**Problem A.5: Writeup**

Answer the following questions in a PDF file called **ps1\_answers.pdf**.

1. What were your results from compare\_cow\_transport\_algorithms? Which algorithm runs faster? Why?

A.: Greedy algorithm runs faster, with average time complexity O(n)/linear. It is computationally more efficient considering that it doesn’t always yield the best global solution. The larger the amount of data, the more obvious the gap between the algorithms. Also, when thinking about the complexity, brute force algorithm makes use of get\_partitions overlooking all partitions of the group of cows. These need to be filtered to those that only make sense (transportable within weight limits). The complexity will be obviously higher than O(n).

1. Does the greedy algorithm return the optimal solution? Why/why not?

A.: No. It returns a practical alternative solution, a locally optimal solution but doesn’t always yield the global optimal solution.

1. Does the brute force algorithm return the optimal solution? Why/why not?

A.: Yes. It considers all possible solutions and picks the best out of them, the global optimal solution.

**Problem B.2: Writeup**

1. Explain why it would be difficult to use a brute force algorithm to solve this problem if there were 30 different egg weights. You do not need to implement a brute force algorithm in order to answer this.

A.: In the brute force algo you will evaluate all possible combinations, the increased amount of egg weights to consider means also the increase in combinations.

(Jeddd)The brute force algorithm for this problem is described as follows: First determine the upper bound for the number of eggs-if all eggs with a weight of 1 are selected, then the value of target\_weight is the maximum number of eggs needed. Find all combinations of eggs whose total number is less than or equal to this upper bound from egg\_weights, and then determine whether each combination meets the requirement that the total weight is equal to target\_weight. For example, if you have 30 eggs of different weights, you need up to 30 eggs of 1 weight. Then try all combinations of 29 eggs to make a total weight equal to 30; then try all combinations of 28 eggs to make a total weight equal to 30; then try all combinations of 27 eggs to make a total weight equal to 30 ... Finally Trying to have only 1 egg can constitute a total weight equal to 30, we have to find the least number of eggs from the above traversal. For combinatorial problems, the time complexity is high and therefore not desirable.

1. If you were to implement a greedy algorithm for finding the minimum number of eggs needed, what would the objective function be? What would the constraints be? What strategy would your greedy algorithm follow to pick which coins to take? You do not need to implement a greedy algorithm in order to answer this.

A.:

(Jeddd) If the greedy algorithm is used and the objective function is the number of eggs, we want to minimize the number of eggs. The constraint is that the sum of the weights of all selected eggs is equal to target\_weight, and the weight of eggs can only be selected from egg\_weights. The strategy of using the greedy algorithm is to first select as many eggs with the largest weight as the target\_weight as much as possible, until if you add one more, it will be exceeded, and then select as many eggs with the largest weight as the remaining weight as much as possible. Because there must be eggs with a weight of 1, this process must end, which means that the problem must be solved.

1. Will a greedy algorithm always return the optimal solution to this problem? Explain why it is optimal or give an example of when it will not return the optimal solution. Again, you do not need to implement a greedy algorithm in order to answer this.

A.:

(Jeddd)No, the greedy algorithm does not necessarily give the optimal solution. The following is a counter example: egg\_weights = (1, 9, 90, 91) and target\_weight = 99. In this example, the process of using the greedy algorithm is-first select 91, the remaining 99-91 = 8, and then only 8 Times 1, the total number is 1 + 8 = 9. However, the optimal solution is to choose a 90 and a 9 for a total number of two. This counterexample can explain that the greedy algorithm does not give the optimal solution.