Analysis of algorithms

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming and Data Structures with Python

Lecture 14, 10 Oct 2023

A real world problem

- Every SIM card needs to be linked to an Aadhaar card
- Validate Aadhaar number for each SIM card

A real world problem

- Every SIM card needs to be linked to an Aadhaar card
- Validate Aadhaar number for each SIM card
- Simple nested loop

```
for each SIM card S:
  for each Aadhaar number A:
     check if Aadhaar number in S
     matches A
```

A real world problem

- Every SIM card needs to be linked to an Aadhaar card
- Validate Aadhaar number for each SIM card
- Simple nested loop
- How long will this take?
  - M SIM cards, N Aadhaar cards
  - Nested loops iterate $M \cdot N$ times

```
for each SIM card S:
  for each Aadhaar number A:
     check if Aadhaar number in S
     matches A
```

M

N

N ≈ pop of India > 100cr

$> 10^9$

M ≈ pop of India

A real world problem

- Every SIM card needs to be linked to an Aadhaar card
- Validate Aadhaar number for each SIM card
- Simple nested loop
- How long will this take?
  - M SIM cards, N Aadhaar cards
  - Nested loops iterate $M \cdot N$ times
- What are M and N
  - Almost everyone in India has an Aadhaar card: $N > 10^9$
  - Number of SIM cards registered is similar: $M > 10^9$

```
for each SIM card S:
  for each Aadhaar number A:
     check if Aadhaar number in S
     matches A
```

$M \cdot N = 10^9 \times 10^9$

A real world problem

- Assume $M = N = 10^9$
- Nested loops execute 10^{18} times

```
for each SIM card S:
  for each Aadhaar number A:
     check if Aadhaar number in S
     matches A
```

A real world problem

- Assume $M = N = 10^9$
- Nested loops execute 10^{18} times
- We calculated that Python can perform 10^7 operations in a second
- This will take at least 10^{11} seconds

```
for each SIM card S:
  for each Aadhaar number A:
     check if Aadhaar number in S
     matches A
```

A real world problem

- Assume $M = N = 10^9$
- Nested loops execute 10^{18} times
- We calculated that Python can perform 10^7 operations in a second
- This will take at least 10^{11} seconds
  - $10^{11}/60 \approx 1.67 \times 10^9$ minutes

```
for each SIM card S:
  for each Aadhaar number A:
     check if Aadhaar number in S
     matches A
```

A real world problem

- Assume $M = N = 10^9$
- Nested loops execute 10^{18} times
- We calculated that Python can perform 10^7 operations in a second
- This will take at least 10^{11} seconds
  - $10^{11}/60 \approx 1.67 \times 10^9$ minutes
  - $(1.67 \times 10^9)/60 \approx 2.8 \times 10^7$ hours

```
for each SIM card S:
  for each Aadhaar number A:
     check if Aadhaar number in S
     matches A
```

A real world problem

- Assume $M = N = 10^9$
- Nested loops execute 10^{18} times
- We calculated that Python can perform 10^7 operations in a second
- This will take at least 10^{11} seconds
  - $10^{11}/60 \approx 1.67 \times 10^9$ minutes
  - $(1.67 \times 10^9)/60 \approx 2.8 \times 10^7$ hours
  - $(2.8 \times 10^7)/24 \approx 1.17 \times 10^6$ days

```
for each SIM card S:
  for each Aadhaar number A:
    check if Aadhaar number in S
    matches A
```

A real world problem

- Assume $M = N = 10^9$
- Nested loops execute 10^{18} times
- We calculated that Python can perform 10^7 operations in a second
- This will take at least 10^{11} seconds
  - $10^{11}/60 \approx 1.67 \times 10^9$ minutes
  - $(1.67 \times 10^9)/60 \approx 2.8 \times 10^7$ hours
  - $(2.8 \times 10^7)/24 \approx 1.17 \times 10^6$ days
  - $(1.17 \times 10^6)/365 \approx 3200$ years!

```
for each SIM card S:
  for each Aadhaar number A:
     check if Aadhaar number in S
     matches A
```

