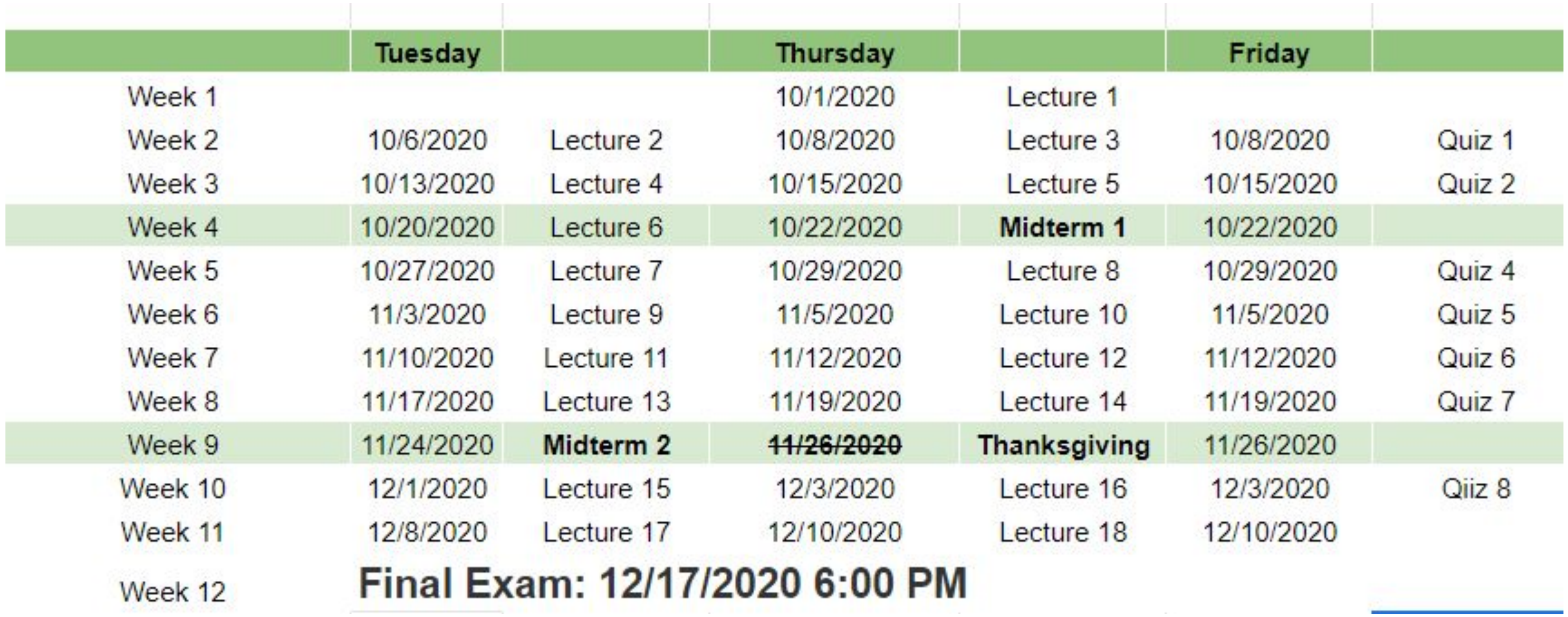
ECS 122A

Operational stuff

-PTA

-Prereq. Petitions

Key Dates



	Tuesday		Thursday		Friday	
Week 1			10/1/2020	Lecture 1		
Week 2	10/6/2020	Lecture 2	10/8/2020	Lecture 3	10/8/2020	Quiz 1
Week 3	10/13/2020	Lecture 4	10/15/2020	Lecture 5	10/15/2020	Quiz 2
Week 4	10/20/2020	Lecture 6	10/22/2020	**Midterm 1**	10/22/2020	
Week 5	10/27/2020	Lecture 7	10/29/2020	Lecture 8	10/29/2020	Quiz 4
Week 6	11/3/2020	Lecture 9	11/5/2020	Lecture 10	11/5/2020	Quiz 5
Week 7	11/10/2020	Lecture 11	11/12/2020	Lecture 12	11/12/2020	Quiz 6
Week 8	11/17/2020	Lecture 13	11/19/2020	Lecture 14	11/19/2020	Quiz 7
Week 9	11/24/2020	**Midterm 2**	**~~11/26/2020~~**	**Thanksgiving**	11/26/2020	
Week 10	12/1/2020	Lecture 15	12/3/2020	Lecture 16	12/3/2020	Qiiz 8
Week 11	12/8/2020	Lecture 17	12/10/2020	Lecture 18	12/10/2020	
Week 12	**Final Exam: 12/17/2020 6:00 PM**					

