Midterm Review

1. Put the growth rates in order from smallest to largest:
   $n^{1/4}$, $2n+8$, 4^{1000}, $n^4 + 3n^2$, $3n^3 + 4n^2$, $n^{1/6}$
   $\Rightarrow 4^{1000} \rightarrow n^{1/6} \rightarrow n^{1/4} \rightarrow 2n + 8 \rightarrow 3n^3 + 4n^2 \rightarrow n^4 + 3n^2$

   $3\log_5 n^6$, $14\log_5 n^3$, $n^{1/3}$, $4(\log n)^4$
   $\Rightarrow 3\log_5 n^6 == 14\log_5 n^3 \rightarrow 4(\log n)^4 \rightarrow n^{1/3}$

   3^{3n+10}, 3^{3n}, 2^{500}, $n!$, $3^{n\log n}$
   $\Rightarrow 2^{500} \rightarrow 3^{3n} == 3^{3n+10} \rightarrow n! \rightarrow 3^{n\log n}$

2. Analyze the worst case running times →

   a.
   ```
   cnt = 0;
   for (i = 1, i < n, i++) {
           for (j = 1, j < n, j += 2) {
                   cnt++;
           }
   }
   ```

   ⇒ For each outer loop (which runs n - 1 times), the inner loop runs n/2 times
   The runtime is $O(n^2)$

   b.
   ```
   cnt = 0;
   for (i = 1, i < n, i *= 3) {
           for (j = 1, j < n, j += 2) {
                   cnt++;
           }
   }
   ```

   ⇒ For each outer loop (which runs $\log_3 n$ times), the inner loop runs n/2 times
   The runtime is $O(n\log_3 n)$

   c.
   ```
   cnt = 0;
   for (i = 1, i < n, i++) {
           for (j = 1, j < n, j *= 2) {
                   cnt++;
           }
   }
   ```

   ⇒ For each outer loop (which runs n-1 times), the inner loop runs $\log_2 n$ times
   The runtime is $O(n\log_2 n)$

3. Solve the recurrence relations

$T(n) = 4T(n-1) + 4$

⇒ By unrolling, $T(n) = 4T(n-1) + 4 = 4^2T(n-2) + 4 + 1 = \ldots = 4^{n-1}T(1) + 4^{n-2} + 4^{n-3} + \ldots + 1$

If $T(1) = \theta(1)$ ⇒ Applying Geometric Progression formula on sum of terms will be $4^n*c = \theta(4^n)$

$T(n) = 3T(n/5) + \Theta(n^2)$

⇒ Master's Theorem - Case 3 → a = 3, b = 5 → $T(n) = \theta(n^2)$

$T(n) = 4T(n/3) + \Theta(1)$

⇒ Master's Theorem - Case 1 → a = 4, b = 3 → $T(n) = \theta(n \wedge (\log_3 4))$

$T(n) = 5T(n/5) + \Theta(n)$

⇒ Master's Theorem - Case 2 → a = 5, b = 5 → $T(n) = \theta(n\log n)$

4. Can master theorem be applied to: $T(n) = 9T(n/3) + n^2 3^n$? Show first that you can use one of the cases by showing that the limit of f(n)/g(n + or - epsilon in the exponent) as n -> infinity satisfies the particular case you are using. If you can use the master theorem then give a value of epsilon(e) > 0, and c > 1 that satisfy the equation.