A real world problem

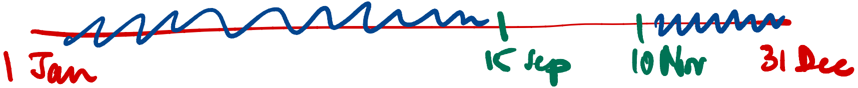
- Assume $M = N = 10^9$
- Nested loops execute 10^{18} times
- We calculated that Python can perform 10^7 operations in a second
- This will take at least 10^{11} seconds
  - $10^{11}/60 \approx 1.67 \times 10^9$ minutes
  - $(1.67 \times 10^9)/60 \approx 2.8 \times 10^7$ hours
  - $(2.8 \times 10^7)/24 \approx 1.17 \times 10^6$ days
  - $(1.17 \times 10^6)/365 \approx 3200$ years!
- How can we fix this?

```
for each SIM card S:
  for each Aadhaar number A:
     check if Aadhaar number in S
     matches A
```

Guess my daughter's birthday

- You propose a date (day and month)
- I answer, *Yes*, *Earlier*, *Later*

15 Sep Later
10 Nov Earlier
15 Oct Earlier
10 Oct Earlier
30 Sep Later
3 Oct later
7 Oct ✓



Guess my daughter's birthday

- You propose a date (day and month)
- I answer, *Yes*, *Earlier*, *Later*
- Suppose my birthday is 12 April

Guess my daughter's birthday

- You propose a date (day and month)
- I answer, *Yes*, *Earlier*, *Later*
- Suppose my birthday is 12 April
- A possible sequence of questions
  - September 12? Earlier
  - February 23? Later
  - July 2? Earlier
  - ...

Guess my daughter's birthday

- You propose a date (day and month)
- I answer, *Yes*, *Earlier*, *Later*
- Suppose my birthday is 12 April
- A possible sequence of questions
  - September 12? Earlier
  - February 23? Later
  - July 2? Earlier
  - ...
- What is the best strategy?

Guess my daughter's birthday

- You propose a date (day and month)
- I answer, *Yes*, *Earlier*, *Later*
- Suppose my birthday is 12 April
- A possible sequence of questions
  - September 12? Earlier
  - February 23? Later
  - July 2? Earlier
  - . . .
- What is the best strategy?

- Interval of possibilities

Guess my daughter's birthday

- You propose a date (day and month)
- I answer, *Yes*, *Earlier*, *Later*
- Suppose my birthday is 12 April
- A possible sequence of questions
  - September 12? Earlier
  - February 23? Later
  - July 2? Earlier
  - ...
- What is the best strategy?

- Interval of possibilities
- Query midpoint — halves the interval
  - June 30? Earlier
  - March 31? Later
  - May 15? Earlier
  - April 22? Earlier
  - April 11? Later
  - April 16? Earlier
  - April 13? Earlier
  - April 12? Yes

Guess my daughter's birthday

- You propose a date (day and month)
- I answer, *Yes*, *Earlier*, *Later*
- Suppose my birthday is 12 April
- A possible sequence of questions
  - September 12? Earlier
  - February 23? Later
  - July 2? Earlier
  - ...
- What is the best strategy?

- Interval of possibilities
- Query midpoint — halves the interval
  - June 30? Earlier
  - March 31? Later
  - May 15? Earlier
  - April 22? Earlier
  - April 11? Later
  - April 16? Earlier
  - April 13? Earlier
  - April 12? Yes
- Interval shrinks from 365 $\rightarrow$ 182 $\rightarrow$ 91 $\rightarrow$ 45 $\rightarrow$ 22 $\rightarrow$ 11 $\rightarrow$ 5 $\rightarrow$ 2 $\rightarrow$ 1

  1 2 3 4 5 6 7 8
- Under 10 questions

A real world problem

- Assume Aadhaar details are sorted by Aadhaar number
- Use the halving strategy to check each SIM card

```
for each SIM card S:  M
  probe sorted Aadhaar list to
  find a match with S
```

A real world problem

- Assume Aadhaar details are sorted by Aadhaar number
- Use the halving strategy to check each SIM card
- Halving 10 times reduces the interval by a factor of 1000, because $2^{10} = 1024$

```
for each SIM card S:
  probe sorted Aadhaar list to
  find a match with S
```

