Lecture 1

Revision: Exponents

$x^a \cdot x^b = x^{a+b}$

$\frac{x^a}{x^b} = x^{a-b}$

$(x^a)^b = x^{ab}$

$x^n + x^n = 2x^n$

$f = x^{2n}$

Example prove $(x^a)^b = x^{ab}$

$x^a \cdot x^1 = x^{a+1}$

$(x^a)^b = \underbrace{x^a + x^a \cdots x^a}_{b}$

$\therefore x^{ab}$

Logarithms:

$c^a = b \iff \log_c b = a$

$\log_c c = 1$

$c^{\log_c a} = a$

$\log_a b = \frac{\log_c b}{\log_c a}, c > 0$

$\log ab = \log a + \log b$

$\log \frac{a}{b} = \log a - \log b$

$\log(a^b) = b \log a$

$\log x < x, \forall x > 0$

- Note: $\iff$ means if and only if

Example:

Prove $\log_a b = \frac{\log_c a}{\log_c b}, c > 0$

$x = \log_a b \iff a^x = b$

$y = \log_c b \iff c^y = b$

$z = \log_c a \iff c^z = a$

$\Rightarrow (c^z)^x = b$

$= c^{xz}$

$= c^y$

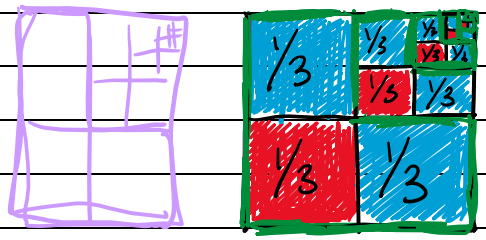
$y = xz$

$x = y/z$

$\therefore y = xz$

Geometric Series

$A = 1/3$ since Red is $1/3$ of each square as the final $1/4$ is once again broken up in $1/3$ and so on. Since the series is infinite it will always be the case of 3 squares left 4^{th} is broken up etc.



$A = \frac{1}{4} + \frac{1}{4} \times \frac{1}{4} + \frac{1}{4} \times \frac{1}{16} \cdots$

$= \frac{1}{4}\left(1 + \frac{1}{4} + \frac{1}{16} \cdots\right)$

$A = 1/3$

$R = 5$

S_4 $\quad 1 + 1(5) + 1(5^2) \cdots$

$$S_n = a + aR + aR^2 + aR^3 + \cdots aR^n = \frac{a(R^n - 1)}{R - 1}$$

cleaned up → $\frac{a(1 - R^n)}{1 - R}$ ie. $\frac{\text{"first"} - \text{"last}+1\text{"}}{1 - R}$

Example :

$$\sum_{i=0}^{n-1} 2^i = 1 + 2^1 + 2^2 + \cdots 2^{n-1} = 2^n - 1$$

If $0 < R < 1$ (when $a = 1$) then the n^{th} term sum is

Note :

$$S_\infty = \frac{a}{1 - R}$$

$$\sum_{i=0}^{n-1} R^i < \sum_{i=0}^{\infty} R^i = \frac{1 - R^\infty}{1 - R} = \frac{1}{1 - R}$$

Example : what is the following sum

$$S = aR^k + aR^{k+1} + aR^{k+2} + aR^{k+3} + \ldots + aR^{n-1}$$

Answer

$$S = \frac{\text{"first"} - \text{"last}+1\text{"}}{1 - R}$$

$$S = \frac{(aR^k) - (aR^{n-1} + 1)}{1 - R}$$ → $aR^{n-1} + 1$

$$= \frac{aR^k - aR^n}{1 - R}$$

[Clean up] $= \frac{aR^k(1 - R^n)}{1 - R}$

Example :

$$\sum_{i=0}^{\infty} \cdot \left(\frac{1}{2}\right)^i$$

Since R is ½ it's between 0 and 1 so we use $\frac{1}{1 - R} = \frac{1}{1 - \frac{1}{2}} = 2$

$$\sum_{i=0}^{n} n$$

for (int i = 0; i < n ; i++){

A common questions that shows up is

$$\sum_{i=0}^{\infty} \frac{i}{2^i}$$

(Solution to how to do these can be found in tutorial 01

Arithmetic Series

An example would be

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2+n}{2} \approx \frac{n^2}{2}$$

n^2 is much larger than n so in most cases we can just drop n

Similarly in this example we can drop n^2 and n as n^3 is much larger than them

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{2n^3+2n^2+n}{6}$$

$$= \frac{2n^3}{6} \approx \frac{n^3}{3}$$

Another example

$$\sum_{i=1}^{n} f(n) = n \times f(n)$$

Note: not $f(i)$

↳ This means that every term there is the same no matter what value of i you choose

