Work with your neighbor. (This will be graded for participation only.)

1. For each code fragment below, state its worst-case big-O complexity.

a)

```
n = int(input())
while n > 0:
    print(n)
    n -= 1
```

b)

```
n = int(input())
n = n/2
while n > 0:
    print(n)
    n -= 1
```

c)

```
n = int( input() )
if n % 2 == 0:
    for i in range(n):
        x = x + 1
else:
    for i in range(n):
        for j in range(n):
            x = x + 1
```

d)

```
m = 100
for x in numlist1:
    for y in range(m):
        print(x + y)
```

e)

```
def count(char, lc_str):
    num = 0
    for i in range(len(lc_str)):
        if char == lc_str[i]:
            num += 1
    return num
```

2. The following function takes a list of integers and performs some kind of mystery transformation on the list. First, determine what the function does. Second, determine its worst-case big-O complexity. (If you recognize what the function does immediately, don't give the answer to your fellow tablemates. Let them work it out for themselves.)

```
def mystery_one(data):
    for i in range(len(data)):
        for j in range(len(data)-1):
            if data[j] > data[j+1]:
                data[j],data[j+1] = data[j+1],data[j]
```

(**Note:** We may not have time for all of the problems below.)

3. Given a list `alist`, write a function `have_two_greater(alist)` that returns a *new* list that contains all of the elements in `alist` which have at least two elements that are greater than itself.

   Requirements:

   - *Do not use sorted() or sort() or sort the list yourself*
   - *Don't use built-in max()*
   - *Don't use pop() or delete – that is, don't modify the argument list passed in*

What is the complexity of your function? Why?

4. Given a list `alist`, write a function `second_largest(alist)` that finds the second largest element of the list argument `alist`.

   What is the complexity of your function? Why?

5. Given your solution to problem 4, can you write a solution for `have_two_greater(alist)` that is O(n)?