$2^{?} = 1024$

$? = 10$

$10 = \log_2 1024$

A real world problem

- Assume Aadhaar details are sorted by Aadhaar number
- Use the halving strategy to check each SIM card
- Halving 10 times reduces the interval by a factor of 1000, because $2^{10} = 1024$
- After 10 queries, interval shrinks to 10^6
- After 20 queries, interval shrinks to 10^3
- After 30 queries, interval shrinks to 1

```
for each SIM card S:
  probe sorted Aadhaar list to
  find a match with S
```

A real world problem

- Assume Aadhaar details are sorted by Aadhaar number
- Use the halving strategy to check each SIM card
- Halving 10 times reduces the interval by a factor of 1000, because $2^{10} = 1024$
- After 10 queries, interval shrinks to 10^6
- After 20 queries, interval shrinks to 10^3
- After 30 queries, interval shrinks to 1
- Total time $\approx 10^9 \times 30$

$M \times \log_2 N$

```
for each SIM card S:
  probe sorted Aadhaar list to
  find a match with S
```

A real world problem

- Assume Aadhaar details are sorted by Aadhaar number
- Use the halving strategy to check each SIM card
- Halving 10 times reduces the interval by a factor of 1000, because $2^{10} = 1024$
- After 10 queries, interval shrinks to 10^6
- After 20 queries, interval shrinks to 10^3
- After 30 queries, interval shrinks to 1
- Total time $\approx 10^9 \times 30$

```
for each SIM card S:
   probe sorted Aadhaar list to
   find a match with S
```

- 3000 seconds, or 50 minutes
- From 3200 years to 50 minutes!

A real world problem

- Assume Aadhaar details are sorted by Aadhaar number
- Use the halving strategy to check each SIM card
- Halving 10 times reduces the interval by a factor of 1000, because $2^{10} = 1024$
- After 10 queries, interval shrinks to 10^6
- After 20 queries, interval shrinks to 10^3
- After 30 queries, interval shrinks to 1
- Total time $\approx 10^9 \times 30$

```
for each SIM card S:
  probe sorted Aadhaar list to
  find a match with S
```

- 3000 seconds, or 50 minutes
- From 3200 years to 50 minutes!
- Of course, to achieve this we have to first sort the Aadhaar cards
- Arranging the data results in a much more efficient solution
- Both algorithms and data structures matter

Comparing orders of magnitude

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming and Data Structures with Python

Lecture 14, 10 Oct 2023

Orders of magnitude

- When comparing $t(n)$, focus on orders of magnitude
  - Ignore constant factors
- $f(n) = n^3$ eventually grows faster than $g(n) = 5000n^2$

$t(n)$ – time to solve a problem of input size n

Size = ? List/array/sequence – length

Checking if a number is prime?

factorlist = - . — run through 1, 2, .. , n

if factorlist == [1, n] : Time proportional to n

n is prime

AKS algorithm Agrawal, Kayal, Saxena

$\approx$ 2002

First poly time algo for primality

What is the size of 10^9?

10 digits — not 10^9 digits

$$\begin{array}{r} 62 \\ +\ 46 \\ \hline 108 \end{array} \qquad \begin{array}{r} 162 \\ +\ 346 \\ \hline 508 \end{array}$$

Arithmetic — input size = # digits

Which inputs of size n?

<u>Sorting</u> Input is already sorted

Input is reverse sorted

⋮

Many "random" permutations

Which one to evaluate?

Sophisticated – average

Average is ideal, but practically impossible to calculate

Instead - pessimistic estimate - "worst case"

Worst case complexity

as a function of input size

How does this "grow" with n? "Asymptotic"

Orders of magnitude

- When comparing $t(n)$, focus on orders of magnitude
  - Ignore constant factors
- $f(n) = n^3$ eventually grows faster than $g(n) = 5000n^2$

  5000^3 5001^3 5000^3 5000 5001 × 5001

- How do we compare functions with respect to orders of magnitude?

