Problem 8.

What is the time efficiency of each of the following Python code fragments. Express this time efficiency using the Big O notation seen in class. The critical operation is "+", i.e., the addition.

a)
```
y = y + 1000
```

The critical operation "+" is executed only once no matter how large our data is (no matter how large **n** is.) So we get a time efficiency of O(1).

b)
```
for each in range(n)
   x = x + y
   y = y + 1000
```

The critical operation "+" is executed twice per iteration of the for loop. The for loop iterates **n** times. So, the total number of time the critical operation "+" is executed is 2n. The Big O notation is O(2n), however, the factor 2 is removed so we get a time efficiency of O(n).

c)
```
count = 0
for each in range(n)
   count = count + 10
for each in range(n)
   count = count + each
```

The critical operation "+" is executed once per iteration of the first for loop. The first for loop iterates **n** times. So, the total number of time the critical operation "+" is executed by the first for loop is n. The same analysis holds for the second loop.
Therefore the time efficiency of the entire code fragment is O(n + n) => O(2n) => O(n) (see b) above).

d)
```
count = 0
for each in range(n)
   for element in range(n)
      count = count + 10
```

The critical operation "+" is executed once per iteration of the inner for loop. The inner for loop iterates **n** times each time the outer for loop iterates once. The outer for loop iterate n times in total. So, the total number of time the critical operation "+" is executed is 1 x n x n -> n^2. Therefore the time efficiency of the entire code fragment is $O(n^2)$.

By the way, the above code fragment is similar to the sorting algorithms we saw in Friday's lecture.