Lab 9

Exercise 1: The US coin denominations are 1, 5, 10, and 25 cents. To minimize the number of coins needed to make change for n cents, it is always optimal to repeatedly take the biggest coin possible. Write a function `makeChange`(n) which outputs the list of coins one should use to make change for n cents using US coins.

Example solution:

```
def makeChange(n):
    if n == 0:
        return []
    elif n >= 25:
        return [25] + makeChange(n-25)
    elif n >= 10:
        return [10] + makeChange(n-10)
    elif n >= 5:
        return [5] + makeChange(n-5)
    else:
        return [1] + makeChange(n-1)
```

Exercise 2: There are n possible activities that you can do throughout the day. Each activity has a start time a_i and a finish time b_i. You cannot participate in two activities if they overlap. Calculate the most number of activities that you could participate in. That is, write a function `mostActivities`(L) where L is of the form $[[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]]$, and where `mostActivities`(L) outputs the largest number of activities that you could participate in.

This problem can be solved using both memoization and the greedy method. What's the running time you get for each method?

Example solution:

```
# greedy implementation
# sort by the b_i and greedily schedule to take the next activity
# which ends the earliest
def mostActivities(L):
    for x in L:
        x[0],x[1] = x[1],x[0]
    L.sort()
    for x in L:
        x[0],x[1] = x[1],x[0]

    ans = 1
    last = L[0][1]
    for x in L:
```

```
        if x[0] < last:
            continue
        else:
            ans += 1
            last = x[1]
    return ans

# memoization implementation
# f(at, last) is the most number of activities that can be scheduled
# amongst activities L[at], L[at+1],... where the last activity
# scheduled was L[last]
def f(L, at, last, mem):
    if at == len(L):
        return 0
    elif mem[at][last] != -1:
        return mem[at][last]
    mem[at][last] = f(L, at+1, last, mem)
    if L[at][0] >= L[last][1]:
        mem[at][last] = max(mem[at][last], 1 + f(L, at+1, at, mem))
    return mem[at][last]

def mostActivities(L):
    for x in L:
        x[0],x[1] = x[1],x[0]
    L.sort()
    for x in L:
        x[0],x[1] = x[1],x[0]

    mem = []
    for i in xrange(len(L)):
        mem += [[-1]*len(L)]
    return 1 + f(L, 1, 0, mem)
```

The memoized recursive implementation takes $\Theta(n^2)$ time, where n is the length of L. The greedy implementation takes $\Theta(n \log n)$ time: the sort in the beginning dominates the running time, since the rest of the algorithm takes only $O(n)$ time.

Exercise 3: You have a bag that can hold W kilograms of stuff. You're at a grain store where you pay only by the total weight of items you buy and not by the total value of the grains, so you want to pack your bag as well as possible to maximize the total value of goods you take. The store has a separate container for each type of grain, and there are w pounds available of each grain, and those w pounds in total are worth v dollars. The grains have all been powdered, so it is easy to take any number of pounds of a grain which is at most w. Write a function `maxGrain`(L, W) which takes a list L = $[[v_1, w_1], [v_2, w_2], \ldots, [v_n, w_n]]$ of the values and weights of all the grain containers, and outputs a number which is the maximum number of dollars' worth of grain you can take out

of the store in your weight-W bag. Your output should be a `float`. You can convert an `int` x to a `float` in Python by either multiplying it by 1.0, or typing `float`(x).

Example solution: The idea of the solution is to sort the items by v_i/w_i in decreasing order, then greedily take the items which are worth the most value per kilogram.

```
def maxGrain(L, W):
    T = []
    for i in xrange(len(L)):
        L[i] = [1.*L[i][0]/L[i][1]] + L[i]
    L.sort()
    L.reverse()
    ans = 0
    for x in L:
        if W == 0:
            break
        ratio = x[0]
        v = x[1]
        w = x[2]
        take = min(w, W)
        ans += ratio*take
        W -= take
    return ans
```