$n^d \log n \quad = \quad n^{\log_b a} \quad > \quad n^d \quad <$

COMP 4030/6030 - Spring 2017
Exam 2

Name: Brandon Hopper

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.

$\Theta(1 + n^2 \cdot 1)$

```
def foo(L):
    sum = 0  1
    for x in L:  n
        for y in L: n
            sum += x + y 1
    return sum
```

Yes, because $O(n^3)$ is in the upper bounds of the equations running time.

10

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

Yes, because Omega shows the lower bounds while Big-O shows upper. Plus the functions Θ is $\approx$ to n^2 which is above Ω.

10

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

$1 \cdot 1 + 1 + (n^2 \cdot 1) + n/3$

$\Theta(1 + 1 + (n^2 \cdot 1) + n/3)$

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

10

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

1. $T(n) = n^4 + 9 \cdot T(\frac{n}{3})$ $a = 9, b = 3, d = 4$ $9 < 3^4$ $\Theta(n^4)$
2. $T(n) = n^2 + 10 \cdot T(\frac{n}{3})$ $a = 10, b = 3, d = 2$ $10 > 3^2$ $\Theta(n^{\log_3 10})$

20

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
```

```
While len(s) != 0:
    [illegible]
    f[s[0]] = f[s[0]] + 1
    s.Remove[0]
```

9

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

$T(n) = n + 4T(\frac{n}{2})$

$T(\frac{n}{2}) = \frac{n}{2} + 4^2 T(\frac{n}{2^2})$

$T(\frac{n}{2^2}) = \frac{n}{2^2} + 4^3 T(\frac{n}{2^3})$

$T(\frac{n}{2}) = \frac{n}{2} + 4^2 T(\frac{n}{2^2})$

$T(\frac{n}{2^2}) = \frac{n}{2^2} + 4^3 T(\frac{n}{2^3})$

5

$\Theta(n^{\log_2 4})$

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
```

```
M = []          # 1
i = 1           # 1
while i <= len(L):   # n/2
    If L[i-1] == L[i]:
        M.append(L[i])
    i = i + 2

If Len[M] == 1:   # 1
    return M[0]   # 1

If Len[M] == 0:   # 1
    return "Nothing"   # 1

return majority(M)   # n/2
```

15

$O(1 + 1 + n/2 + 1 \cdot 1 + 1 \cdot 1 + n/2)$

$\Theta(2 + n/2 + 2 + n/2)$