The goal here is to devise an algorithm creating the optimal schedule for transportation of a group of cows from other to the home planet of some alien species. Two algorithms are considered, including a greedy algorithm and a brute force solution. These solutions are compared.

To begin with, we have to read our data (initally stored in a text file) into the program. This is accomplished by load_cows below

Problem 1

In [2]:

```
def load_cows(filename):
    """
    Read the contents of the given file.  Assumes the file contents contain
    data in the form of comma-separated cow name, weight pairs, and return a
    dictionary containing cow names as keys and corresponding weights as values.

    Parameters:
    filename - the name of the data file as a string

    Returns:
    a dictionary of cow name (string), weight (int) pairs
    """
    cow_dict = {}
    file = open(filename, 'r')
    #For each line, create a list object [a,b] where a is cow name and b is the
    for line in file :
        newline = line.split(',')
        name = newline[0]
        weight = int(newline[1])
        #assigns each name as the key to a dictionary entry whose value is the 
        cow_dict[name] = weight
    return cow_dict
```

The output of load_cows is a dictionary, whose keys are the cow names and values are the weight of the cows (in come unspecified unit).

The module operator is imported to sort by values in the input dictionary.

Problem 2:

In [3]:

```
import operator
def greedy_cow_transport(cows,limit=10):
    """
    Uses a greedy heuristic to determine an allocation of cows that attempts to
    minimize the number of spaceship trips needed to transport all the cows. The
    returned allocation of cows may or may not be optimal.
    The greedy heuristic should follow the following method:

    1. As long as the current trip can fit another cow, add the largest cow that
       to the trip
    2. Once the trip is full, begin a new trip to transport the remaining cows
```

```
    Does not mutate the given dictionary of cows.

    Parameters:
    cows - a dictionary of name (string), weight (int) pairs
    limit - weight limit of the spaceship (an int)

    Returns:
    A list of lists, with each inner list containing the names of cows
    transported on a particular trip and the overall list containing all the
    trips
    """
    new_list = sorted(cows.items(), key=operator.itemgetter(1), reverse = True)
    print(new_list)
    result = []
    totalWeight = 0

    while len(new_list) > 0 :
        trip = []
        fake_list = new_list[:]
        for pair in new_list :
            if (totalWeight + pair[1] <= limit ) :
                trip.append(pair[0])
                totalWeight+= pair[1]
                fake_list.remove(pair)
        result.append(trip)
        new_list = fake_list
        totalWeight = 0
    return result
```

This program has a few features that are worth commenting on. Implementation-wise, new_list is a sorted list of tuples (name, weight), which is generated from the inputed dictionary *cows*. The output *result* is a list of lists, each output of *result* being a list representing one trip.

Below the algorithm is tested on two inputs.

First, the dicitonary objects are generated from the input files using **`load_cow`**.

In [4]:

```
cows_1 = load_cows('ps1_cow_data.txt')
cows_2 = load_cows('ps1_cow_data_2.txt')
```

Next, we run the greedy algorithm on each input:

In [5]:

```
res_1 = greedy_cow_transport(cows_1)
print(cows_1, ': the following  trip is (greedy) optimal: ', res_1)
len(res_1)
```

```
[('Betsy', 9), ('Henrietta', 9), ('Herman', 7), ('Oreo', 6), ('Millie', 5), ('Ma
ggie', 3), ('Moo Moo', 3), ('Milkshake', 2), ('Lola', 2), ('Florence', 2)]
{'Maggie': 3, 'Herman': 7, 'Betsy': 9, 'Oreo': 6, 'Moo Moo': 3, 'Milkshake': 2,
'Millie': 5, 'Lola': 2, 'Florence': 2, 'Henrietta': 9} : the following  trip is
(greedy) optimal:  [['Betsy'], ['Henrietta'], ['Herman', 'Maggie'], ['Oreo', 'Mo
o Moo'], ['Millie', 'Milkshake', 'Lola'], ['Florence']]
```