Today

Intro on topics covered

Background on what you should be coming in with

Topics Covered -Part 1

1. Growth functions and asymptotic notation
    a. **I.e. Big-O, Big-Ω, Big-ϴ**
2. Growth functions for recursive equations/algorithms

Topics Covered -Part 2

3.) Divide and Conquer

4.)Greedy Algorithms

5.) Dynamic Programming

Topics Covered -Part 3

Apply design techniques we learned to a specific segment/specific context

6.) Graph Algorithms

Topics Covered -Part 4

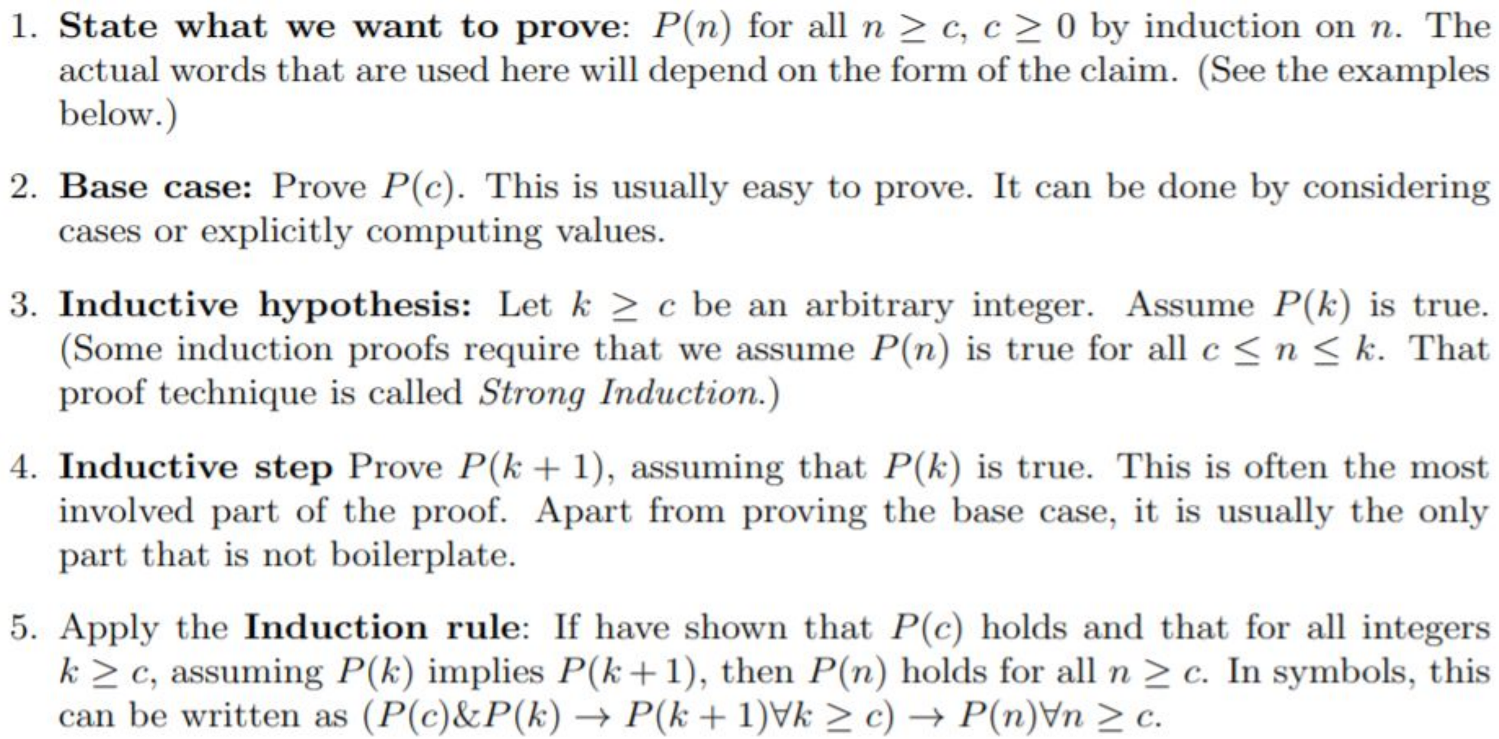
Finally we will look at ability to solve things efficiently.