So what really happens if for eg $i = 3$ your adding n 3 times i.e. $[3 \times f(3)]$ hence $f(n) = n \times f(n)$

Example if i doesn't start at 1

$$\sum_{i=c}^{n} f(i)$$ first we break it up into 2 sums

$$\sum_{i=1}^{n} f(i) - \sum_{i=1}^{c-1} f(i)$$

In the first case we go from 1 to n then we take away the uneeded numbers from 1 to c-1 in the second case

Lecture 2

Main proof techniques

- induction (most common)
- contradiction
- Counter-example
- contra-positive

Proof by Induction

- Suppose we have a theorem quantified by a variable n
  - ↳ proof by induction proves that the theorem is true for $n = k+1$ by using the fact that the theorem is true for some number k.
  - ↳ this only works for proving things about integers or other "discrete" objects
  - ↳ in the inductive step, we show that the theorem is true for $n = k+1$ assuming

that it is true for $n = k$.

↳ for this to work we also need a starting point or, base case.

- Example

$$\sum_{i=1}^{n} (2i-1) = n^2, \quad n > 0 \qquad \text{Why?}$$

$$\sum_{i=1}^{n} (2i-1) = \sum_{i=1}^{n} 2i + \sum_{i=1}^{n} (-1)$$

first we need to break up the sum into 2 parts

$$= 2\sum_{i=1}^{n} i + (-n)$$

then we clean it up

now we know that $\sum_{i=1}^{n} i$ looks like this as an equation: $\frac{n(n+1)}{2}$ so we sub it in

$$= 2\left(\frac{n(n+1)}{2}\right) - n$$

clean this up

$$= \cancel{2}\left(\frac{n(n+1)}{\cancel{2}}\right) - n$$

$$= n^2 + \cancel{n} - \cancel{n}$$

$$= n^2$$

So how does that prove it true?
We are told n must be bigger than 0, ∴ the base case must be 1 meaning if n = 1 the equation is going to be true

$$\sum_{i=1}^{1} (2i-1) = 1^2$$

$$(2(1)-1) = 1$$
$$2-1 = 1$$
$$1 = 1 \quad \checkmark \text{ true}$$

now we assume that n = k is true meaning that for any number subsituted for k (as long as it greater than 0) the equation will hold. Since n = k is true then n = k+1 must also be true and this is the part we need to prove. n = k+1

assume: $\sum_{i=1}^{k+1} (2i-1) = (k+1)^2$ [ie sub n for k+1]

Proof

$$\sum_{i=1}^{k+1} (2i-1) = \sum_{i=1}^{k} (2i-1) + (2(k+1)-1)$$

$(2i-1)$ where $i = k+1$

work it out

$$= k^2 + 2k + 2 - 1$$
$$= k^2 + 2k + 1$$
$$= (k+1)(k+1)$$
$$= \boxed{(k+1)^2}$$

Since our proof is the same as our assumption we know that it's correct and the proof is true.

Sometime we aren't given a base case and due to this we need to find it

Example

$5n - 1 < n^2$

Easiest way is trial and error by subbing in numbers for n

$n = 4$: $5(4) - 1 < 4^2$

$19 < 16$ ✗

$n = 5$: $5(5) - 1 < 5^2$

$24 < 25$ ✓

n=5 is true but we should at least the next 2 n's as for it to be a base case all numbers after a base case will result in true

