

2: GROWTH OF FUNCTIONS

Asymptotix : Asymptotic *relations* for understanding the relative GROWTH for functions.

NOTATION

There is a variety of notations, differing in details.

- $f(x) \sim g(x)$ as $x \rightarrow \infty \Leftrightarrow$

$$\lim_{x \rightarrow \infty} f(x)/g(x) = 1$$

- $f(x) = o(g(x))$ as $x \rightarrow \infty \Leftrightarrow$

$$\lim_{x \rightarrow \infty} f(x)/g(x) = 0$$

Where f and g are positive COST functions
and (c, x, n) are positive values

Slide Lecture 2 -1

BIG OMICRON

- $f(x) = \mathcal{O}(g(x)) \Leftrightarrow \exists c, n_0: f(x) \leq c g(x) \forall x > n_0$
(or $\lim_{x \rightarrow \infty} f(x)/g(x) = c$)

\mathcal{O} - notation used in *Upperbounds*, i.e. complexity of a 'best' algorithm solving a certain problem.

BIG OMEGA

- $f(x) = \Omega(g(x)) \Leftrightarrow \exists c, n_0: f(x) \geq c g(x) \forall x > n_0$

Ω - notation used in *Lowerbound*, i.e. a certain problem has AT LEAST certain complexity.

Slide Lecture 2 -2

LITTLE-o

$o(g(n)) = \{f(n) : \text{For any positive constant } c > 0 \text{ there exist } n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0 \}$

$$\lim_{x \rightarrow \infty} f(x)/g(x) = 0$$

Therefore, $2n = o(n^2)$, but $2n^2 \neq o(n^2)$
since it does not hold **forall** constants.

LITTLE- ω

$\omega(g(n)) = \{f(n) : \text{For any positive constant } c > 0 \text{ there exist } n_0 > 0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0 \}$

$$\lim_{x \rightarrow \infty} f(x)/g(x) = \infty$$

Slide Lecture 2 -3

Asymptotic Notation : $2n^2 + 3n + 1 = 2n^2 + \mathcal{O}(n)$
 $= \mathcal{O}(n^2)$

$2n^2 + \mathcal{O}(n) = \mathcal{O}(n^2)$

$\mathcal{O}(n)$ is used here as an "ANONYMOUS" function that makes
this assertion true

Note:

$$f(n) = g(n) + \mathcal{O}(n)$$

also means

$$f(n) - g(n) = \mathcal{O}(n)$$

Slide Lecture 2 -4

BIG THETA

• $f(x) = \Theta(g(x)) \Leftrightarrow f(x) = \mathcal{O}(g(x)) \wedge f(x) = \Omega(g(x))$

Θ - Big Theta : *tight bound*

ALSO

A notation used for algorithms with complexity $\mathcal{O}(g(x))$ solving problems with lowerbounds $\Omega(g(x))$

Slide Lecture 2 -5

→ A **CLOSED PROBLEM**

has an asymptotically optimal algorithm.

e.g. searching in an ordered list

has lowerbound $\Omega(\log(n))$

Upperbound $\mathcal{O}(\log(n))$ (e.g. binary search)

(Note the confusing Overloaded meaning of Lowerbound—on the problem and on the algorithm.)

Slide Lecture 2 -6

Some more notation:

$$(1) f(x) = g(x) + o(h(x))$$

careful: "=" suggest identity but here it *isn't*

$$(1) \text{ means } f(x) - g(x) = o(h(x))$$

IN INEQUALITIES:

$$(2) x + o(x) < 2x \quad \text{as } x \rightarrow \infty$$

$$(2) \text{ means } \forall (f(x) = o(x)) \exists x_0 : x + f(x) < 2x \quad \forall x > x_0$$

Slide Lecture 2 -7

So when asymptotic notation is used, the two sides of a relation ($=, <$) DO NOT PLAY A SYMMETRIC ROLE!

The right-hand-side contains less information and cannot replace the left-hand-side in every context.

$$O(x) = f(x) \Leftarrow \text{MEANINGLESS}$$
$$x = O(x^2) \text{ and } x^2 = O(x^2) \text{ does not imply } x = x^2$$

Slide Lecture 2 -8

No difficulties if \mathcal{O} , o on r.h.s. of a relation, but as in examples above it is sometimes convenient to use on l.h.s.

ex:

$$\frac{x}{x \leftrightarrow 1} = 1 + \frac{1}{x} + \mathcal{O}\left(\frac{1}{x^2}\right) = 1 + \frac{1}{x} + o\left(\frac{1}{x}\right) = 1 + o(1)$$

- notice that we lose information going from left to right.
- Θ - notation implies tight upper and lower bound

Slide Lecture 2 -9

\mathcal{O} - notation implies (tight) upper bound.

