Logistics:
Course website: https://cs170.org/ (lots of information)
CS 170: Algorithms

**Logistics:**
Course website: https://cs170.org/ (lots of information)
Course communications: Edstem
CS 170: Algorithms

Logistics:
Course website: https://cs170.org/ (lots of information)
Course communications: Edstem
Weekly homeworks + 2 midterms + 1 final exam.
Logistics:
Course website: https://cs170.org/ (lots of information)
Course communications: Edstem
Weekly homeworks + 2 midterms + 1 final exam.
Midterm 1: M 2/26/24 from 7pm-9pm.
Midterm 2: Tu 4/2/24 from 7pm-9pm
CS 170: Algorithms

Logistics:
Course website: https://cs170.org/ (lots of information)
Course communications: Edstem
Weekly homeworks + 2 midterms + 1 final exam.
Midterm 1: M 2/26/24 from 7pm-9pm.
Midterm 2: Tu 4/2/24 from 7pm-9pm
Instructors:
CS 170: Algorithms

**Logistics:**
Course website: https://cs170.org/ (lots of information)
Course communications: Edstem
Weekly homeworks + 2 midterms + 1 final exam.

Midterm 1: M 2/26/24 from 7pm-9pm.
Midterm 2: Tu 4/2/24 from 7pm-9pm

Instructors:
  Christian Borgs
Logistics:
Course website: https://cs170.org/ (lots of information)
Course communications: Edstem
Weekly homeworks + 2 midterms + 1 final exam.
Midterm 1: M 2/26/24 from 7pm-9pm.
Midterm 2: Tu 4/2/24 from 7pm-9pm
Instructors:
  Christian Borgs
  Prasad Raghavendra
Logistics:
Course website: https://cs170.org/ (lots of information)
Course communications: Edstem
Weekly homeworks + 2 midterms + 1 final exam.
Midterm 1: M 2/26/24 from 7pm-9pm.
Midterm 2: Tu 4/2/24 from 7pm-9pm

Instructors:
  Christian Borgs
  Prasad Raghavendra
CS 170: Algorithms

**Logistics:**
Course website: https://cs170.org/ (lots of information)
Course communications: Edstem
Weekly homeworks + 2 midterms + 1 final exam.
Midterm 1: M 2/26/24 from 7pm-9pm.
Midterm 2: Tu 4/2/24 from 7pm-9pm

Instructors:
  - Christian Borgs
  - Prasad Raghavendra

Today: Satish Rao
A puzzle.

Does a list have a cycle?
A puzzle.

Does a list have a cycle?
Access to list is a pointer to the “first element.”
A puzzle.

Does a list have a cycle?
Access to list is a pointer to the “first element.”
Mark first node.
A puzzle.

Does a list have a cycle?

Access to list is a pointer to the “first element.”

Mark first node.

While next cell not marked, go to next cell.
A puzzle.

Does a list have a cycle?
Access to list is a pointer to the “first element.”
Mark first node.
While next cell not marked, go to next cell.
Claim: either there is no next cell, or detects cycle.
A puzzle.

Does a list have a cycle?
Access to list is a pointer to the “first element.”
Mark first node.
While next cell not marked, go to next cell.
Claim: either there is no next cell, or detects cycle.
A puzzle.

Does a list have a cycle?
Access to list is a pointer to the “first element.”
Mark first node.
While next cell not marked, go to next cell.
Claim: either there is no next cell, or detects cycle.
  Intuition: if on cycle, must return.
A puzzle.

Does a list have a cycle?

Access to list is a pointer to the “first element.”

Mark first node.
While next cell not marked, go to next cell.

Claim: either there is no next cell, or detects cycle.

   Intuition: if on cycle, must return.

Quiz: Does this work?
A puzzle.

Does a list have a cycle?

Access to list is a pointer to the “first element.”

Mark first node.

While next cell not marked, go to next cell.

Claim: either there is no next cell, or detects cycle.

  Intuition: if on cycle, must return.

Quiz: Does this work?

a) Yes. b) No.
A puzzle.