Out[5]: 6

In [6]:

```
res_2 = greedy_cow_transport(cows_2)
print(cows_2, ': the following  trip is (greedy) optimal: ', res_2)
```

```
len(res_2)
```

```
[('Lotus', 10), ('Horns', 9), ('Dottie', 6), ('Betsy', 5), ('Milkshake', 4), ('M
iss Moo-dy', 3), ('Rose', 3), ('Miss Bella', 2)]
{'Miss Moo-dy': 3, 'Milkshake': 4, 'Lotus': 10, 'Miss Bella': 2, 'Horns': 9, 'Be
tsy': 5, 'Rose': 3, 'Dottie': 6} : the following  trip is (greedy) optimal:
[['Lotus'], ['Horns'], ['Dottie', 'Milkshake'], ['Betsy', 'Miss Moo-dy', 'Miss B
ella'], ['Rose']]
```

Out[6]: 5

Problem 3

Now we wish to greate a brute force solution to the transport problem, one where all possible sequences of trips, or results, are considered. Since the optimal result is sure to be among the set of possible results, examining each result will find us the optimal solution.

Just like the greedy program, this will take a dictionary as an input, again with a weight limit (default value of 2).

In [7]:

```
import ps1_partition
def brute_force_cow_transport(cows,limit=10):
    """
    Finds the allocation of cows that minimizes the number of spaceship trips
    via brute force.  The brute force algorithm should follow the following meth

    1. Enumerate all possible ways that the cows can be divided into separate tr
        Use the given get_partitions function in ps1_partition.py to help you!
    2. Select the allocation that minimizes the number of trips without making a
        that does not obey the weight limitation

    Does not mutate the given dictionary of cows.

    Parameters:
    cows - a dictionary of name (string), weight (int) pairs
    limit - weight limit of the spaceship (an int)

    Returns:
    A list of lists, with each inner list containing the names of cows
    transported on a particular trip and the overall list containing all the
    trips
    """
    def goodWeight(result, my_dict, limit):
        for trip in result :
            weight = 0
            for cow in trip:
                if weight + my_dict[cow] <= limit :
                    weight += my_dict[cow]

                else:
                    return False

        return True


    all_results = ps1_partition.get_partitions(cows)
    best_result = None
    max_length = len(cows)
```

```python
    for result in all_results:
        if goodWeight(result, cows, limit) and len(result) < max_length :
            best_result = result
            max_length = len(result)
    return best_result
```

In [8]:

```python
optimal = brute_force_cow_transport(cows_1)
print(cows_1, ': the optimal solution is ', optimal )
len(optimal)
```

```
{'Maggie': 3, 'Herman': 7, 'Betsy': 9, 'Oreo': 6, 'Moo Moo': 3, 'Milkshake': 2,
'Millie': 5, 'Lola': 2, 'Florence': 2, 'Henrietta': 9} : the optimal solution is
[['Henrietta'], ['Florence', 'Lola', 'Oreo'], ['Moo Moo', 'Herman'], ['Milkshak
e', 'Millie', 'Maggie'], ['Betsy']]
```

Out[8]: 5

In [9]:

```python
cows_3 = {'Jesse': 6, 'Maybel': 3, 'Callie': 2, 'Maggie': 5}
print(greedy_cow_transport(cows_3))
print(brute_force_cow_transport(cows_3))
```

```
[('Jesse', 6), ('Maggie', 5), ('Maybel', 3), ('Callie', 2)]
[['Jesse', 'Maybel'], ['Maggie', 'Callie']]
[['Callie', 'Maggie', 'Maybel'], ['Jesse']]
```

Problem 4

We now wish to compare the two algorithms.

