Problem Set #4 Solution Key

1. A. Before the entire reversal process is complete, we first would like to stack the pancakes in increasing order, with the smallest pancake on the bottom (after this step we simply flip the whole stack and we’re done). At the beginning of our algorithm, label all pancakes as unsorted. To get to the configuration described above, we do the following for each new (unsorted) pancake:
   1) Insert spatula under the smallest unsorted pancake and flip
   2) Insert under the first unsorted (i.e., the first pancake vertically above the sorted set of pancakes) pancake and flip
Repeat until all pancakes are stacked in increasing order (from smallest to biggest, counting vertically up). Flip!

B. Follow the instructions in A ;)
C. 2N-3
D. Depending on your algorithm, this would be a stack that would require the max number of steps to bring to the final state (2N-3 in this example)
E. Depending on your algorithm, this solution will be faster or as fast as your algorithm.

2. A) 17, 17, 10 (plotting all four as functions y (number of steps) of x (=N) would give you an idea of where the graphs intersect, and thus, where (what value of N) one algorithm surpasses another in the number of steps (y)).
B) C, C (simply putting in very large values of N into the respective equations would give you an idea of which algorithms is faster; as we find out, C takes less steps than both B and D)
C) Closest to B – both are polynomials of order 2

3. For this problem, simply pasting the code in the Python console window and pressing Enter should work. In parts C and D, functions are provided along with 1 (in the case of C) and 2 (in the case of D) function calls: here you should first enter the function, and then execute the function calls one at a time. Record the output.

A)
0 1
1 2
2 4
3 8
4 16
5 32
6 64
7 128
8 256
9 512

B) “else”

C) 5040

D) “tset a si siht”
   [7,6,5,4,3,2,1,0]

4. See next page for an example of how this program could have been written. The important thing to keep in mind is the overall structure of the code: we have a while loop which will execute its content over and over
again while “done” is False. Within the loop, we execute code to perform and handle 1) the player’s turn and 2) the computer’s strategy.

// begin my_cookie_game.py

import random
cookies = random.randint(15,24)
print “There are”, cookies, “cookies”
done = False
while (done != True):
    human = raw_input (“How many do you want to take”?)
    if (“123”.find(human) <0) or len(human)!= 1:
        print “Invalid input”, human
    if (human>cookies):
        print “You cant take that many. Try again!”
elif (human <1) or (human>3):
    print “You can only take 1, 2, or 3.”
else:
    cookies = cookies – human
    print “There are”, cookies, “left”
    if (cookies == 1):
        print “You won, I’ll bake the next batch.”
        done = True
    elif (cookies<=4):
        computer = cookies – 1
        print “I’ll take”, computer
        cookies = cookies – computer
        print “leaving”,cookies
        print “I win, you cook the next batch”
        done = True
    elif (cookies < 1):
        print “I guess you want to cook another batch”
        done = True
    elif (cookies > 5) and (cookies <=8):
        computer = cookies – 5
        print “I’ll take “,computer
        cookies= cookies - computer
        print “leaving”, cookies
    else:
        computer = 3
        cookies = cookies – computer
        print “I take”, computer
        print “leaving”,cookies