Does a list have a cycle?
Access to list is a pointer to the “first element.”
Mark first node.
While next cell not marked, go to next cell.
Claim: either there is no next cell, or detects cycle.
   Intuition: if on cycle, must return.
Quiz: Does this work?
a) Yes. b) No.
A puzzle.

Does a list have a cycle?

Access to list is a pointer to the “first element.”

Mark first node.

While next cell not marked, go to next cell.

Claim: either there is no next cell, or detects cycle.

Intuition: if on cycle, must return.

Quiz: Does this work?

a) Yes. b) No.

First node not in cycle!
A puzzle.

Does a list have a cycle?
Access to list is a pointer to the “first element.”
Mark first node.
While next cell not marked, go to next cell.
Claim: either there is no next cell, or detects cycle.

Intuition: if on cycle, must return.

Quiz: Does this work?
a) Yes. b) No.

First node not in cycle!
Answer is no.
A puzzle.

Does a list have a cycle?
Access to list is a pointer to the “first element.”
Mark first node.
While next cell not marked, go to next cell.
Claim: either there is no next cell, or detects cycle.
   Intuition: if on cycle, must return.
Quiz: Does this work?
   a) Yes. b) No.

First node not in cycle!
Answer is no. “Oracle” gave us example.
A puzzle.

Does a list have a cycle?
Access to list is a pointer to the “first element.”
Mark first node.
While next cell not marked, go to next cell.
Claim: either there is no next cell, or detects cycle.
   Intuition: if on cycle, must return.
Quiz: Does this work?
a) Yes. b) No.

First node not in cycle!
Answer is no. “Oracle” gave us example.
Problem: starting point is not on cycle?
A puzzle.

Does a list have a cycle?

Access to list is a pointer to the “first element.”

Mark first node.
While next cell not marked, go to next cell.

Claim: either there is no next cell, or detects cycle.

Intuition: if on cycle, must return.

Quiz: Does this work?

a) Yes. b) No.

First node not in cycle!

Answer is no. “Oracle” gave us example.

Problem: starting point is not on cycle?
Construct example.
Does a list have a cycle?

Two ptrs:
Step: advance ptr 1 twice,
advance ptr 2 once.

If ever at the same place, report cycle.

Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.

d - distance from fast ptr to slow ptr.
d decreases every step.

Runtime:
n steps to cycle
n steps to catch up.
O(n).

Additional storage: two pointers.
O(1).
Does a list have a cycle?

Two ptrs:
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 twice,
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once.
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 \textit{twice}, advance ptr 2 \textit{once}. If ever at the same place, report cycle.
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once. If ever at the same place, report cycle.

Correctness: If no cycle, slow pointer never catches fast one. If cycle, both pointers will enter cycle at some time. $d$ decreases every step.

Runtime: $n$ steps to cycle $n$ steps to catch up. $O(n)$. Additional storage: two pointers. $O(1)$. 

Does a list have a cycle?

Two ptrs: Step: advance ptr 1 *twice*, advance ptr 2 *once*. If ever at the same place, report cycle.

Correctness:
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 \textit{twice}, advance ptr 2 \textit{once}. If ever at the same place, report cycle.

Correctness:
If no cycle, slow pointer never catches fast one.
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 *twice*, advance ptr 2 *once*. If ever at the same place, report cycle.

Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once. If ever at the same place, report cycle.

Correctness:
If no cycle, slow pointer never catches fast one. If cycle, both pointers will enter cycle at some time. $d$ - distance from fast ptr to slow ptr.
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 **twice**, advance ptr 2 **once**. If ever at the same place, report cycle.

Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.
$d$ - distance from fast ptr to slow ptr.

$d$ decreases every step.
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once. If ever at the same place, report cycle.

Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time. $d$ - distance from fast ptr to slow ptr.

$d$ decreases every step.
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 **twice**, advance ptr 2 **once**. If ever at the same place, report cycle.

Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.
$d$ - distance from fast ptr to slow ptr.

$d$ decreases every step.

Runtime: $n$ steps to cycle
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 \textit{twice}, advance ptr 2 \textit{once}. If ever at the same place, report cycle.

Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.
\(d\) - distance from fast ptr to slow ptr.
\(d\) decreases every step.