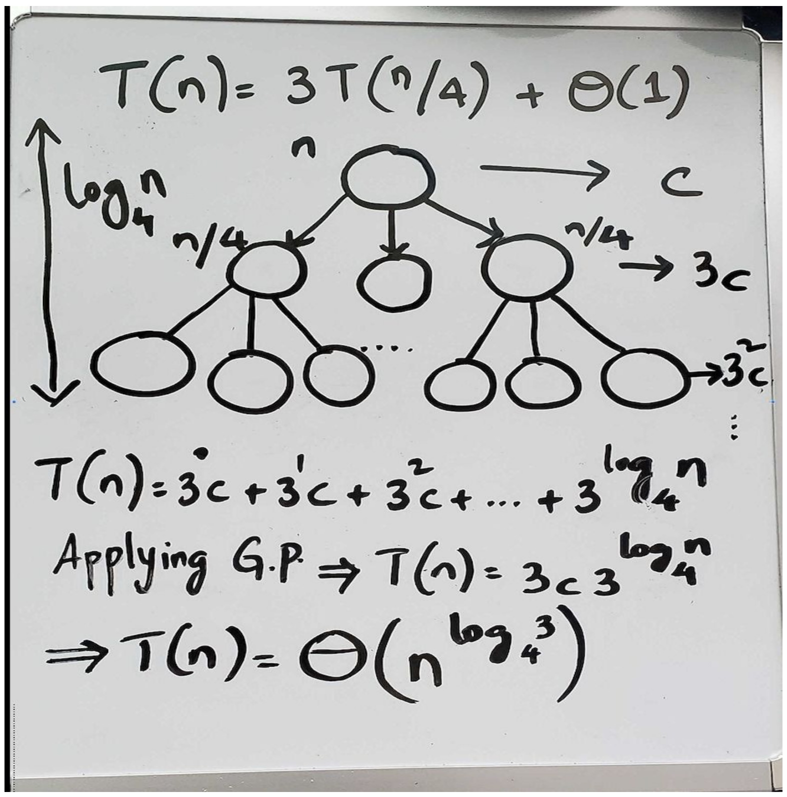
Break down the problem into terms that we know: aT(n/b) + f(n)

1) $n^{log\_b(a)} = n^{log\_3(9)} = n^2$
2) $f(n) = n^2 3^n$
3) What is the relationship between n^2 and $n^2 3^n$
4) Here you should see that we want to apply the third case of the master theorem
   $f(n) = \mathbf{\Omega}(n^{log\_b(a)+e})$ where e > 0. The other two cases do not apply, can you explain why?
5) If this case is true, then the lim as n -> infinity of f(n)/(g(n) plus e in the exponent) will be infinity. So, we check this value: lim $(n^2 3^n)/(n^{2+e})$
   Remove the n^2 from top and bottom lim $(3^n)/(n^e)$
6) Here it is common knowledge that 3^n grows faster than n^e so the limit is equal to infinity. This shows that we can use case 3. For any e > 0, so e = 1.
7) Finally, we need to show that a*f(n/b) <= c*f(n)
   $9 * (n^2/3^2) * 3^{n/3} <= c* n^2 3^n$, Reduce to remove 9 from the front of the left equations
   $n^2 * 3^{n/3} <= c* n^2 3^n$,Remove n^2 from both sides
   $3^{n/3} <= c*3^n$, Divide both sides by 3^n. We don't need to flip the inequality because we have all values to be positive numbers
   $3^{-(2n/3)} <= c$, Here we can see that as n goes to infinity this value becomes very small
   Set c to some value, c = 0.99
8) Here you should see that this holds for sufficiently large values of n.
9) So our runtime is $T(n) = \Theta(n^2 3^n)$

5. Recursion Tree Method:

a.
```
def fun(n) {
        if (n >= 0) {
                print(" … ");
                fun(n/4);
                fun(n/4);
                fun(n/4);
        }
}
```

$\Rightarrow$ T(n) = 3T(n/4) + θ(1)



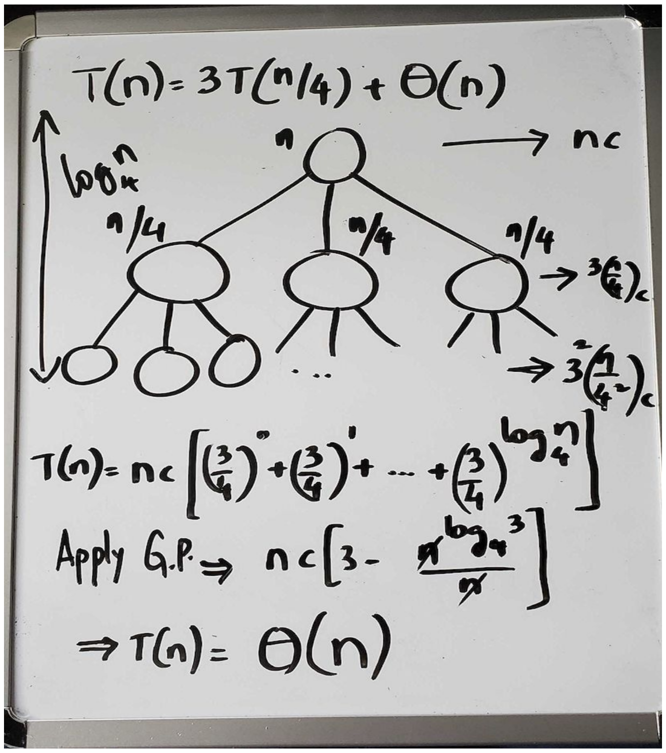
$T(n) = 3^0c + 3^1c + 3^2c + \dots + 3^{\log_4 n}$