7.) Define P, NP, NP-Hard and NP-Complete

Knowledge you should come in with

- Inductive proofs
- Basic code Analysis
- Definition of Big-O, and limit lemma
- Studied different algorithms that achieved the same goal
  - I.e. insertion sort, selection sort, quick sort, merge sort.
- Data Structures
  - Lists, queue, stack, Heaps, Disjoint Sets
- Graph Algorithm Intro
  - BFS, DFS, MST Krsukals, shortest path Dikstra

Light Review- Inductive Proof



1. **State what we want to prove**: $P(n)$ for all $n \geq c$, $c \geq 0$ by induction on n. The actual words that are used here will depend on the form of the claim. (See the examples below.)
2. **Base case:** Prove $P(c)$. This is usually easy to prove. It can be done by considering cases or explicitly computing values.
3. **Inductive hypothesis:** Let $k \geq c$ be an arbitrary integer. Assume $P(k)$ is true. (Some induction proofs require that we assume $P(n)$ is true for all $c \leq n \leq k$. That proof technique is called *Strong Induction*.)
4. **Inductive step** Prove $P(k+1)$, assuming that $P(k)$ is true. This is often the most involved part of the proof. Apart from proving the base case, it is usually the only part that is not boilerplate.
5. Apply the **Induction rule**: If have shown that $P(c)$ holds and that for all integers $k \geq c$, assuming $P(k)$ implies $P(k+1)$, then $P(n)$ holds for all $n \geq c$. In symbols, this can be written as $(P(c) \& P(k) \rightarrow P(k+1) \forall k \geq c) \rightarrow P(n) \forall n \geq c$.

Prove $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, for all $n \geq 1$.

Base case: $\sum_{i=1}^{1} i = 1$ and $\frac{1 \cdot (1+1)}{2} = 1$. $P(1)$ is true.

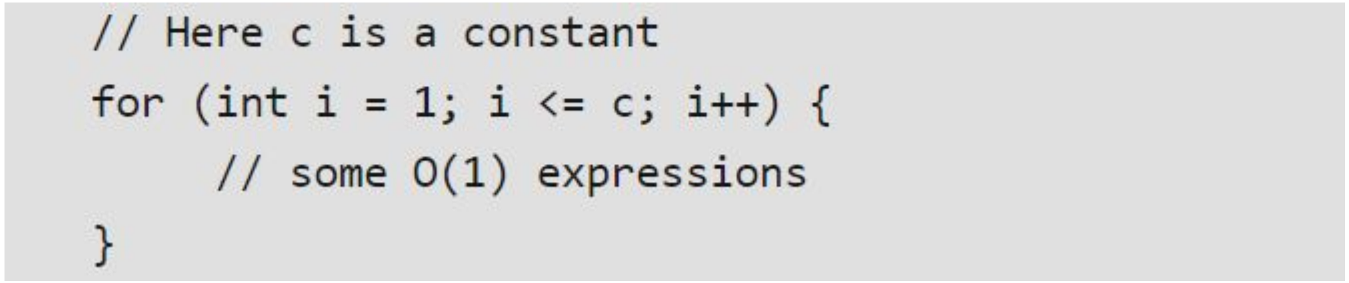
Inductive hypothesis: Let $k \geq c$ be an arbitrary integer. Assume $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$.

Inductive step: We want to prove $\sum_{i=1}^{k+1} i = \frac{(k+1)(k+2)}{2}$.

$$\begin{aligned}
\sum_{i=1}^{k+1} i &= 1 + 2 + \ldots k + (k+1) \\
&= (1 + 2 + \ldots k) + (k+1) \\
&= \frac{k(k+1)}{2} + k + 1 \\
&= \frac{k(k+1) + 2(k+1)}{2} \\
&= \frac{(k+1)(k+2)}{2}
\end{aligned}$$