Runtime: \(n\) steps to cycle \(n\) steps to catch up.
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once. If ever at the same place, report cycle.

Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.
$d$ - distance from fast ptr to slow ptr.

$d$ decreases every step.

Runtime: $n$ steps to cycle $n$ steps to catch up. $O(n)$
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once. If ever at the same place, report cycle.

Correctness:
If no cycle, slow pointer never catches fast one. If cycle, both pointers will enter cycle at some time.

\( d \) - distance from fast ptr to slow ptr.

\( d \) decreases every step.

Runtime: \( n \) steps to cycle \( n \) steps to catch up. \( O(n) \)

Additional storage: two pointers.
Does a list have a cycle?

Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once. If ever at the same place, report cycle.

Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time. $d$ - distance from fast ptr to slow ptr. $d$ decreases every step.

Runtime: $n$ steps to cycle $n$ steps to catch up. $O(n)$
Additional storage: two pointers. $O(1)$. 
Puzzles

Solutions
Puzzles..

Solutions
..are Algorithms...
Solutions
..are Algorithms...
which...
Puzzles ..

Solutions ..are Algorithms...
which...
..are correct...
Solutions
..are Algorithms...
which...
..are correct...and (in this class) efficient.
Solutions
..are Algorithms...
which...
..are correct...and (in this class) efficient.
Is this a useful process?
Algorithms for the Human Genome Project

Reconstruct DNA...
Reconstruct DNA...

ACTGAAACTGAGTAGATA....
Reconstruct DNA...

ACTGAAACTGAGTAGATA....

Read first, then next, then next, ...3.1 billion times...
.. slow... error prone...
Reconstruct DNA...

ACTGAAACTGAGTAGATA....

Read first, then next, then next, ...3.1 billion times...
.. slow... error prone...

Parallel sequencing yields chunks of overlapping DNA.
Algorithms for the Human Genome Project

Reconstruct DNA...
ACTGAAACTGAGTAGATA....
Read first, then next, then next, ...3.1 billion times...
.. slow... error prone...
Parallel sequencing yields chunks of overlapping DNA.
AGTAG, AGATA, TGAGT , ACTGAA , CTGAA , AAACTG
Reconstruct DNA...

ACTGAAACTGAGTAGATA....

Read first, then next, then next, ...3.1 billion times...
.. slow... error prone...

Parallel sequencing yields chunks of overlapping DNA.
AGTAG, AGATA, TGAGT , ACTGAA , CTGAA , AAACTG

Assemble into a consistent string?
Algorithms for the Human Genome Project

Reconstruct DNA...
ACTGAAACTGAGTAGATA....

Read first, then next, then next, ...3.1 billion times...
.. slow... error prone...

Parallel sequencing yields chunks of overlapping DNA.
AGTAG, AGATA, TGAGT , ACTGAA , CTGAA , AAACTG
Assemble into a consistent string?

ACTGAA
Reconstruct DNA...

ACTGAA ACTGAGTAGATA....

Read first, then next, then next, ...3.1 billion times...
.. slow... error prone...

Parallel sequencing yields chunks of overlapping DNA.
AGTAG, AGATA, TGAGT, ACTGAA, CTGAA, AAACTG

Assemble into a consistent string?

ACTGAA
   AA ACTG

AAACTG
Reconstruct DNA...
ACTGAAACTGAGTAGATA....
Read first, then next, then next, ...3.1 billion times...
.. slow... error prone...
Parallel sequencing yields chunks of overlapping DNA.
AGTAG, AGATA, TGAGT , ACTGAA , CTGAA , AAACTG
Assemble into a consistent string?

ACTGAA
   AAACTG
      TGAGT
Algorithms for the Human Genome Project

Reconstruct DNA...
ACTGAAACTGAGTAGATA....
Read first, then next, then next, ...3.1 billion times...
.. slow... error prone...
Parallel sequencing yields chunks of overlapping DNA.
AGTAG, AGATA, TGAGT, ACTGAA, CTGAA, AAACTG
Assemble into a consistent string?