$n = 6$: $5(6) - 1 < 6^2$

$29 < 36$ ✓

$n = 7$: $5(7) - 1 < 7^2$

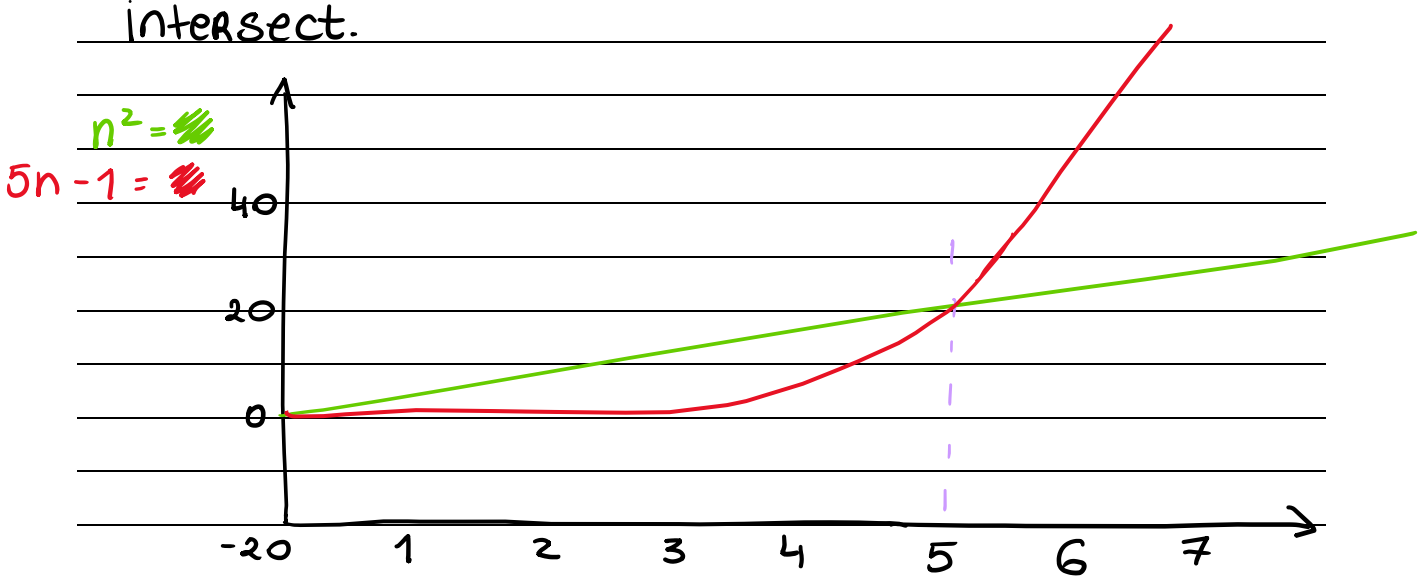
$34 < 42$ ✓

Now we are confident that $n = 5$ is the base case so we can say

$5n - 1 < n^2$ as long as $n \geq 5$

[• You can now do the proof for it]

Another way to find the base case is to plot the 2 equations on a graph and see where they intersect.



Proof by Contradiction

- Proof by contradiction relies on the fact that everything is OR isn't true.
- To prove some fact, we attempt to prove the opposite hypothesis instead.
  - ▶ simple example $2(n) = 2$ if and only if $n = 1$

$$2(1) = 2$$
$$2 = 2 \quad \checkmark$$

Contradiction: Let's say $n = 2$

$$2(2) = 2$$
$$4 \neq 2$$

this proves that $2(n) = 2$ is only true for $n = 1$

- By encountering a contradiction somewhere in this "proof" it shows that our new hypothesis must be false, meaning that the original was true
- So to prove theorem "$A \Rightarrow B$", we would attempt to prove the opposite, "$A \Rightarrow \neg B$"

- Example

There are an infinite number of prime numbers.

Proof: Suppose this is not true, Then we assume there are only a finite number of primes, say m of them. Then we can write these m numbers

$$p_1, p_2, p_3 \cdots p_m$$

where p_m is the largest prime possible.

Now consider the number

$$K = 1 + p_1 \cdot p_2 \cdot p_3 \cdot \cdots p_m$$

This number cannot have any divisors so it must be prime. But $K > p_m$, which contradicts our assumption.

Proof by Counterexample

- Can only use this to prove a statement false.
- Example

$$5n - 1 < n^2, \; n > 0$$

Proof:

When $n = 1$, $5 - 1 \not< 1$

↳ Theorem would have been true if condition was $n \geq 5$. In that case, prove by induction.