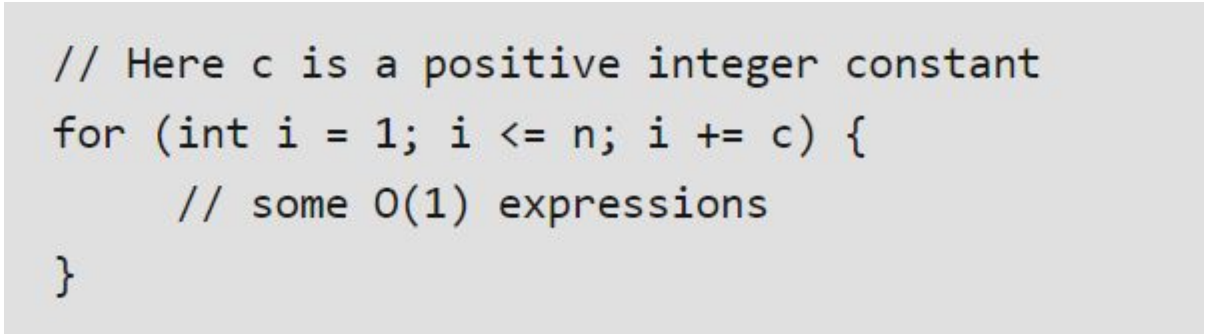
By the **Induction rule**, $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, for all $n \geq 1$.

Basic Code analysis -O(1)



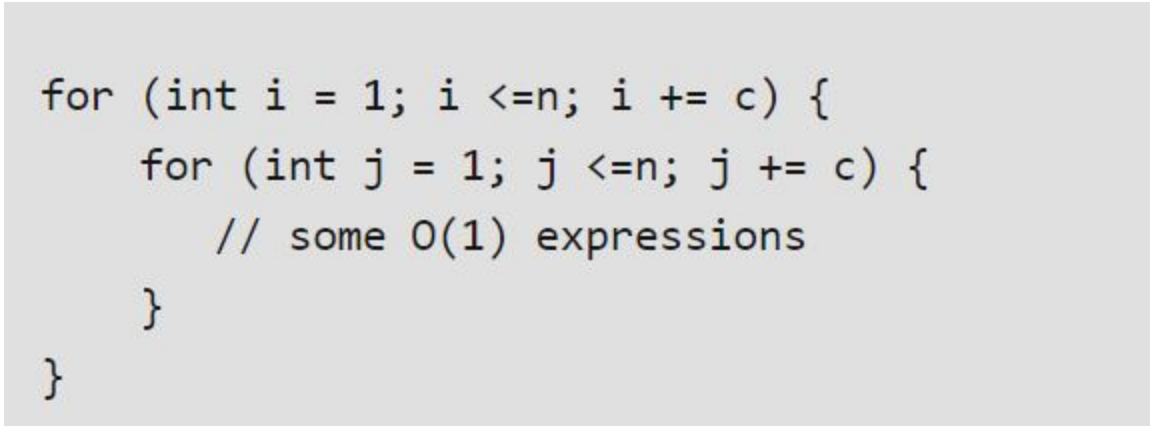
```
// Here c is a constant
for (int i = 1; i <= c; i++) {
     // some O(1) expressions
}
```

Basic Code analysis -O(n)



```
// Here c is a positive integer constant
for (int i = 1; i <= n; i += c) {
     // some O(1) expressions
}
```

Basic Code analysis -O(n^2)



```
for (int i = 1; i <=n; i += c) {
    for (int j = 1; j <=n; j += c) {
       // some O(1) expressions
    }
}
```

LETS start ECS 122A

Definition of algorithm

Step1: Well specified computation problem

An algorithm is a tool for solving this problem.

Step 2: Algorithm

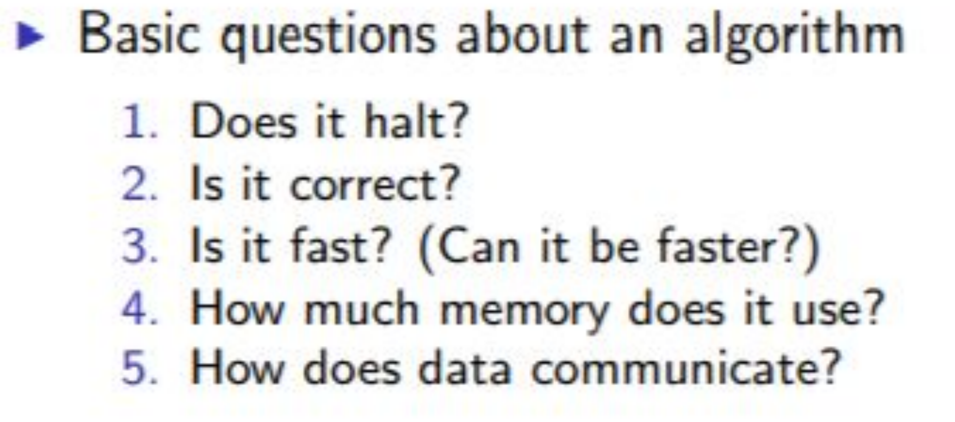
A well-defined procedure for transforming some input into desired output.