(the tight requirement is not formal)

$$\mathcal{O}(g(n)) \Rightarrow f(n) : \exists c, n_0, \quad 0 \leq f(n) \leq cg(n), \quad \forall n \geq n_0$$

$$\text{Note } f(n) = \Theta(g(n)) \Rightarrow f(n) = \mathcal{O}(g(n))$$

since Θ is stronger.

However $\mathcal{O}(f(n))$ implies $an+b$ is in $\mathcal{O}(n^2)$

Bounded but not tight.

Usually use "little o" here.

Big-O is used informally for Θ

Intuitively we throw away lower order terms

Slide Lecture 2 -10

E.g. $1/2n^2 - 3n = O(n^2)$

Formally, for Θ we need constants c_1, c_2, n_0 such that,

$$c_1 n^2 \leq 1/2 n^2 \Leftrightarrow 3n \leq c_2 n^2$$

for all $n \geq n_0$

Dividing by n^2

$$c_1 \leq 1/2 \Leftrightarrow 3/n \leq c_2$$

For all $n \geq 7$ we know $3/n < 1/2$

and $c_1 \leq \frac{1}{14}$ $c_2 \geq 1/2$ when $n_0 = 7$

Slide Lecture 2 -11

Note

$$c_1 \leq \frac{7}{14} \Leftrightarrow \frac{6}{14} \leq c_2$$

$$\frac{1}{14} \leq \frac{7}{14} \Leftrightarrow \frac{6}{14} \leq \frac{1}{2}$$

Also note:

$(1/2)n^2 \Leftrightarrow 3n = 0$, $(1/2)n^2 = 3n$, $1/2n = 3$, $n = 6$
so growth is positive at 7 and above.

Slide Lecture 2 -12

Consider any quadratic function :

$$f(n) = an^2 + bn + c$$

where a, b and c are constants, $a > 0$

Let $c_1 = a/4, c_2 = 7a/4$

$$n_0 = 2\{MAX(|b|/a, \sqrt{|c|/a})\}$$

For c_2 , worst case is when b & c positive

$$\frac{a}{4}n^2 \leq an^2 + bn + c \leq \frac{7}{4}an^2$$

Divide by n^2

$$a\frac{1}{4} \leq a + \frac{b}{n} + \frac{c}{n^2} \leq \frac{7}{4}a$$

Slide Lecture 2 -13

Divide by a

$$\frac{1}{4} \leq 1 + \frac{b}{na} + \frac{c}{an^2} \leq \frac{7}{4}$$

CHOICE of $n_0 = 2\{MAX(|b|/a, \sqrt{|c|/a})\}$ makes this approximately

$$1 + 1/2 + 1/4 \leq 7/4 \quad 1/4 \leq 1 \Leftrightarrow 1/2 \Leftrightarrow 1/4$$

Slide Lecture 2 -14

GROWTH RATES

Some Common Bounds

(1) $f(n) = O(1)$ **CONSTANT COMPLEXITY**

Complexity of the program is constant,

i.e. independent of the input size.

e.g. scalar operations, when input size not measured in # bits.

('a+b')

Slide Lecture 2 -15

(2) $f(n) = O(\log n)$ **LOGARITHMIC COMPLEXITY**

Some characteristics of logarithms

(a) 'Def': $b^{\log_b a} = a$ $\log_b b = 1$

(b) $\log(x_1 * x_2) = \log x_1 + \log x_2$

Note: $b^{\log_b(x_1 x_2)} = (b^{\log_b x_1})(b^{\log_b x_2})$

Then: $(b^{\log_b x_1})(b^{\log_b x_2}) = b^{\log_b x_1 + \log_b x_2}$ since $a^x a^y = a^{(x+y)}$