Applying G.P. $\Rightarrow T(n) = 3c\,3^{\log_4 n}$

$\Rightarrow T(n) = \Theta(n^{\log_4 3})$

b.
```
def fun(n) {
        if (n >= 0) {
                for i ← 1 to n/2 {
                        print(" … ");
                }
                fun(n/4);
                fun(n/4);
                fun(n/4);
        }
}
```

$T(n) = 3T(n/4) + \theta(n)$


$T(n) = 3T(n/4) + \Theta(n)$
$\log_4^n$
n
$\rightarrow nc$
$n/4$
$n/4$
$n/4$
$\rightarrow 3(\frac{n}{4})c$
$\rightarrow 3^2(\frac{n}{4^2})c$
$\cdots$

$$T(n) = nc\left[\left(\frac{3}{4}\right)^0 + \left(\frac{3}{4}\right)^1 + \cdots + \left(\frac{3}{4}\right)^{\log_4^n}\right]$$

$$\text{Apply G.P.} \Rightarrow nc\left[3 - \frac{\cancel{n}^{\log_4 3}}{\cancel{n}}\right]$$

$$\Rightarrow T(n) = \Theta(n)$$

6. Unenrolling method: Solve the following:

```
def fun( A[1, 2, 3, …, n] ) {
        // base case condition
        cubicSum = 1;

        for (i ← 1 to n/2) {
                for (j ← 1 to n/3) {
                        for (k ← 1 to n/4) {
                                cubicSum++;
                        }
                }
        }

        return 4 + fun( A[1, 2, 3, …, n-3] );
}
```

⇒ $T(n) = T(n-3) + n^3$
$T(n) = T(n-6) + (n-3)^3 + n^3$
$T(n) = T(1) + ((n - (n-3))^3 + \dots + (n-6)^3 + (n-3)^3 + n^3$

These are sum of cubes of natural numbers which are multiples of 3
This sum is less than sum of cubes of first n natural numbers → $T(n) = (n^2 . (n+1)^2) / 4$
⇒ $T(n) = \theta(n^4)$

7. Following CLRS pseudo code for partitioning in Quick Sort, solve the following:

Array ← [10, 35, 64, 50, 5, 80, 60]

a. Value of pivot in the call partition(A, 1, 7) [based on 1-indexing]
⇒ 60
b. Index of the pivot value at the end of the above call
⇒ 5
c. On the next recursion call, which subarray does the partition() function evaluate?
⇒ [10, 35, 50, 5]

8. Consider the hash function H(k) = floor(200k + 512) for all keys k for a table size of 200. You have 3 applications →

A1: Keys are generated uniformly at random in the range [0.6, 1.6]
A2: Keys are generated uniformly at random in the range [0.2, 0.8] U [1.2, 1.8]
A3: Keys are generated uniformly at random in the range [0, 2]

a. For which application does the hash function perform worse?
⇒ Application 1 performs worse as it consumes the least number of available slots in the hash table.

b. In which applications, does the hash function satisfy uniform hashing property?
⇒ Application 3 satisfies uniform hashing property as keys generated are uniformly distributed over the range of values in the hash table.

