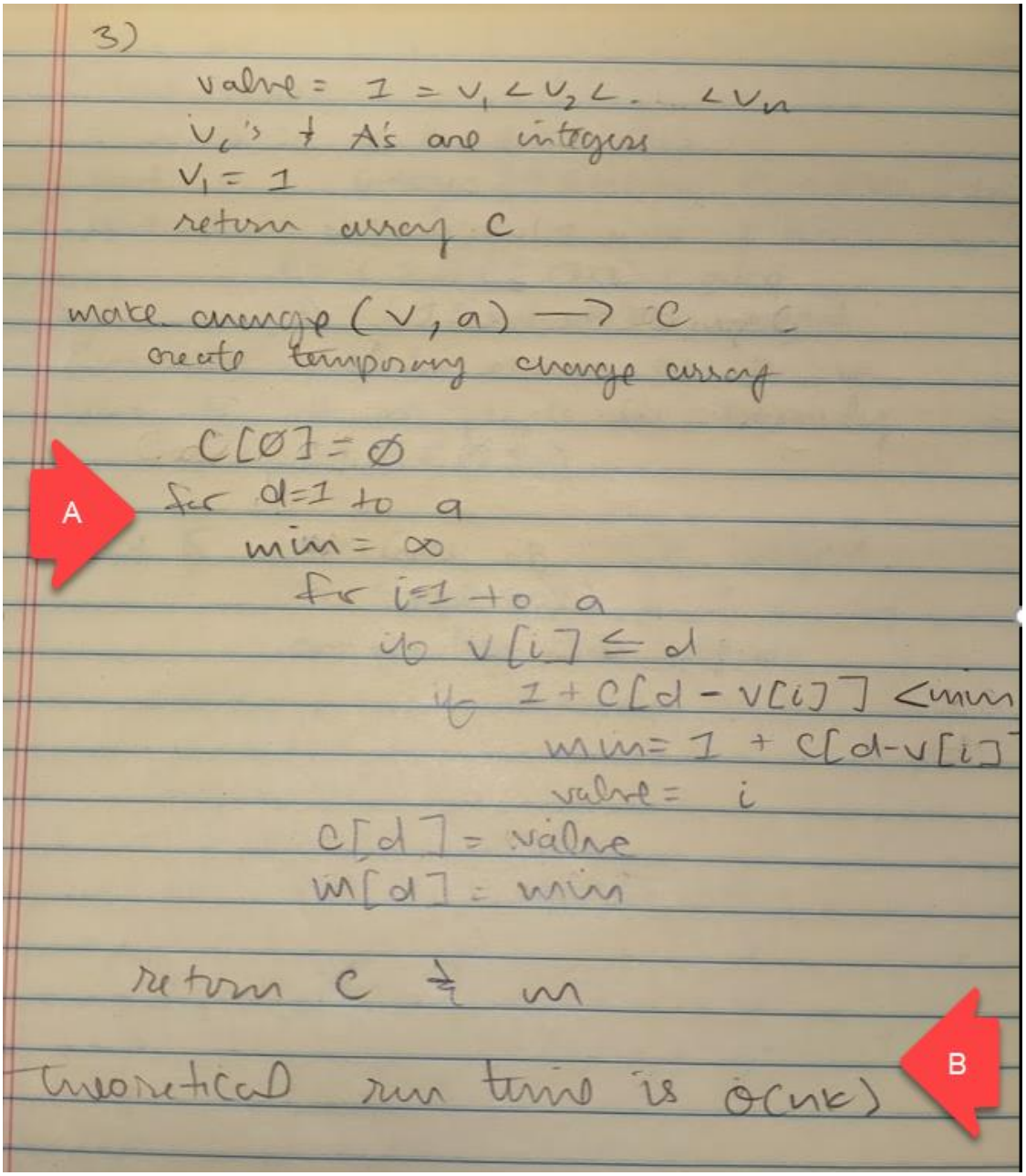
The greedy strategy here would cut at 4-1 = 3, this means you would have i = 1 left over for a total of 25. This result is not optimized as cutting at 2 would yield i = 2 left over for a total value of 60.

2)



The rod cutting algorithm here is modified at line (1) to have the cost removed from the total revenue.

3)



This is a coin change dp problem, the pseudo code is similar to the knapsack problem that is, we're also making a dp table to solve the subproblems and decide whether we're taking the coin to make change with or not. This comparison is seen in the second for loop, second if statement.

In this implementation I did not consider the backtracking or creation of the dp table. This is written near the top of the pseudocode but I didn't think it necessary for its implementation since the runtime

would remain the same. A more complete version, where a matrix is included, can be seen in the pseudocode for #4.

4)

a) This problem is a 0-1 knapsack problem; however, the algorithm needs to be applied to the entire family. Here each family member would be run through the knapsack algorithm and the results compiled back on the way out. The knapsack algorithm creates an empty two-dimensional matrix, and uses this matrix to create a dynamic programming table. Iterating through from a bottom up fashion, the algorithm compares weight values vs the amount of weigh that can be stored per column table. If the value can be stored within the temporary sack, it takes the max of the value directly above(meaning one row up, but in the same column) or the combination of the new value and the previous best solution. This solution is then added to the table. A backtracking loop will determine the order of items taken to give us the actual numbers that were taken along with the total value.

Pseudocode:

1) File handling imports from shopping.txt and creates arrays consisting of the item prices, item weights, the and the weights the family can carry. Values for the case number, number of items, and size of the family will also be needed.
2) Each family member is passed into the knapsack algorithm:
    a. Knapsack(weight, value, maxcapacity, number of items):
        i. Create a multidimensional matrix
        ii. Loop through ranges in passed weight and:
            1. If the item cannot fit into the bag, assign a 0
            2. If the item does fit in the bag:
                a. Take the maximum of the previous solved item in the same column or the value of the current item plus the value of the last solved problem with the current item's weight deducted.
3) Continue iteration above until the matrix is filled, run the matrix through a backtracking iteration to find the items and total value
4) Write these values to the text.out file.

b) This algorithm's run time would be dependent on the weight and, and values being passed so it would be O(nk)

Shopping.py ReadMe:

1. Open the shopping.py file and make sure your data.txt file is in the proper directory for it to be imported.
2. The shopping.py file will output a results.txt file.
3. The shopping.txt file will out put each case in this format:

    Test Case [n]
    Total Price:
    Member Items
    k(member): n(items taken by the member)