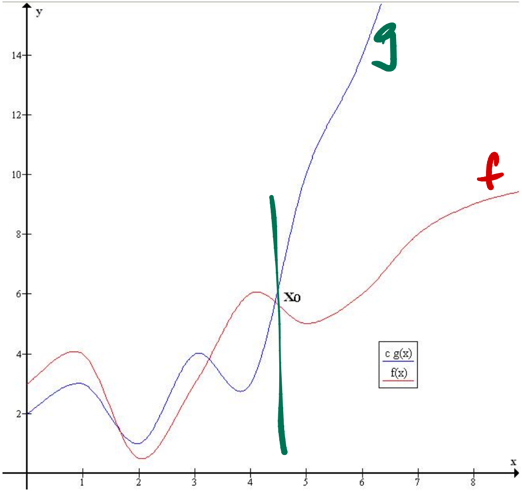
$n = 5000$
5001

Upper bounds

- $f(x)$ is said to be $O(g(x))$ if we can find constants c and x_0 such that $c \cdot g(x)$ is an upper bound for $f(x)$ for x beyond x_0

Upper bounds

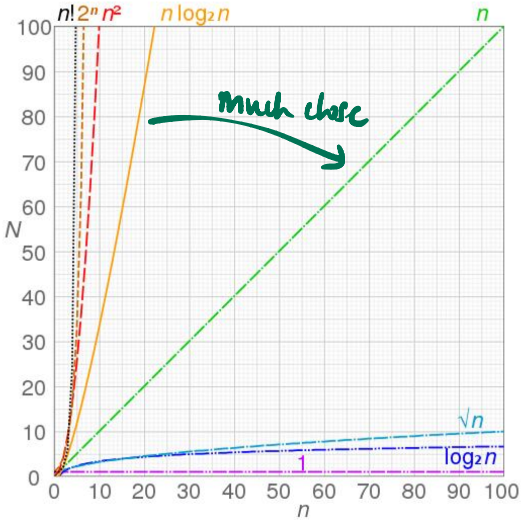
- $f(x)$ is said to be $O(g(x))$ if we can find constants c and x_0 such that $c \cdot g(x)$ is an upper bound for $f(x)$ for x beyond x_0
- $f(x) \leq cg(x)$ for every $x \geq x_0$



Upper bounds

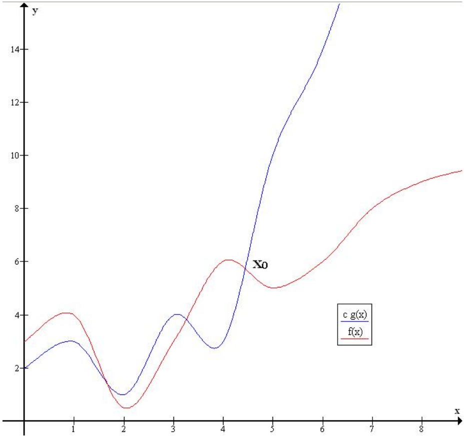
- $f(x)$ is said to be $O(g(x))$ if we can find constants c and x_0 such that $c \cdot g(x)$ is an upper bound for $f(x)$ for x beyond x_0
- $f(x) \leq cg(x)$ for every $x \geq x_0$
- Graphs of typical functions we have seen

f is $O(1)$ $\qquad f \leq c\,1$



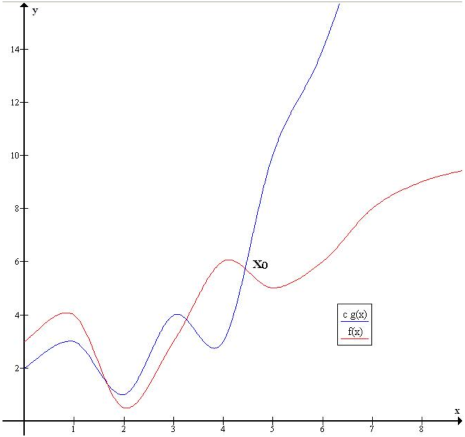
Examples

- $100n + 5$ is $O(n^2)$
  - $100n + 5 \leq 100n + n = 101n$, for $n \geq 5$
  - $101n \leq 101n^2$
  - Choose $n_0 = 5$, $c = 101$