Examples

- Example1:
  - Computation problem: Sort a list of objects
  - Algorithm: Insertion Sort
- Exampe 2:
  - Computation problem: Find the shortest route between two cities
  - Algorithm: Dijstra’s shortest path algorithm

What is always needed? Decide what the input will look like, what the output will look like.

What do we look at when designing an algorithm



- Basic questions about an algorithm
  1. Does it halt?
  2. Is it correct?
  3. Is it fast? (Can it be faster?)
  4. How much memory does it use?
  5. How does data communicate?

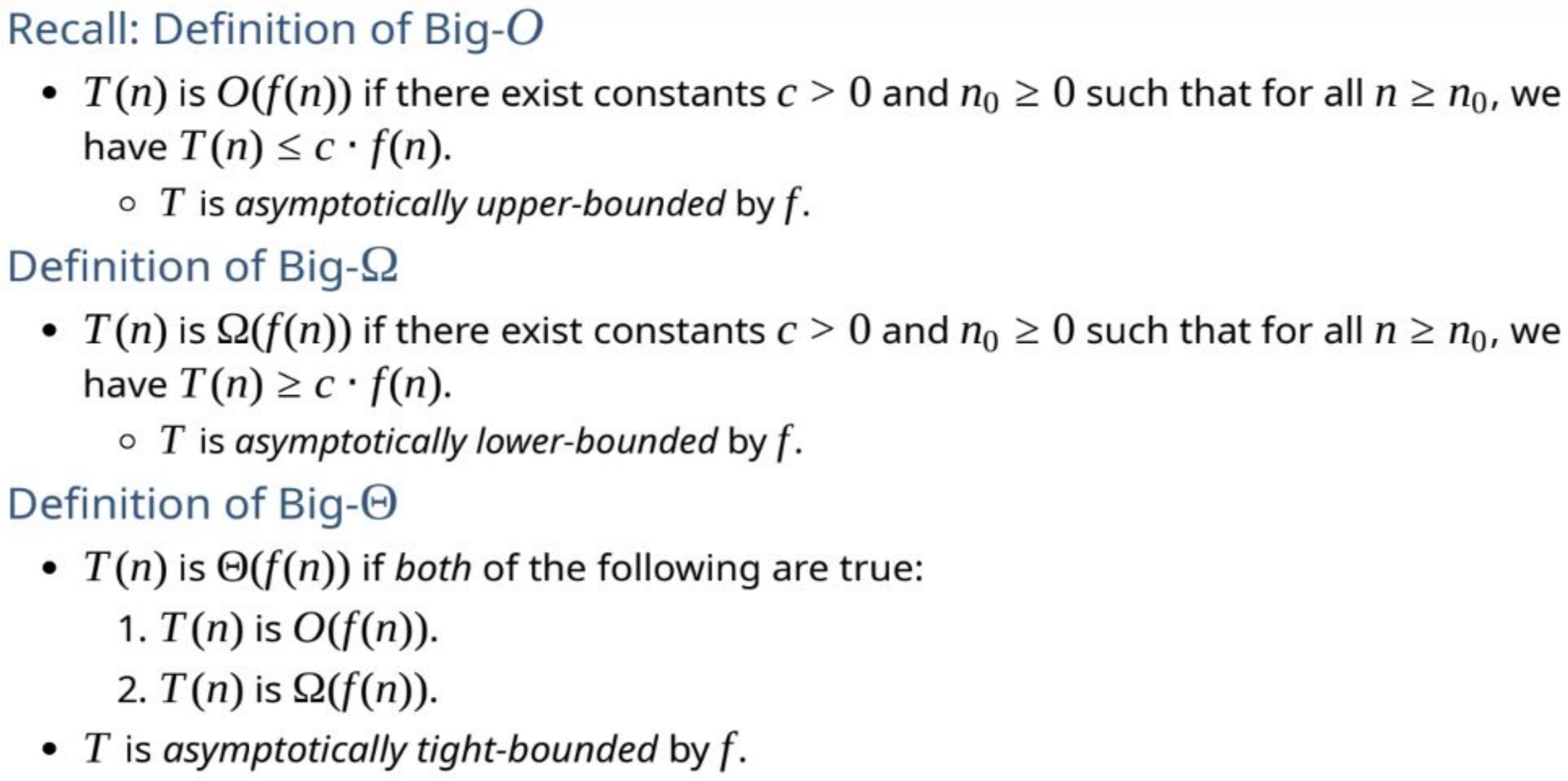
- Problem statment:

  Input: a sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$

  Output: a permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the a-sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

  In short, *sorting*
- Algorithms:
  1. Insertion sort
  2. Merge sort

Quick Review of Asymptotic Analysis (Kaloti 2020)



Recall: Definition of Big-O

- $T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $T(n) \leq c \cdot f(n)$.
  - T is *asymptotically upper-bounded* by f.

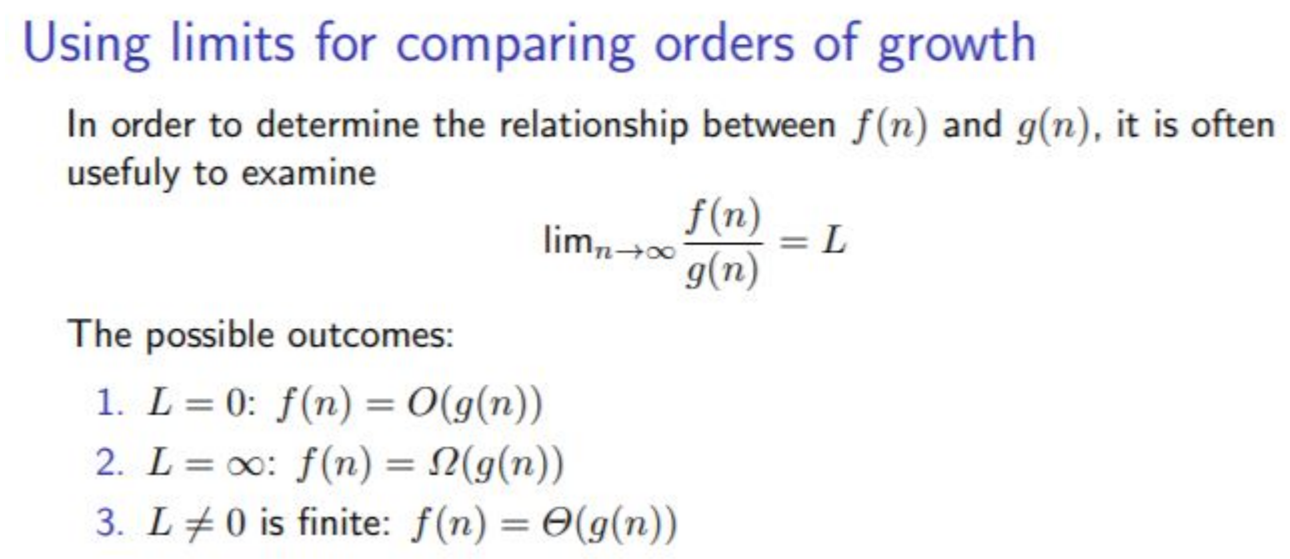
Definition of Big-Ω

- $T(n)$ is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $T(n) \geq c \cdot f(n)$.
  - T is *asymptotically lower-bounded* by f.

Definition of Big-Θ

- $T(n)$ is $\Theta(f(n))$ if *both* of the following are true:
  1. $T(n)$ is $O(f(n))$.
  2. $T(n)$ is $\Omega(f(n))$.
- T is *asymptotically tight-bounded* by f.

Limit Lemma Theorem



Using limits for comparing orders of growth

In order to determine the relationship between $f(n)$ and $g(n)$, it is often usefuly to examine

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

The possible outcomes:

1. $L = 0$: $f(n) = O(g(n))$
2. $L = \infty$: $f(n) = \Omega(g(n))$
3. $L \neq 0$ is finite: $f(n) = \Theta(g(n))$