- Example

"The sun never shines in Limerick"

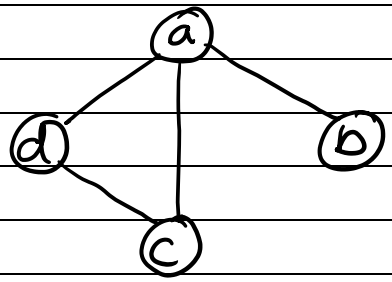
Proof:

Take a look outside: the sun is shining!

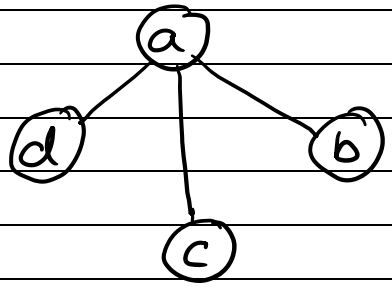
Proof by Contrapositive

- To prove "$A \Rightarrow B$", prove instead "$\neg B \Rightarrow \neg A$"
  ↳ Both are equivalent; technique often called Modus Tollens
- Example (we haven't done graph theory yet so no need to look at it)

"Prove that if every vertex in a graph $G = (V, E)$ has degree greater than 1, then there must be some cycle in the graph"



a cyclic graph

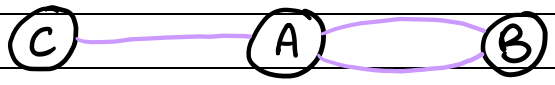


A tree

Note: what the theorem says → if every vertex has a degree of two, or more, then there must be a cycle, yet if some vertices have degree one, there could be still a cycle.

- Proof (Ver. 1)
  - ↳ If every vertex has degree 2 or more, then it has at least two edges.
  - ↳ Pick any vertex v as a starting point, "mark" it and leave the vertex on one of its edges.
  - ↳ This takes us to a new vertex and, since its degree is greater than 1, we can exit it on a different edge
  - ↳ Keep doing this, marking each vertex as we go, and never walking across an edge twice
  - ↳ Since there are a finite number of vertices, we must visit some vertex that we've seen before.

- The trouble with walking to vertices of degree 1



Proof (Ver. 2)

↳ If we abbreviate "there must be some cycle in the graph" by "$cyc(G)$" (G has a cycle) then we can write the theorem as

$$\forall v \in V,\ d(v) \geq 2 \Rightarrow cyc(G)$$

↳ The contrapositive of this is "if there is no cycle in G then there must be some vertex v whose degree is not 2 or more" ie:

$$\neg cyc(G) \Rightarrow \exists v \in V,\ \neg d(v) \geq 2$$
$$\neg cyc(G) \Rightarrow \exists v \in V,\ d(v) \leq 1$$

↳ Looking for a contradiction to prove the opposite

$$\neg cyc(G) \Rightarrow \forall v \in V,\ d(v) \geq 2$$

↳ This cannot be true because the previous tree clearly has no cycles yet the leaves have degree 1

$$\neg cyc(G) \Rightarrow \forall v \in V,\ d(v) \geq 2$$

meaning both

$$\neg cyc(G) \Rightarrow \exists v \in V,\ \neg d(v) \geq 2$$
$$\forall v \in V,\ d(v) \geq 2 \Rightarrow cyc(G)$$

are true.

Recursion Recap

- To solve a problem recursively we break the problem in to one or more subproblems that have the same description as the original
- Each of the subproblems can then be solved using the algorithm.
- An example of recursion would be mergesort.
  - To sort an array in two, sort the two halves (independently) and then merge the two haves into one.

Lecture 3

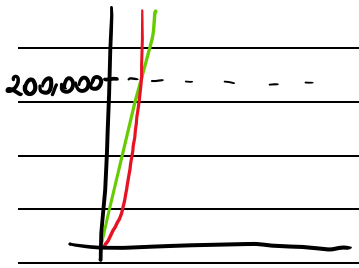
- Chief concern in algorithm analysis is determining the underlying Running time and space requirements in the long term when the input size grows larger
- Running-time is highly dependant on machine architecture, so analysis cannot be machine-specific. We need to do something that takes the variability out of it.
- To do this we introduce the order notation that describes the time and storage requirements only in terms of the algorithm and its parameters.