(c) $\log(x_1/x_2) = \log x_1 \Leftrightarrow \log x_2$

(d) $\log x^a = a \log x$

$$\text{e.g. } \log 8^2 = 2(\log 8)$$

$$\log 64 = 2 * 3 = 6$$

Slide Lecture 2 -16

$$(e) \log_a x = \frac{\log_b x}{\log_b a} \Rightarrow \log_a x = \frac{1}{\log_b a} \log_b x$$

Proof:

$$b^{\log_b x} = x = a^{\log_a x} = (b^{\log_b a})^{\log_a x} = b^{(\log_b a)(\log_a x)}$$

$$b^{\log_b x} = b^{(\log_b a)(\log_a x)}$$

$$\log_b x = (\log_b a)(\log_a x)$$

$$\log_a x = \frac{1}{\log_b a} (\log_b x)$$

Note $\frac{1}{\log_b a}$ is a constant.

Slide Lecture 2 -17

$$(f) x^{\log_b y} = y^{\log_b x}$$

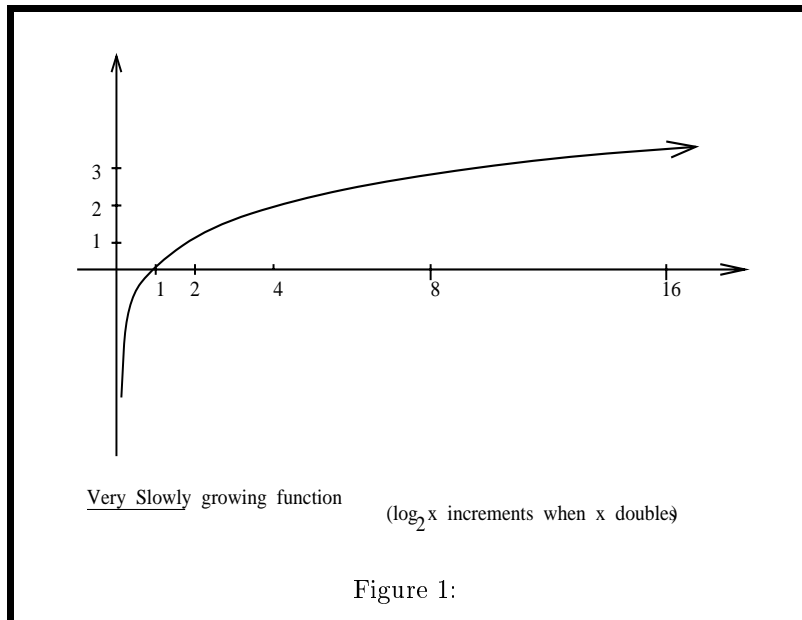
$$\text{Proof: } x^{\log_b y} = (y^{\log_b x})^{\log_b y} = y^{\frac{\log_b x}{\log_b y} \log_b y} = y^{\log_b x}$$

(g) $\log x$ is a 1 to 1 function, strictly increasing

$$\log y = \log z \quad IFF \quad y=z$$

$$(\log_b 1 = 0 \quad \forall b)$$

Slide Lecture 2 -18



Slide Lecture 2 –19

CHANGING THE BASE OF A LOGARITHM PRODUCED A CONSTANT RATE OF CHANGE IN GROWTH FACTOR SO WE USE " $\log_2 n$ " or " $\lg n$ " in \mathcal{O} -notation.

Algorithms with $\mathcal{O}(\log n)$ complexity, often have the following form :

Solve P:

- (1) Do some $\mathcal{O}(1)$ work to test trivial case.
- (2) If not trivial do some $\mathcal{O}(1)$ work to split the problem in d equal parts, $p_1 \dots p_d$ and select a part p_i
- (3) Recursively or iteratively, solve the one problem p_i .

Slide Lecture 2 –20

This is a simple form of the well-known Divide & Conquer technique :

e.g.

Divco(P) \rightarrow R

if trivial(P) then Solve_direct(P) \rightarrow R

else *split* P \rightarrow $P_1 \dots P_n$

$\forall P_i$ Divco(P_i) $\rightarrow r_i$

combine: $r_1 \dots r_n \rightarrow$ R

Slide Lecture 2 -21

$f(n) = O(n)$

e.g. linear search has $O(n)$ (worst case) complexity (*assume unsorted list*)

other example : **Polynomial evaluation**

$A(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 (a_n \neq 0)$

polynomial of *degree* n

Problem : given a polynomial A(x) and a value for x: x_0

evaluate **A**(x_0)

Slide Lecture 2 -22

HOW NOT TO DO IT :

$a_n * \text{power}(x_0, n) + a_{n-1} * \text{power}(x_0, n-1) + \dots$

because this involves : n *-s, n+-s, n powers.

