CPS688 Midterm 1 Review:

- Midterm Consist of True and False
- Loop Analysis
- No Induction, No 3 sums, No math
  - Answers are simple 1 line answers.

Lets start off with Loop Analysis
- Prof Painter (youtube)*

#1)

What is the running time of the following code snippet?

```
1: x = 0                    C1
2: for i = 1 to 25 do       C2
3:     x = x + i            C3
4: end for
```

T= C1 + 25C2 + 25C3

- Need to count how many times each line runs

*Google: How long does assignment take, How long does adding take, ≥, ≤, how long does retrieval take how long does each of the thing the code must do take and directly time it.

#2)

x=0 is irrelevant

What is the running time of the following code snippet?

```
1: x = 0
2: for i = 1 to n do        C1
3:     x = x + i            C2
4: end for
```

$T(n) = c_1 + nc_2$

So, $T(n) \in \Theta(n)$.

What is the running time of the following code snippet?

```
1: x = 0
2: for i = 1 to n do        C1 (crossed out)
3:     x = x + i            C2
4: end for
```

$T(n) = c_1 + nc_2$ (c_1 crossed out)

So, $T(n) \in \Theta(n)$.

What is the running time of the following code snippet?

```
1: x = 0
2: for i = 1 to n do
3:     x = x + i            C
4: end for
```

$T(n) = nc$

So, $T(n) \in \Theta(n)$.

$T(n)$ = Time complexity of an Algorithm

$\Theta(n)$ = (Theta notation)

Represents tight or asymptotic bound on growth rate of algorithm running time.

$\in$ = Is the Elements of

Chapter 1 : Introduction to Mathematical Background.

★ Order Notation: (Asymptotic Complexity

Big O → O(n) — Upper Bound

Omega → Ω(n) — Lower Bound

Theta → Θ(n) — Tight Bound (Inbetween) Runtime for both Lower and Upper

Small o → o(n) — Upper Bound (Grows slower)

Small Omega → ω(n) — Lower Bound (Grows Faster)

Formula Sheet

Definition 1. $f(n) \in O(g(n))$ if **there exists** constant $c > 0$ and $n_0 > 0$ such that

$$0 \leq f(n) \leq cg(n),$$

for all $n \geq n_0$. The complexity of f is **not higher** than that of g.

→ Big O Notation: Defines upper bound f(n) does not grow faster then g(n) after certain Point no.

Definition 2. $f(n) \in \Omega(g(n))$ if **there exists** constant $c > 0$ and $n_0 > 0$ such that

$$0 \leq cg(n) \leq f(n),$$

for all $n \geq n_0$. The complexity of f is **not lower** than that of g.

→ Big Omega Notation: Defines lower bound f(n), indicating that f(n) grows atleast as fast as g(n) after certain point no.

Definition 3. $f(n) \in \theta(g(n))$ if $f(n)$ is both $O(g(n))$ and $\Omega(g(n))$.

→ Big Theta Notation: f(n) is both O(g(n)) and Ω(g(n)) meaning f(n) grows at same rate as g(n).

Definition 4. $f(n) \in o(g(n))$ if **for all** constant $c > 0$, **there exists** $n_0 > 0$ such that

$$0 \leq f(n) \leq cg(n),$$

for all $n \geq n_0$. The complexity of f is **lower** than that of g.

→ Little o Notation: f(n) grows strictly slower then g(n). f(n) will be less then c*g(n) for all n larger then some no.

Definition 5. $f(n) \in \omega(g(n))$ if **for all** constant $c > 0$, **there exists** $n_0 > 0$ such that

$$0 \leq cg(n) \leq f(n),$$

for all $n \geq n_0$. The complexity of f is **higher** than that of g.

→ Little Omega ω Notation: Opposite of little o Notation f(n) grows strictly faster then g(n).

Fact 1. Suppose that $f(n) > 0$ and $g(n) > 0$ for all $n \geq n_0$. Suppose that

$$L = \lim_{n \to \infty} \frac{f(n)}{g(n)}.$$

Then

$$f(n) \in \begin{cases} o(g(n)) & \text{if } L = 0, \\ \Theta(g(n)) & \text{if } 0 < L < \infty, \\ \omega(g(n)) & \text{if } L = \infty. \end{cases}$$

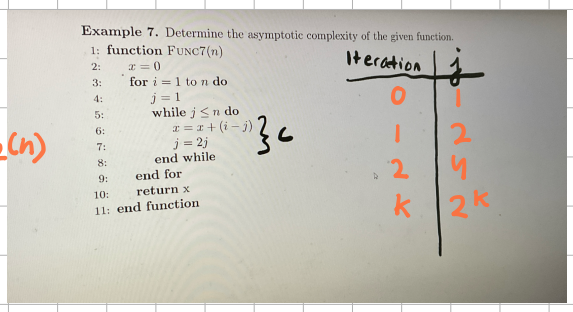
→ If Limit of f(n)/g(n) as n approaches infinity exists, you can use it to determine which notations apply to f(n).