Google-sized statistics

- few desktop machines will execute more than a billion operations per second
- Google indexes over 40 billion pages (2015)
  35 trillion pages (2021)
  - The 40 billion suggests a desktop would take over 40s to execute a single search query; or 40 desktops to perform a search in 1s
  - Google recieves 40,000 search queries per second (2012) [70,000 (2019)]
  - So they would need $40{,}000 \times 40 = 1{,}600{,}000$ very fast servers just to maintain a 1s response time
  - How do they do it?
    - Their search algorithm is as fast and efficient as possible

Families of Functions

- Which function is bigger $f(n) = 20 \cdot n$ or $g(n) = 0.002 \cdot n^2$

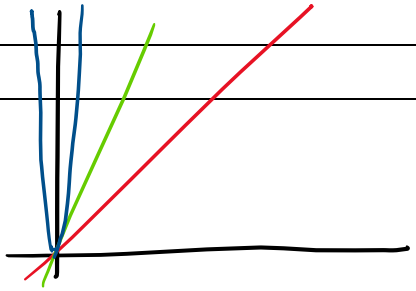


- Although $f(n)$ is larger initially, the n^2 of $g(n)$ dominates eventually
- Likewise $f(n) = a \cdot n^3$ will outgrow $g(n) = b \cdot n^2$ sooner or later.

The constants a and b only determine when

- When comparing functions you should ignore the multiplier and focus on the power
- With this method of comparison two functions with the same power will behave broadly similarly.

$g(n) = 2n$
$f(n) = 4n$
$h(n) = 2n^2$



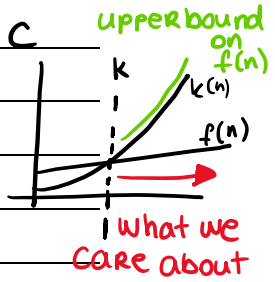
↳ To help us quantify functions we introduce four definitions

O(n) – Big-Oh n is the number of inputs (ie. rows in a db)

[Note: T denotes time]

$T(n) = O(f(n))$ if there are constants c and n_0 so that

$T(n) \leq cf(n)$ when $n \geq n_0$



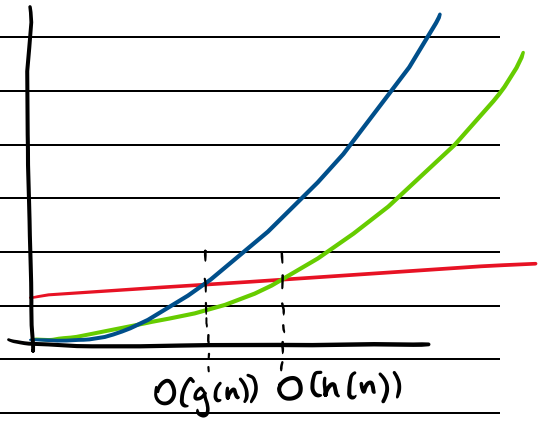
→ When we get to the interesting numbers out past n_0 (ie k), if we can argue that our operation is always $\leq$ some multiple of (fn) (ie c), that the functions is then $T(n) = O(f(n))$.

T(n) is "less than or equal to" f(n)

↳ That is once $n \geq n_0$, T(n) is always less than or equal to some constant, c, times f(n)

↳ $f(n) = n+5$
$g(n) = n^2$
$h(n) = 2n^2$

↳ $f(n) = O(g(n))$ and
$f(n) = O(h(n))$

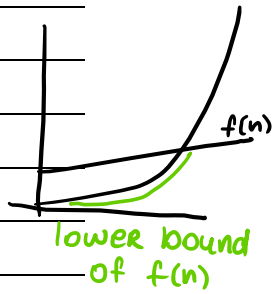


$\Omega(n)$ - Big-Omega

$T(n) = \Omega(g(n))$ if there are constants c and n_0 so that

$T(n) \geq c\,g(n)$ when $n \geq n_0$

- This is the opposite of the previous definition, where we are interested in the lower bound