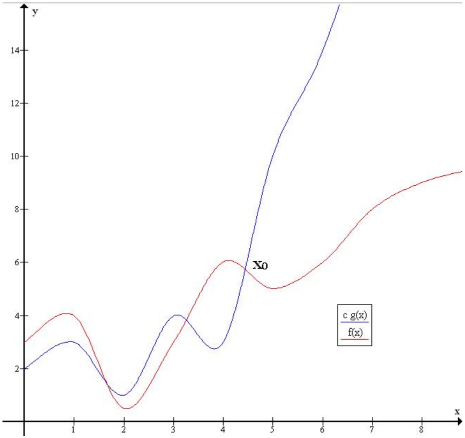
Examples

- $100n + 5$ is $O(n^2)$
  - $100n + 5 \leq 100n + n = 101n$, for $n \geq 5$
  - $101n \leq 101n^2$
  - Choose $n_0 = 5$, $c = 101$
- Alternatively
  - $100n + 5 \leq 100n + 5n = 105n$, for $n \geq 1$
  - $105n \leq 105n^2$
  - Choose $n_0 = 1$, $c = 105$



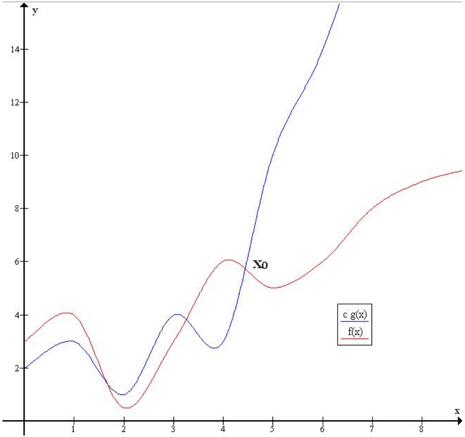
Examples

- $100n + 5$ is $O(n^2)$
  - $100n + 5 \leq 100n + n = 101n$, for $n \geq 5$
  - $101n \leq 101n^2$
  - Choose $n_0 = 5$, $c = 101$
- Alternatively
  - $100n + 5 \leq 100n + 5n = 105n$, for $n \geq 1$
  - $105n \leq 105n^2$
  - Choose $n_0 = 1$, $c = 105$
- Choice of n_0, c not unique



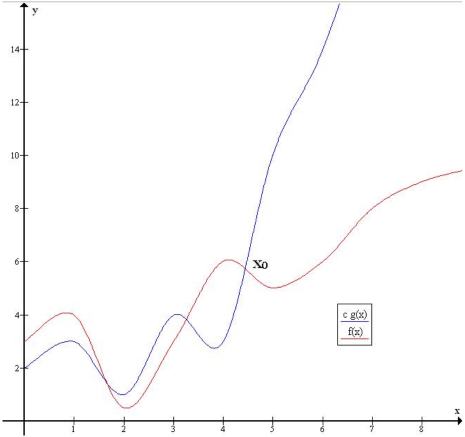
Examples . . .

- $100n^2 + 20n + 5$ is $O(n^2)$
  - $100n^2 + 20n + 5 \leq 100n^2 + 20n^2 + 5n^2$, for $n \geq 1$
  - $100n^2 + 20n + 5 \leq 125n^2$, for $n \geq 1$
  - Choose $n_0 = 1$, $c = 125$



Examples . . .

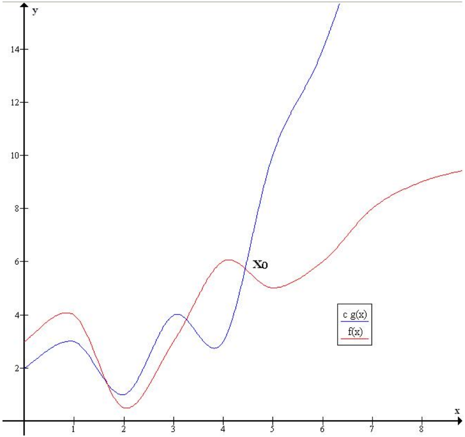
- $100n^2 + 20n + 5$ is $O(n^2)$
  - $100n^2 + 20n + 5 \leq 100n^2 + 20n^2 + 5n^2$, for $n \geq 1$
  - $100n^2 + 20n + 5 \leq 125n^2$, for $n \geq 1$
  - Choose $n_0 = 1$, $c = 125$
- What matters is the highest term
  - $20n + 5$ is dominated by $100n^2$



Examples . . .