ACTGAA
   AAACTG
      TGAGT
         AGTAG
Reconstruct DNA...
ACTGAAACTGAGTAGATA....
Read first, then next, then next, ...3.1 billion times...
.. slow... error prone...
Parallel sequencing yields chunks of overlapping DNA.
AGTAG, AGATA, TGAGT , ACTGAA , CTGAA , AAACTG
Assemble into a consistent string?

ACTGAA
   AAACTG
      TGAGT
         AGTAG
            AGATA
Reconstruct DNA...

ACTGAAACTGAGTAGATA....

Read first, then next, then next, ...3.1 billion times...
.. slow... error prone...

Parallel sequencing yields chunks of overlapping DNA.
AGTAG, AGATA, TGAGT, ACTGAA, CTGAA, AAACTG

Assemble into a consistent string?

ACTGAA
    AAACTG
    TGAGT
    AGTAG
    AGATA
-------------------
ACTGAAACTGAGTAGATA
Problem: What is good on the web?
Problem: What is good on the web?
Website that pays search engine most?


Random Surfer Model (Brin-Page):
Follow link, follow link,
.. occasionally jump to random page (with prob. $\varepsilon$).

Popular pages are desirable pages.
PageRank = popularity for random surfer.
Sort search results by PageRank!

Made us happier then.
Google.

Issues: make a bunch of webpages that point to each other.

New Model for user: google.
Problem: What is good on the web?
Website that pays search engine most? Goto.com.
Problem: What is good on the web?
Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page):
Problem: What is good on the web?
Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page): Follow link, follow link,
Problem: What is good on the web?
Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page): Follow link, follow link, occasionally jump to random page (with prob. $\varepsilon$).
Problem: What is good on the web?
Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page): Follow link, follow link,
.. occasionally jump to random page (with prob. $\varepsilon$).

Popular pages are desirable pages.
Problem: What is good on the web?
Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page): Follow link, follow link, .. occasionally jump to random page (with prob. $\varepsilon$).

Popular pages are desirable pages.
PageRank = popularity for random surfer.
Problem: What is good on the web?

Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page): Follow link, follow link, .. occasionally jump to random page (with prob. $\epsilon$).

Popular pages are desirable pages.

PageRank = popularity for random surfer.

Sort search results by PageRank!
Problem: What is good on the web?

Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page): Follow link, follow link, .. occasionally jump to random page (with prob. $\varepsilon$).

Popular pages are desirable pages.

PageRank = popularity for random surfer.

Sort search results by PageRank!

Made us happier then.
Problem: What is good on the web?
Website that pays search engine most? Goto.com.

**Random Surfer Model (Brin-Page):** Follow link, follow link, .. occasionally jump to random page (with prob. $\epsilon$).

Popular pages are desirable pages.

PageRank = popularity for random surfer.

Sort search results by PageRank!

Made us happier then.

Google.
Problem: What is good on the web?
Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page): Follow link, follow link, .. occasionally jump to random page (with prob. $\varepsilon$).

Popular pages are desirable pages.

PageRank = popularity for random surfer.

Sort search results by PageRank!

Made us happier then.

Google.
Problem: What is good on the web?

Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page): Follow link, follow link, .. occasionally jump to random page (with prob. $\varepsilon$).

Popular pages are desirable pages.

PageRank = popularity for random surfer.

Sort search results by PageRank!

Made us happier then.

Google.

Issues: make a bunch of webpages that point to each other.
Problem: What is good on the web?
Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page): Follow link, follow link, .. occasionally jump to random page (with prob. $\epsilon$).

Popular pages are desirable pages.
PageRank = popularity for random surfer.
Sort search results by PageRank!
Made us happier then.
Google.

Issues: make a bunch of webpages that point to each other.
New Model for user: google.
Algorithms...

Driving Directions
Algorithms...

Driving Directions
Airline Scheduling
Algorithms...

Driving Directions
Airline Scheduling
Compiling
Algorithms...

Driving Directions
Airline Scheduling
Compiling
Compression
Algorithms...

Driving Directions
Airline Scheduling
Compiling
Compression
Cryptography
Driving Directions
Airline Scheduling
Compiling
Compression
Cryptography
Optimization
Algorithms...