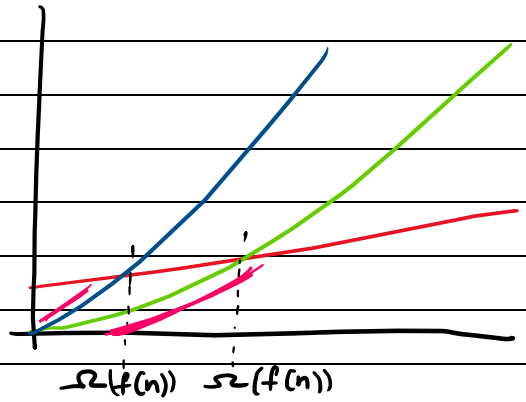


$T(n)$ is "greater than or equal to" $g(n)$

ie if $f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$

- $f(n) = n + 5$
- $g(n) = n^2$
- $h(n) = 2n^2$

- $g(n) = \Omega(f(n))$ and $h(n) = \Omega(f(n))$



$\Theta(n)$ - Big-Theta

$T(n) = \Theta(h(n))$ if and only if $T(n) = O(h(n))$ and $T(n) = \Omega(h(n))$

$T(n)$ is "behaves like" $h(n)$

ie $\lim_{n \to \infty} \frac{T(n)}{h(n)} \to c$

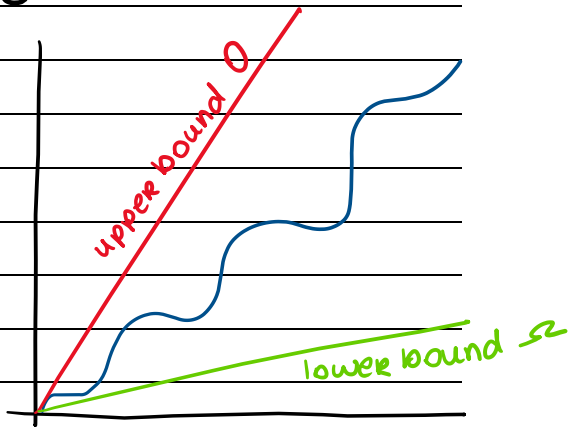
- $h_1(n) = 6n$
- $h_2(n) = 0.5n$
- $f(n) = \sin$

$f(n)$ is stuck between 2 different functions of the same multiple

$h_1(n) = k_1 f(n)$ $k_1 = 6$

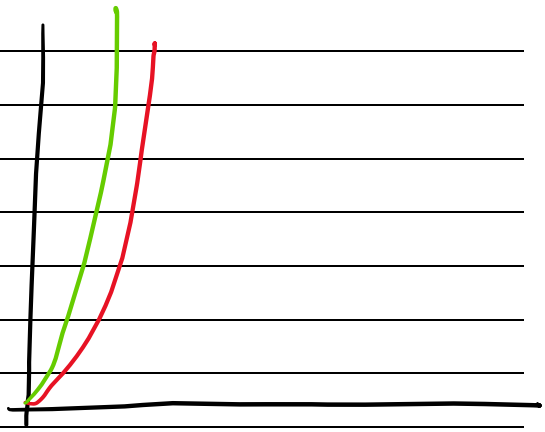
$h_2(n) = k_2 f(n)$ $k_2 = 0.5$

$f(n) = n$



$f(n) = 2n^2$
$g(n) = 6n^2$ then

$f(n) = O(g(n))$
$f(n) \neq o(g(n))$



o(n) - Little-oh

$T(n) = o(p(n))$ if $T(n) = O(p(n))$ and
$T(n) f = \Theta(p(n))$

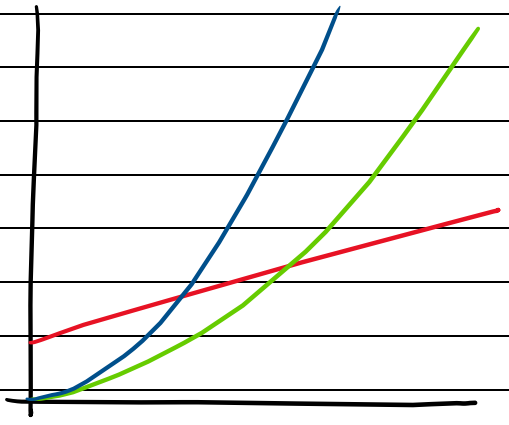
T(n) is " is smaller than " p(n);

ie $\lim_{n \to \infty} \frac{T(n)}{p(n)} \to 0$

↳ $T(n) = O(p(n))$ allows for T(n) be having possibly broadly similar to p(n); whereas $T(n) = o(p(n))$ insists on p(n) dominating T(n) - with the gab widening all the time