Fact 2. Let $f : \mathbb{N} \to [0, \infty)$ and $g : \mathbb{N} \to [0, \infty)$ be two functions. Then we have that $O(f(n) + g(n)) = O(\max\{f(n), g(n)\})$. More specifically, let $p : \mathbb{N} \to [0, \infty)$ and $q : \mathbb{N} \to [0, \infty)$ be defined by $p(n) = f(n) + g(n)$ and $q(n) = \max\{f(n), g(n)\}$, respectively. Consider an arbitrary function $t : \mathbb{N} \to [0, \infty)$, then we have that $t(n) \in O(p(n))$ **if and only if** $t(n) \in O(q(n))$.

→ Relates to sum of two functions, stating the complexity of the sum is bounded by the maximum complexity of the individual functions.

Example 1:

While Loop Analysis:



Example 7. Determine the asymptotic complexity of the given function.

```
1: function FUNC7(n)
2:     x = 0
3:     for i = 1 to n do
4:         j = 1
5:         while j ≤ n do
6:             x = x + (i − j)
7:             j = 2j
8:         end while
9:     end for
10:    return x
11: end function
```

Iteration	j
0	1
1	2
2	4
k	2^k

$c \log_2(n)$

Iteration table only for while loop
j starts at 1.

$2^k = n$

$k = \log_2(n)$

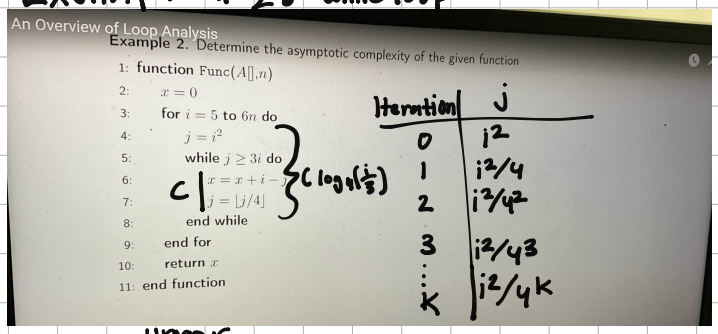
You find entire runtime of while loop

$$T(n) = \sum_{i=1}^{n} c \log_2(n)$$

$$= cn \log_2(n)$$

$T(n) \in \Theta(n \log(n))$

Example #2: while loop



An Overview of Loop Analysis

Example 2. Determine the asymptotic complexity of the given function

```
1: function Func(A[],n)
2:     x = 0
3:     for i = 5 to 6n do
4:         j = i^2
5:         while j ≥ 3i do
6:             x = x + i − j
7:             j = ⌊j/4⌋
8:         end while
9:     end for
10:    return x
11: end function
```

Iteration	j
0	i^2
1	$i^2/4$
2	$i^2/4^2$
3	$i^2/4^3$
⋮	
k	$i^2/4^k$

$c \log_4(i/3)$

Stops when $\frac{i^2}{4^k} = 3i \cdot 4^{k'}$

$\frac{i}{3} = 4^{k'}$

$k' = \log_4(i/3)$

upper

$$T(n) = \sum_{i=5}^{6n} c \log_4(i/3)$$

upper = lower bound

$$\leq \sum_{i=1}^{6n} c \log_4(i/3)$$

$$\leq \sum_{i=1}^{6n} c \log_4\left(\frac{6n}{3}\right)$$

$$\leq 6n \cdot c \log_4(2n) \rightarrow O(n \log(n))$$

Lower

$$T(n) \sum_{i=5}^{6n} c \log_4(i/3)$$

$$\geq \sum_{i=5}^{6n} c \log_4(i/3)$$

$$\geq \left(\sum_{i=3n}^{6n} c \log_4(3n/3)\right)$$

$$\geq c(6n - 3n + 1)\, c \log_4(n) \rightarrow \Omega(n \log(n))$$

$\therefore T(n) \in \Theta(n \log(n))$

Example 3

Example 11. Determine the asymptotic complexity of the given function.

```
1:  function FUNC11(n)
2:      x = 0
3:      i = 1
4:      while i ≤ n do
5:          j = 1
6:          while j ≤ 2i do
7:              x = x + (i − j)
8:              j = j + 4
9:          end while
10:         i = 3i
11:     end while
12:     return x
13: end function
```

Note: $c = \frac{ci}{2}$

Ci — $3c'$

Iteration	J
0	1
1	5
2	9
k	1+4k

Stops when $1+4k = 2i$

$k = \frac{2i-1}{4}$

$k = 2i/4 = i/$

$T(n) = C(1) + C(3) + C(9) + Cn$

$T(n)$

Iteration	i
0	1
1	3
2	9
⋮	⋮
0	3

stops when

$3^2 = n$

$r = \log_3$