Aaron Shehan
Het Patel

Assignment 4

1. See knapsack.py
2. See pseudocode.txt
3.
    a. Perform 0-1 knapsack algorithm, treat 'f' items as 'w' items at first
    b. Here is the table generated with the 0-1 knapsack algorithm
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    [0, 0, 0, 50, 50, 50, 50, 50, 50, 50, 50]
    [0, 0, 25, 50, 50, 75, 75, 75, 75, 75, 75]
    [0, 0, 25, 50, 60, 75, 85, 110, 110, 135, 135]
    [0, 15, 25, 50, 65, 75, 90, 110, 125, 135, 150]
    [0, 15, 25, 50, 65, 75, 90, 110, 125, 135, 150]
    [0, 15, 30, 50, 65, 80, 95, 110, 125, 140, 155]
    [0, 15, 30, 50, 65, 80, 100, 115, 130, 145, 160]
    [0, 15, 30, 50, 65, 80, 100, 115, 130, 145, 160]
    [0, 15, 30, 50, 65, 80, 100, 115, 130, 145, 160]
    [0, 15, 30, 50, 65, 80, 100, 115, 130, 145, 160]
    c. The value found was 160, the full capacity of 10 pounds was used
    d. Get all items that were used to find max value
    e. Items with id: 1, 3, and 7 were used in this case
    f. Check to see if there is any remaining capacity left. In this case there is none.
        i. If there was a remaining capacity, the fractional knapsack algorithm would be used with the remaining 'f' items to generate the max remaining value
    g. Now perform the fractional knapsack algorithm with only the 'f' items provided.
    h. The value for this is 140
    i. Take the max of 160 and 140. 160 is greater than 140, so 160 is the max value
4. Our algorithm is correct because we used the brute force method to compare our algorithms result. Both the algorithms had similar results. We also computed the knapsack problem by hand on small input sizes that where generated using our randomized input generating code to see if our algorithm worked in small cases. Although the run time on our algorithm can be improved, our algorithm calculates the value between two algorithms by combining them, and then using the max function it decides which value is the best from them and displays it as output.
    a. The 0-1 algorithm part of our program takes maximum weight W, the number of items n, and a sequence of objects which each have a value and weight [*vn and wn*], where n is the id of the item. The algorithm stores items in a matrix *c[0..n, 0..W]* where the first row of *c* is filled in from left to right, all the way to the *nth* row. At the end of the algorithm in the matrix, *c[n, W]* contains the maximum value. Our source for this is from http://www.ccs.neu.edu/home/vip/teach/Algorithms/6_dyn_prog/notes/knapsack/knapsack.pdf
    b. Then our algorithm checks If there is a remaining capacity left. If there is some remaining capacity left, the fractional knapsack algorithm is used to compute the

highest remaining value. Below are the steps our fractional knapsack algorithm uses for correctness.

i. Let every remaining 'f' item be labeled i with weight wi and cost vi.
ii. Every item is sorted in descending order by their value (vi/wi).
iii. Suppose there exists an optimal solution in which the complete amount of i was not added to the knapsack.
iv. If the knapsack is not full, then add more of item i. Now the solution has a higher value.
v. Now we assume the knapsack is full.
vi. There must exist some item k != j with vk/wk < vi/wj that is already in the knapsack.
vii. All of i must not all be in the knapsack.
viii. A piece of k, with weight W can be taken out of the knapsack. A piece of i with weight W can replace the piece of k that was removed.
ix. The knapsack value was increased by a value of W(vi/wi) – W(vk/wk), which is greater than zero.
x. Thus, the solution is optimal for the remaining items.
xi. Our source for this is from http://www.columbia.edu/~cs2035/courses/csor4231.F11/greedy.pdf

c. We do the fractional knapsack problem again but only considering the fractional items.
   i. The proof for this part is the same as part b as shown above.
d. The maximum value is the max of part (a +b) and part c.

5. The time complexity is: O(nW).
   a. The code is commented to provide time complexities for different sections in knapsack.py
6.

Let all items have weight W
Let all items have cost C
Let K = amount of items
Let n = m = K/2
Let the limit of the weight be V = (K/2)*W
Since every item has weight W and cost C, every subset is equal to one another. So it does not matter which subset you choose.
Consider the equation V = (K/2)*W. Since there are K items and in the formula K is being divided by 2, then only half of K is sufficient to find the max weight. There are K items each having weight W. So the weight of all items combined is K*W. In this case the max value V is K * W. If we take any subset of half of the items then, there are K/2 items. The new value for V is (K/2)*W. Thus, half of K is needed to find the maximum value.