↳ $f(n) = n + 5$
↳ $g(n) = n^2$
↳ $h(n) = 2n^2$

↳ $g(n) = \Theta(h(n))$ and
$\therefore h(n) = \Theta(g(n))$
↳ $f(n) = o(g(n))$



→ He skips the next 2 slides

Lecture 4

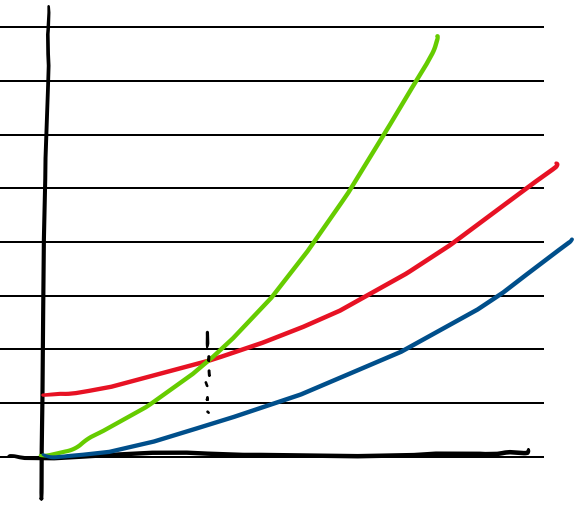
Growth Rates of Some Commonly Occurring Functions

Function	Name	Example
c	Constant	Adding two numbers
$\log \log n$	"Log log"	Sieve of Eratosthenes (Q2.21)
$\log n$	Logarithmic	Searching a balanced binary tree
$\log^2 n$	Log squared	
n	Linear	Searching a list
$n \log n$	"n log n"	merge sort, quicksort
n^2	Quadratic	Insertion / Selection / Bubblesort
n^3	Cubic	Multiplying 2 $n \times n$ matrices
$2^n, a^n, n!$	Exponential	No. of ways to permute n objects ($n!$) No. of unique patterns of n bits (2^n)

→ He goes back to explain the 2 slides from lecture 3

- $g(n) = \Theta(n^2)$
  - $g(n) = n^2 + 5$
  - $h(n) = n^2$
  - $2 \cdot h(n)$
  - $1/2 \cdot h(n)$

  - so we can say that $g(n) = \Theta(n^2)$



Some Rules

→ Some rules for combining "Big-Ohs"

Rule 1 [learn these rules]

If $T_1(n) = O(f(n))$ and $T_2(n) = O(g(n))$, then

(a) $T_1(n) + T_2(n) = \max(O(f(n)), O(g(n)))$;

(b) $T_1(n) * T_2(n) = O(f(n) * g(n))$

(a) The overall Running of the Time is the larger of the 2

(b) most frequently used with nested loops ie if you have a piece of code T_1 that calls a second piece of code T_2 multiple times so the overall effect is the product of the 2

Rule 2

If $T(n)$ is a polynomial of degree k, then

$T(n) = \Theta(n^k)$

[Note: ' means prime]

$T(n) = 5n^2 + 20n \rightarrow T(400) = 808,000$]

$T'(n) = 5n^2 \rightarrow T'(400) = 800,000$]

$T(400)/T'(400) = 808,000/800,000 = 1.01$

$T(4000)/T'(4000) = 8,008,000/8,000,000 = 1.001$]

etc

$\therefore T(n) \approx 5n^2$

Example of 20n is nearly irrelevant in this equation

20n? vs n^2

n=1 20 vs 1 ;

n=100 2000 vs 10,000

Rule 3

$\log^k n = O(n)$ for any constant k.

Logarithms grow very slowly