But **USE HORNER'S RULE** (A.K.A. Horne's Scheme)

$$A(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 \quad (1)$$

$$= (a_n x^{n-1} + a_{n-1} x^{n-2} + \dots + a_1) x + a_0 \quad (2)$$

$$\dots \quad (3)$$

$$= (\dots (a_n x + a_{n-1}) x + a_{n-2}) x + \dots + a_1) x + a_0 \quad (4)$$

$$(5)$$

Slide Lecture 2 -23

CODE

```
y := a[n]
```

```
for i := n-1 down to 0 do  
  y := y * x + a[i]
```

Slide Lecture 2 -24

e.g. bit string evaluation

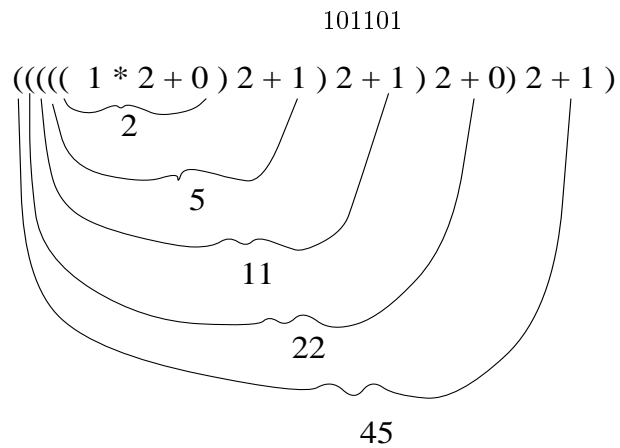


Figure 2:

Slide Lecture 2 -25

$$2^5 + 2^3 + 2^2 + 2^0 =$$
$$32 + 8 + 4 + 1 = 45$$
$$(2 * 2 * 2 * 2 * 2) + (2 * 2 * 2) + (2 * 2) + 1$$

Slide Lecture 2 -26

Horner's Complexity : $\mathcal{O}(n)$
Lowerbound Polynomial Evaluation : $\Omega(n)$
because we have to access each a_i at least once
So: Polynomial Evaluation is *Closed*

Slide Lecture 2 -27

But ... what is $A(x) = x^n$
then horner is **not optimal**
 $x^{2n} = x^n \cdot x^n$
 $x^{2n+1} = x^{2n} \cdot x$ Also suggests a recursive solution
Power(x,n)
{
 y:=1
 while n<>0 {
 if n odd then y:=y*x;
 x:=x*x;
 n:= $\lfloor n/2 \rfloor$
 }
 return(y)
}

Slide Lecture 2 -28

e.g. $n=11 \rightarrow$

x	y	n	$y.x^n$
x	1	11	x^{11}
x^2	x	5	$x.(x^2)^5$
x^4	x^3	2	$x^3.(x^4)^2$
x^8	x^3	1	$x^3.x^8$
x^{16}	x^{11}	0	$x^{11}.(x^{16})^0$

Slide Lecture 2 -29

Complexity Power : $\mathcal{O}(\log(n))$ - *cuts problem in half*

Power examines n 's bits from right to left.

Q: Can you write a program Powerlr that examines n 's bits from left to right ?

