COMP 4030/6030 - Spring 2017
Exam 2

Name: Madilan Pope

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
```

b

n^2

$T(n) = c \times n^2 + b$

$c \times n^2 + b \le (c+b)n^3$ for all values $n>1$;
therefore,
yes the function is in $O(n^3)$
(also in $O(n^2)$ when $c = c+b$)

10

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

$c \times n^2 + b \ge 1 \times n$ for all values $n > 1$;
therefore,
yes, the function is in $\Omega(n)$
(also in $\Omega(n^2)$ when $c=1$)

10

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])
```

$T(1)=1$

b

n^2

20

$T(n) = c \times n^2 + T(\frac{n}{3}) + b = n^2 + T(\frac{n}{3})$

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

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

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

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

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

...

$= n^2(1 + \frac{1}{9} + \frac{1}{9^2} + \dots + \frac{1}{9^{k-1}}) + T(\frac{n}{3^k})$

$\frac{n}{3^k} = 1$

$n = 3^k$

$\log_3 n = k$

$= n^2\left(\frac{\frac{1}{a^k} - 1}{\frac{1}{a} - 1}\right) + 1 = n^2\left(\frac{1 - \frac{1}{9^k}}{1 - \frac{1}{9}}\right) + 1 =$

$a<1$ so as $k\uparrow$ approaches constant

$\boxed{T(n) = n^2 + 1}$

$n^2 + 1 \le 2n^2$ for all values $n \ge 1$ $O(n^2)$
$n^2 + 1 \ge n^2$ for all values $n > 1$ $\Omega(n^2)$
proves $\Theta(n^2)$

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

1) $a = 9$, $b = 3$, $d = 4$

$\log_3 9 = 2$, $d = 4$, so $\Theta(n^4)$

2) $a = 10$, $b = 3$, $d = 2$

$\log_3 10 > 2$, $d = 2$, so $\Theta(n^{\log_3 10})$

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:

did we go over dictionaries?

no return

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

def count(s, freq):
    # provide your code here
    for item in s:
        if item in freq:
            freq(item: count +1)
        else:
            freq(item:1)
```

15

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

$T(n) = n + 4 \cdot T(\frac{n}{2}) \quad T(1) = 1$

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

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

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

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

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

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

20

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

...

$= n(1 + 2 + 2^2 + 2^3 + \dots + 2^{k-1}) + 4^k T(\frac{n}{2^k})$

$= n(\frac{2^k - 1}{2 - 1}) + 4^k T(\frac{n}{2^k}) \qquad \frac{n}{2^k} = 1$

$n = 2^k$

$= n(\frac{n-1}{1}) + 4^{\log_2 n} \cdot 1 \qquad \log_2 n = k$

$= n^2 - n + n^2$

$\boxed{T(n) = 2n^2 - n}$

$2n^2 - n \le 2n^2$ for all values $n > 1$ $O(n^2)$

$2n^2 - n \ge n^2$ for all values $n > 1$ $\Omega(n^2)$

proves $\Theta(n^2)$

majority $> \frac{n}{2}$

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.

1)

```
def majority(L):
    # provide your code here
    if len(L) <= 1:          T(1)=1
        return L
    m = []
    if len(L)%2==1:
        m += L[len(L)-1]
    x, y = 0, 1                        b
    while y < len(L):
        if L[x] == L[y]:
            m += L[x]                  n
        x += 2
        y += 2
    major_element = majority(m)  - T(n/2 at most if everything is same)
    count = 0
    for item in L:
        if major_element == item:      n
            count += 1
    if count > len(L)/2:
        return major_element
    else:                              b
        return None
```

20

2) $T(n) = c \times n + d \times n + b + T(\frac{n}{2})$

$T(n) = (2c \times d)n + b + T(\frac{n}{2})$

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

$T(1) = 1$

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

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

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

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

...

$= kn + T(\frac{n}{2^k})$ $\quad \frac{n}{2^k} = 1$

$n = 2^k$

2) $= n\log_2 n + 1$ $\quad k = \log_2 n$

3) proves $\Theta(n\log n)$

$O(n\log n) \to n\log n + 1 \leq 2n\log n$ for all values $n \geq 1$

$\Omega(n\log n) \to n\log n + 1 \geq n\log n$ for all values $n \geq 1$