$n^7 \leq 5000 \cdot n^2$

- $100n^2 + 20n + 5$ is $O(n^2)$
  - $100n^2 + 20n + 5 \leq 100n^2 + 20n^2 + 5n^2$, for $n \geq 1$
  - $100n^2 + 20n + 5 \leq 125n^2$, for $n \geq 1$
  - Choose $n_0 = 1$, $c = 125$
- What matters is the highest term
  - $20n + 5$ is dominated by $100n^2$
- n^3 is not $O(n^2)$
  - No matter what c we choose, cn^2 will be dominated by n^3 for $n \geq c$



Useful properties

- Algorithm has two phases
  - Phase 1 takes time $O(g_1(n))$ — Sort Aadhaar Cards
  - Phase 2 takes time $O(g_2(n))$ — Scan each SIM card

What can we say about the algorithm as a whole?

Useful properties

- Algorithm has two phases
  - Phase 1 takes time $O(g_1(n))$
  - Phase 2 takes time $O(g_2(n))$

  What can we say about the algorithm as a whole?

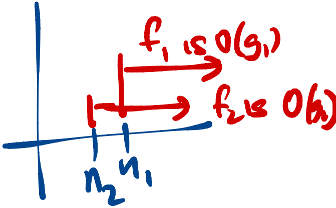
- If $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$, then $f_1(n) + f_2(n)$ is $O(\max(g_1(n), g_2(n)))$

Useful properties

- Algorithm has two phases
  - Phase 1 takes time $O(g_1(n))$
  - Phase 2 takes time $O(g_2(n))$

  What can we say about the algorithm as a whole?

- If $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$, then $f_1(n) + f_2(n)$ is $O(\max(g_1(n), g_2(n)))$



- Proof
  - $f_1(n) \leq c_1 g_1(n)$ for $n > n_1$, $f_2(n) \leq c_2 g_2(n)$ for $n > n_2$

Useful properties

- Algorithm has two phases
  - Phase 1 takes time $O(g_1(n))$
  - Phase 2 takes time $O(g_2(n))$

  What can we say about the algorithm as a whole?

- If $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$, then $f_1(n) + f_2(n)$ is $O(\max(g_1(n), g_2(n)))$

- Proof
  - $f_1(n) \leq c_1 g_1(n)$ for $n > n_1$, $f_2(n) \leq c_2 g_2(n)$ for $n > n_2$
  - Let $c_3 = \max(c_1, c_2)$, $n_3 = \max(n_1, n_2)$

Useful properties

- Algorithm has two phases
  - Phase 1 takes time $O(g_1(n))$
  - Phase 2 takes time $O(g_2(n))$

  What can we say about the algorithm as a whole?

- If $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$, then $f_1(n) + f_2(n)$ is $O(\max(g_1(n), g_2(n)))$

- Proof
  - $f_1(n) \leq c_1 g_1(n)$ for $n > n_1$, $f_2(n) \leq c_2 g_2(n)$ for $n > n_2$
  - Let $c_3 = \max(c_1, c_2)$, $n_3 = \max(n_1, n_2)$
  - For $n \geq n_3$, $f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n)$

c_3

Useful properties

- Algorithm has two phases
  - Phase 1 takes time $O(g_1(n))$
  - Phase 2 takes time $O(g_2(n))$

  What can we say about the algorithm as a whole?

17 32

$\leq 2 \cdot \max(17, 32)$

- If $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$, then $f_1(n) + f_2(n)$ is $O(\max(g_1(n), g_2(n)))$
- Proof
  - $f_1(n) \leq c_1 g_1(n)$ for $n > n_1$, $f_2(n) \leq c_2 g_2(n)$ for $n > n_2$
  - Let $c_3 = \max(c_1, c_2)$, $n_3 = \max(n_1, n_2)$
  - For $n \geq n_3$, $f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n) \leq c_3(g_1(n) + g_2(n))$

Useful properties

- Algorithm has two phases
  - Phase 1 takes time $O(g_1(n))$
  - Phase 2 takes time $O(g_2(n))$

  What can we say about the algorithm as a whole?