$$f(n) = \mathcal{O}(n \log n)$$

Results e.g. from *Divide & Conquer* algorithms, where

- split & combine take *linear* time
- sub-programs have equal size

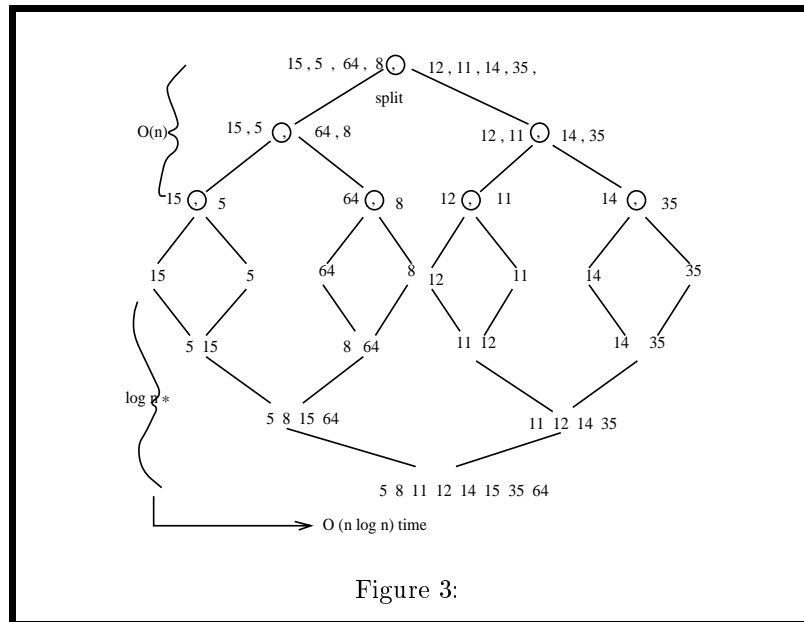
Slide Lecture 2 -30

Example :
Merge Sort

```

MERGESORT(Array) : \
  if (size(Array) = 1) then return(Array)
  else MERGE(MERGESORT(LEFTSPLIT(A))
             MERGESORT(RIGHTSPLIT(A)))
  
```

Slide Lecture 2 -31



Slide Lecture 2 -32

Linearity:

$$\sum_{k=1}^n (C a_k + b_k) = C \sum_{k=1}^n a_k + \sum_{k=1}^n b_k$$

Similarly:

$$\sum_{k=1}^n \mathcal{O}(f(k)) = \mathcal{O}\left(\sum_{k=1}^n f(k)\right)$$

Slide Lecture 2 -33

Arithmetic Series:

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

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

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

$$\sum_{i=1}^n i = \mathcal{O}(n^2)$$

Slide Lecture 2 -34

Arithmetic Series:

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

And in General:

$$\sum_{i=1}^n i^k = \frac{1}{k+1}n^{k+1} + \mathcal{O}(n^k)$$

Slide Lecture 2 -35

Geometric Series:

$$\sum_{i=0}^n x^i = 1 + x + x^2 + x^3 + \dots + x^n = \mathcal{O}(x^n)$$

If x is real and $x \neq 1$ then:

$$\sum_{i=0}^n x^i = \frac{x^{n+1} \Leftrightarrow 1}{x \Leftrightarrow 1} = \mathcal{O}(x^n)$$

If the summation is infinite and $|x| < 1$ then:

$$\sum_{i=0}^{\infty} x^i = \frac{x^{\infty} \Leftrightarrow 1}{x \Leftrightarrow 1} = \frac{1}{1 \Leftrightarrow x}$$

If the summation is infinite and $|x| < 1$ then:

$$\sum_{i=0}^{\infty} i x^i = \frac{x}{(1 \Leftrightarrow x)^2}$$

Slide Lecture 2 -36

Also:

$$\sum_{i=0}^n a_i x^i = \mathcal{O}(x^n) \text{ if } a_n \neq 0$$

Slide Lecture 2 -37

Harmonic Series and the **harmonic number**, H_n :

$$H_n = 1 + 1/2 + 1/3 + 1/4 + \dots + 1/n$$

$$H_n = \sum_{i=1}^n 1/i = \ln n + O(1)$$

Slide Lecture 2 -38

Products:

$$\prod_{i=1}^N a_i = a_1 a_2 a_3 \dots a_n$$

$$\lg(\prod_{i=1}^N a_i) = \sum_{i=1}^N \lg a_i$$

(Note $\lg(x \cdot y) = \lg x + \lg y$)

Slide Lecture 2 -39

Using INTEGRALS to determine bounds on SUMS.