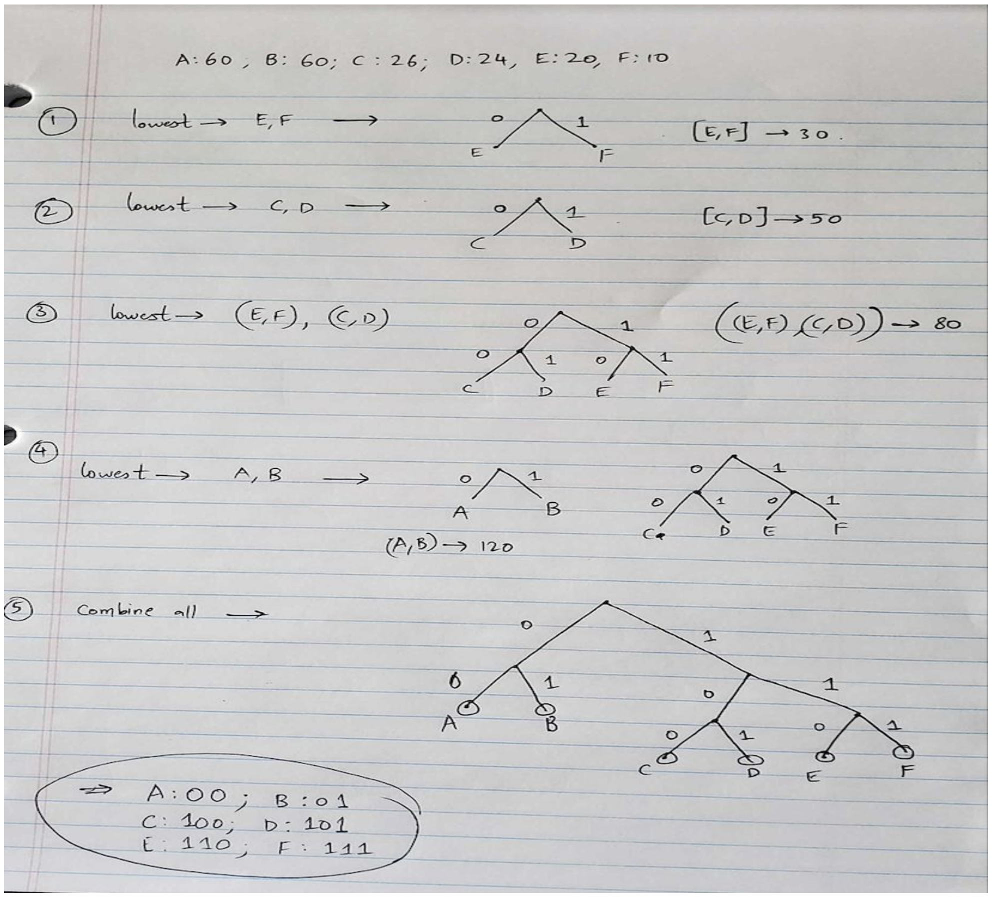
c. Suppose you have n keys in total for each application. What is the load factor α for each application?
⇒ A1: n/50; A2: n/60; A3: n/100

9. Check whether the set of prefix-codes are valid or not?

- x - 1; y - 01; z - 001; w - 000
⇒ Valid
- a - 00; b - 01; c - 011; d - 100
⇒ Not valid (b is a prefix of c)
- a - 000; b - 001; c - 010; d - 011; e - 100; f - 101; g - 110; h - 111
⇒ valid

10. Given a set of characters and their corresponding frequencies, design a possible Huffman code In this example, the larger values will go on the left and the smaller on the right:

{
A: 60,
B: 60,
C: 26,
D: 24,
E: 20,
F: 10
}



11. Given prefix codes → {a: 0, b: 10; c: 110; d: 111}; decode the string “011010010”
⇒ ‘acbab’

12. Activity Selection Problem:
We have a certain number of activities with start and end times. Given these, we need to finish as many activities as possible such that no two activities can conflict in terms of their time intervals.
We have 3 types of algorithms to solve this problem and only one of them works. Provide a counter example for the other 2 algorithms as to why they don’t work always:

Example (start time, end time): (4, 5); (1, 7); (7, 9); (0, 3)
We can complete tasks (0, 3); (4, 5); (7, 9) → 3 tasks in total

Algorithm 1: Sort by duration length and finish shortest tasks first.
⇒ (1, 4); (3, 5); (4, 7) → Optimal is 2 tasks
If we sort by duration length ⇒ (3, 5); (1, 4); (4, 7) and only (3, 5) can be performed

Algorithm 2: Sort tasks by earlier start times and break ties with shorter duration tasks
⇒ (1, 8); (2, 3); (4, 5); (6, 7) → if we sort by earliest start times ⇒ only (1, 8) can be performed where as the optimal number of tasks would be 3 → (2, 3); (4, 5); (6, 7)

Algorithm 3: Sort tasks by earlier end times and break ties with shorter duration tasks