COMP 4030/6030 - Spring 2017
Exam 2

Name: Omer Al Sabri

Note: Do not write on the backs; only front pages are digitized and graded.

1. (*10 points*) Is the running time of the following function is in $O(n^3)$, where n is the number of items in the list L? Explain briefly.

```
def foo(L):
    sum = 0
    for x in L:
        for y in L:
            sum += x + y
    return sum
```

4

No, the running time is $O(n^2)$ because there is nested loop looping for two items in the same list.

2. (*10 points*) Is the running time of the function foo above $\Omega(n)$? Explain briefly.

No

1

3. (*20 points*) Find the running time equation, $T(n)$, of the function bar, and then determine its complexity in terms of Θ:

```
def bar(L):
    if len(L) <= 1:
        return 1
    sum = 0
    for x in L:
        for y in L:
            sum += x + y
    return sum + bar(L[0:len(L)//3])
```

$T(n) = n^2 + 1$ X

$\Theta(n^2)$

10

4. (*20 points*) Use the Master's theorem to find the complexity of the following functions (assuming $T(1) = 1$):

1. $T(n) = n^4 + 9 \cdot T(\frac{n}{3})$
2. $T(n) = n^2 + 10 \cdot T(\frac{n}{3})$

0

5. (*10 points*) Write a Python function to count the frequencies of all characters in a string in $\Theta(n)$. The inputs are: s (a string) and freq (a dictionary). The output is none, but the effect is that freq will store the frequencies of characters in s. Example, after this sequence:

```
f = {}
count("hello", f)
# now f is {'h':1, 'e':1, 'l':2, 'o':1}

def count(s, freq):
   # provide your code here
```

for currentChar in S:
  for nextChar in S:
    if currentChar == nextChar:
      freq += 1

2

6. (*20 points*) Use substitution to find the complexity of this function: $T(1) = 1$, $T(n) = n + 4 \cdot T(\frac{n}{2})$.

7. (*20 points*) Recall that the majority element in a list of n items is an element with frequency greater than $\frac{n}{2}$. We can find the majority element in a list L based on the following strategy:

- Pair up the elements in L into $\frac{n}{2}$ pairs arbitrarily (i.e. in any order you'd like).
- Look at each pair, if both items are the same, keep one. If not, throw both items away. Store the remaining elements in a list M.
- Use the same strategy on the list M.
- Hints: (A) If the list L has a majority element, then that element will also be the majority element of the list M. But the majority element of M might not be the majority element of L; (B) If L has an odd number of elements, remove one of them and place it in M, then L will have an even number of elements.

Do these three things: (1) Implement this strategy in a Python **recursive** program; (2) Write down the running time equation; and (3) Find the running time of your program in terms of Θ.

```
def majority(L):
    # provide your code here
```

(1)

```
for i in L:
majority(L)/2
if a=!b
    a.remove(L)
    b.remove(L)
for j in L:
    elements.append(M)

for ic in L:
    if element %2 != 0
        element.remove(L)
        element.append(M)
```

4

(2) $n^3 + C$

(3) $\Theta(n^3)$