In [10]:

```python
import time
def compare_cow_transport_algorithms():
    """
    Using the data from ps1_cow_data.txt and the specified weight limit, run you
    greedy_cow_transport and brute_force_cow_transport functions here. Use the
    default weight limits of 10 for both greedy_cow_transport and
    brute_force_cow_transport.

    Print out the number of trips returned by each method, and how long each
    method takes to run in seconds.

    Returns:
    Does not return anything.
    """
    cows = load_cows('ps1_cow_data.txt')
    #greedy algorithm
    #Time that program starts running.
    start = time.time()
    #run greedy algorithm
    result_greedy = greedy_cow_transport(cows)
    #time that program finishes running, and difference.
    finish = time.time()
    time_greedy = finish - start
    num_greedy = len(result_greedy)
    print('The greedy algorithm requires a minimum of ', num_greedy, 'trips.')
    print('It took ', time_greedy, ' seconds to run')
    #Time that program starts running.
```

```
    start = time.time()
    result_brute = brute_force_cow_transport(cows)
    finish = time.time()
    time_brute = finish - start
    num_brute = len(result_brute)
    print('The brute algorithm requires a minimum of ', num_brute, 'trips.')
    print('It took ', time_brute, ' seconds to run')
compare_cow_transport_algorithms()
```

```
[('Betsy', 9), ('Henrietta', 9), ('Herman', 7), ('Oreo', 6), ('Millie', 5), ('Ma
ggie', 3), ('Moo Moo', 3), ('Milkshake', 2), ('Lola', 2), ('Florence', 2)]
The greedy algorithm requires a minimum of  6 trips.
It took  0.00014972686767578125  seconds to run
The brute algorithm requires a minimum of  5 trips.
It took  0.6065704822540283  seconds to run
```

Write up

We see that, although the greedy algorithm runs considerably faster than the brute force algorithm (several orders of magnitude faster), it does not return an optimal solution.

Why is this? It's clear why the brute force algorithm works. It considers every possible solution (gen_paritions effectively generating the power set of the set of cows, and hence all possiblle sequences of trips), so it must find the best solution. The downside is that this algorithm has an exponential order of growth, and in fact takes considerably longer to run.

Now why doesn't the greedy algorithm return the optimal solution? To see why let's analyze an example. We'll consider the set of cows below:

In [11]:
```
print(sorted(cows_1.items(), key=operator.itemgetter(1), reverse = True))
optimal_greedy = greedy_cow_transport(cows_1)
print('Greedy solution: the optimal result is', optimal_greedy )
len(optimal_greedy)
```

```
[('Betsy', 9), ('Henrietta', 9), ('Herman', 7), ('Oreo', 6), ('Millie', 5), ('Ma
ggie', 3), ('Moo Moo', 3), ('Milkshake', 2), ('Lola', 2), ('Florence', 2)]
[('Betsy', 9), ('Henrietta', 9), ('Herman', 7), ('Oreo', 6), ('Millie', 5), ('Ma
ggie', 3), ('Moo Moo', 3), ('Milkshake', 2), ('Lola', 2), ('Florence', 2)]
Greedy solution: the optimal result is [['Betsy'], ['Henrietta'], ['Herman', 'Ma
ggie'], ['Oreo', 'Moo Moo'], ['Millie', 'Milkshake', 'Lola'], ['Florence']]
```

Out[11]: 6

In this set of cows, Betsy and Henrietta are tied for the biggest. In this case the algorithm selects the cow that comes first alphabetically (Betsy) and sends it out on a trip by itself. Now it selects the other biggest cow (Henrietta) and does the same. In general, the algorithm is designed to *always pick the biggest available cow.*

Now' let's examine the solution returned by the brute force algorithm, which (recall) is guaranteed to be the optimal solution:

In [12]:
```
optimal_brute = brute_force_cow_transport(cows_1)
print('Brute force solution: the optimal result is', optimal_brute )
len(optimal_brute)
```

```
Brute force solution: the optimal result is [['Henrietta'], ['Florence', 'Lola', 'Oreo'], ['Moo Moo', 'Herman'], ['Milkshake', 'Millie', 'Maggie'], ['Betsy']]
```

Out[12]:
```
5
```

Betsy and Henrietta are still sent on their own. Similarly, Herman is still going on a trip with a cow of weight 3 (in this case Moo Moo). In other words, the three heaviest cows behave similarly. The difference comes with Oreo. In the greedy algorithm, Oreo was forced to go on a trip with on of the cows of weight 3 (Moo Moo or Maggie), because the greedy algorithm could only optimize locally. It had to pick the next heaviest cow that satisfied the weight constraint. Then Millie had to shack up with Lola, leavin