If $f(x)$ is a monotonically increasing continuous function then

$$\int_a^b f(x) dx$$

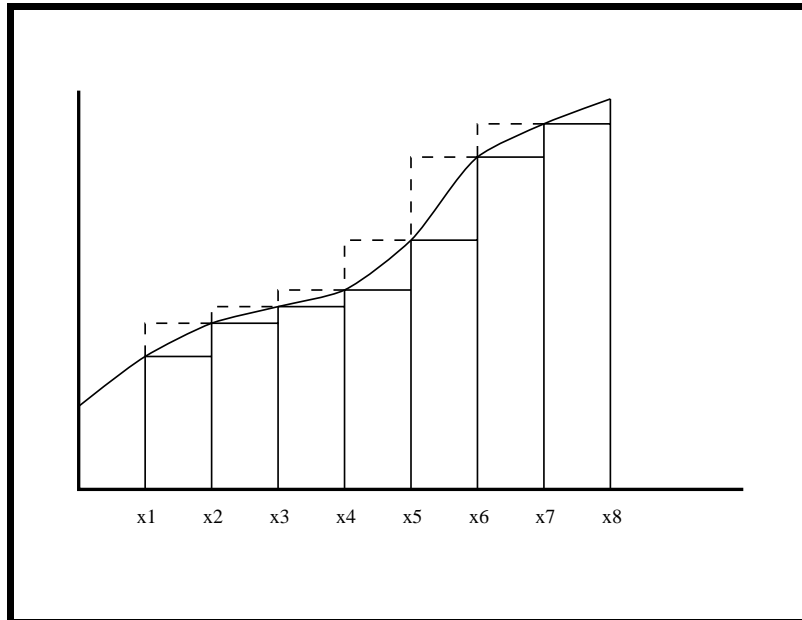
is bounded by lower and upper sums.

Consider:

$$x_i = i$$

$$\sum_{i=1}^n f(i) \leq \int_1^{n+1} f(x) dx \leq \sum_{i=1}^n f(i+1)$$

Slide Lecture 2 -40



Slide Lecture 2 -41

Note the following

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

Thus

$$\sum_{i=1}^n f(i) + f(n+1) \Leftrightarrow f(1) \geq \int_1^{n+1} f(x) \delta x$$

And

$$\sum_{i=1}^n f(i) \geq \int_1^{n+1} f(x) \delta x \Leftrightarrow f(n+1) + f(1)$$

Slide Lecture 2 -42

And

$$\int_1^{n+1} f(x) \delta x \Leftrightarrow f(n+1) + f(1) \leq \sum_{i=1}^n f(i) \leq \dots$$
$$\dots \int_1^{n+1} f(x) \delta x \leq \sum_{i=1}^n f(i+1)$$

Slide Lecture 2 -43

An Example:

Bounds for $\sum_{k=1}^n \lg k$

$$\int_1^{n+1} \lg x \delta x = x \lg x \Leftrightarrow x \Big|_1^{n+1}$$
$$= ((n+1) \lg(n+1) \Leftrightarrow (n+1)) \Leftrightarrow (1 \lg 1 \Leftrightarrow 1)$$
$$= ((n+1) \lg(n+1) \Leftrightarrow (n+1)) \Leftrightarrow (0 \Leftrightarrow 1)$$
$$= (n+1) \lg(n+1) \Leftrightarrow n \Leftrightarrow 1 + 1$$
$$= (n+1) \lg(n+1) \Leftrightarrow n$$

Slide Lecture 2 -44

So:

$$\int_1^{n+1} \lg x \, \delta x \Leftrightarrow f(n+1) + f(1) \leq \sum_{i=1}^n f(i) \leq \int_1^{n+1} \lg x \, \delta x$$

Implies:

$$(n+1) \lg(n+1) \Leftrightarrow n \Leftrightarrow \lg(n+1) + \lg(1) \leq \sum_{i=1}^n \lg i \leq (n+1) \lg(n+1) \Leftrightarrow n$$

So:

$$\sum_{i=1}^n \lg i = (n+1) \lg(n+1) \Leftrightarrow n \Leftrightarrow O(\lg n)$$

$$\sum_{i=1}^n \lg i = O(n \lg n)$$

Slide Lecture 2 -45

An Application:

$$\ln N! = \ln \prod_{i=1}^N i$$

$$\ln N! = \sum_{i=1}^N \ln i$$

(Note $\lg(x \cdot y) = \lg x + \lg y$)

$$\ln N! = O(N \lg N)$$

How can we use this to show that SORTING
is at least $O(N \lg N)$ in complexity?

Slide Lecture 2 -46

Stirling's Approximation:

$$N! \approx \sqrt{2\pi N} N^N e^{-N}$$

Or in Cormen:

$$N! = \sqrt{2\pi N} N^N e^{-N} \left(1 + \mathcal{O}\left(\frac{1}{N}\right)\right)$$

which also suggests:

$$\ln N! = O(\ln N^N) = O(N \lg N)$$