- If $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$, then $f_1(n) + f_2(n)$ is $O(\max(g_1(n), g_2(n)))$

- Proof
  - $f_1(n) \leq c_1 g_1(n)$ for $n > n_1$, $f_2(n) \leq c_2 g_2(n)$ for $n > n_2$
  - Let $c_3 = \max(c_1, c_2)$, $n_3 = \max(n_1, n_2)$
  - For $n \geq n_3$, $f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n) \leq c_3(g_1(n) + g_2(n))$ $\leq 2c_3(\max(g_1(n), g_2(n)))$

Useful properties

- Algorithm has two phases
  - Phase 1 takes time $O(g_1(n))$
  - Phase 2 takes time $O(g_2(n))$

  What can we say about the algorithm as a whole?

- If $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$, then $f_1(n) + f_2(n)$ is $O(\max(g_1(n), g_2(n)))$

- Algorithm as a whole takes time $\max(O(g_1(n), g_2(n)))$

- Least efficient phase is the upper bound for the whole algorithm

Lower bounds

- $f(x)$ is said to be $\Omega(g(x))$ if we can find constants c and x_0 such that $cg(x)$ is a lower bound for $f(x)$ for x beyond x_0
  - $f(x) \geq cg(x)$ for every $x \geq x_0$

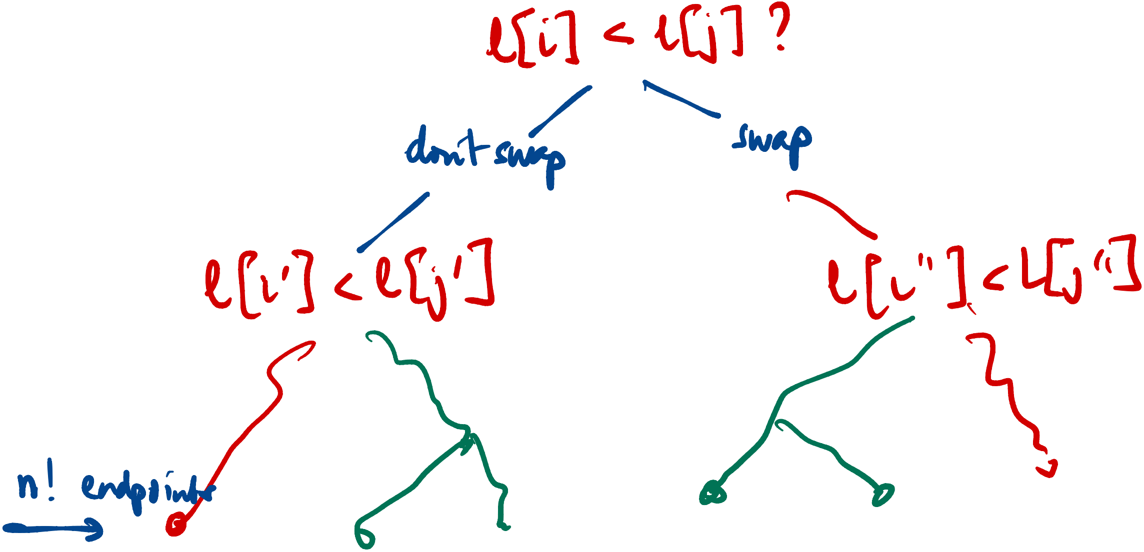
Lower bounds

- $f(x)$ is said to be $\Omega(g(x))$ if we can find constants c and x_0 such that $cg(x)$ is a lower bound for $f(x)$ for x beyond x_0
  - $f(x) \geq cg(x)$ for every $x \geq x_0$
- n^3 is $\Omega(n^2)$
  - $n^3 > n^2$ for all n, so $n_0 = 1$, $c = 1$

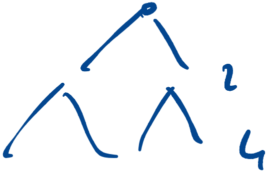
Lower bounds

- $f(x)$ is said to be $\Omega(g(x))$ if we can find constants c and x_0 such that $cg(x)$ is a lower bound for $f(x)$ for x beyond x_0
  - $f(x) \geq cg(x)$ for every $x \geq x_0$
- n^3 is $\Omega(n^2)$
  - $n^3 > n^2$ for all n, so $n_0 = 1$, $c = 1$
- Typically we establish lower bounds for a problem rather than an individual algorithm
  - If we sort a list by comparing elements and swapping them, we require $\Omega(n \log n)$ comparisons
  - This is independent of the algorithm we use for sorting

n element list



Binary



k levels

2^k endpoints

$$2^k \geq n!$$

$$k \geq \log_2(n!) \overset{\leq}{\approx} n \log_2(n)$$

Tight bounds

- $f(x)$ is said to be $\Theta(g(x))$ if it is both $O(g(x))$ and $\Omega(g(x))$
  - Find constants c_1, c_2, x_0 such that $c_1 g(x) \leq f(x) \leq c_2 g(x)$ for every $x \geq x_0$

Tight bounds

- $f(x)$ is said to be $\Theta(g(x))$ if it is both $O(g(x))$ and $\Omega(g(x))$
  - Find constants c_1, c_2, x_0 such that $c_1 g(x) \leq f(x) \leq c_2 g(x)$ for every $x \geq x_0$
- $n(n-1)/2$ is $\Theta(n^2)$

Tight bounds

- $f(x)$ is said to be $\Theta(g(x))$ if it is both $O(g(x))$ and $\Omega(g(x))$
  - Find constants c_1, c_2, x_0 such that $c_1 g(x) \leq f(x) \leq c_2 g(x)$ for every $x \geq x_0$
- $n(n-1)/2$ is $\Theta(n^2)$
  - Upper bound
    - $n(n-1)/2 = n^2/2 - n/2 \leq n^2/2$ for all $n \geq 0$

Tight bounds

- $f(x)$ is said to be $\Theta(g(x))$ if it is both $O(g(x))$ and $\Omega(g(x))$
  - Find constants c_1, c_2, x_0 such that $c_1 g(x) \leq f(x) \leq c_2 g(x)$ for every $x \geq x_0$
- $n(n-1)/2$ is $\Theta(n^2)$
  - Upper bound
    - $n(n-1)/2 = n^2/2 - n/2 \leq n^2/2$ for all $n \geq 0$
  - Lower bound
    - $n(n-1)/2 = n^2/2 - n/2 \geq n^2/2 - (n/2 \times n/2) \geq n^2/4$ for $n \geq 2$

Tight bounds

- $f(x)$ is said to be $\Theta(g(x))$ if it is both $O(g(x))$ and $\Omega(g(x))$
  - Find constants c_1, c_2, x_0 such that $c_1 g(x) \leq f(x) \leq c_2 g(x)$ for every $x \geq x_0$
- $n(n-1)/2$ is $\Theta(n^2)$
  - Upper bound
    - $n(n-1)/2 = n^2/2 - n/2 \leq n^2/2$ for all $n \geq 0$
  - Lower bound
    - $n(n-1)/2 = n^2/2 - n/2 \geq n^2/2 - (n/2 \times n/2) \geq n^2/4$ for $n \geq 2$
  - Choose $n_0 = 2$, $c_1 = 1/4$, $c_2 = 1/2$

Summary

- $f(n)$ is $O(g(n))$ means $g(n)$ is an upper bound for $f(n)$
  - Useful to describe asymptotic worst case running time

Summary

- $f(n)$ is $O(g(n))$ means $g(n)$ is an upper bound for $f(n)$
  - Useful to describe asymptotic worst case running time
- $f(n)$ is $\Omega(g(n))$ means $g(n)$ is a lower bound for $f(n)$
  - Typically used for a problem as a whole, rather than an individual algorihm

Summary

- $f(n)$ is $O(g(n))$ means $g(n)$ is an upper bound for $f(n)$
  - Useful to describe asymptotic worst case running time
- $f(n)$ is $\Omega(g(n))$ means $g(n)$ is a lower bound for $f(n)$
  - Typically used for a problem as a whole, rather than an individual algorihm
- $f(n)$ is $\Theta(g(n))$: matching upper and lower bounds
  - We have found an optimal algorithm for a problem