Driving Directions
Airline Scheduling
Compiling
Compression
Cryptography
Optimization
Search
Algorithms...

Driving Directions
Airline Scheduling
Compiling
Compression
Cryptography
Optimization
Search
Targeted Advertising
Algorithms...

Driving Directions
Airline Scheduling
Compiling
Compression
Cryptography
Optimization
Search
Targeted Advertising
.
.
.
Calculating: 300 BC through Middle Ages in Europe.

I – one

Add them?

1448 + 676 = 2124

676 years since the Gutenberg printing press.

Reading and writing!

For everyone.

Multiply roman numbers?
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten

Add them?

1448 + 676 =

2024

676 years since the Gutenberg printing press.

For everyone.

Multiply roman numbers?
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred

Add them?

1448 + 676 = 2124
676 years since the Gutenberg printing press.
Reading and writing!
Multiply roman numbers?
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred
D – five hundred

Add them?

1448 + 676 =

2024

676 years since the Gutenberg printing press.

Reading and writing!

Multiply roman numbers?
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred
D – five hundred
M – a thousand

Add them?

1448 + 676 = 2124

676 years since the Gutenberg printing press.

Reading and writing!

Multiply roman numbers?
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred
D – five hundred
M – a thousand

1448 + 676 = 2124
676 years since the Gutenberg printing press.
Reading and writing!

Multiply roman numbers?
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred
D – five hundred
M – a thousand

VIII – eight

Add them?

1448 + 676 =

676 years since the Gutenberg printing press.

Reading and writing!

Multiply roman numbers?
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred
D – five hundred
M – a thousand

VIII – eight
DCLXXVI – five hundred plus a hundred plus fifty plus ten plus ten.
Calculating: 300 BC through Middle Ages in Europe.

I – one  
V – five  
X – ten  
C – one hundred  
D – five hundred  
M – a thousand  

VIII – eight  
DCLXXVI – five hundred plus a hundred plus fifty plus ten plus ten plus ten.  
MCDLXVIII – one thousand five hundred minus one hundred ....
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred
D – five hundred
M – a thousand

VIII – eight
DCLXXVI – five hundred plus a hundred plus fifty plus ten plus ten..
MCDLXVIII – one thousand five hundred minus one hundred ....
Add them?
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred
D – five hundred
M – a thousand

VIII – eight
DCLXXVI – five hundred plus a hundred plus fifty plus ten plus ten
MCDLXVIII – one thousand five hundred minus one hundred

Add them?
1448 + 676 =
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred
D – five hundred
M – a thousand

VIII – eight
DCLXXVI – five hundred plus a hundred plus fifty plus ten plus ten...
MCDLXVIII – one thousand five hundred minus one hundred ....

Add them?

1448 + 676 = 2024
Calculating: 300 BC through Middle Ages in Europe.

I – one  
V – five  
X – ten  
C – one hundred  
D – five hundred  
M – a thousand

VIII – eight  
DCLXXVI – five hundred plus a hundred plus fifty plus ten plus ten.  
MCDLXVIII – one thousand five hundred minus one hundred ....  
Add them?  
1448 + 676 = 2024  
676 years since the Gutenberg printing press.
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred
D – five hundred
M – a thousand

VIII – eight
DCLXXVI – five hundred plus a hundred plus fifty plus ten plus ten.
MCDLXVIII – one thousand five hundred minus one hundred.

Add them?

1448 + 676 = 2024

676 years since the Gutenberg printing press.

Reading and writing!
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred
D – five hundred
M – a thousand

VIII – eight
DCLXXVI – five hundred plus a hundred plus fifty plus ten plus ten...
MCDLXVIII – one thousand five hundred minus one hundred ....

Add them?

1448 + 676 = 2024

676 years since the Gutenberg printing press.
   Reading and writing! For everyone.
Calculating: 300 BC through Middle Ages in Europe.

I – one
V – five
X – ten
C – one hundred
D – five hundred
M – a thousand

VIII – eight
DCLXXVI – five hundred plus a hundred plus fifty plus ten plus ten..
MCDLXVIII – one thousand five hundred minus one hundred ....

Add them?
1448 + 676 = 2024

676 years since the Gutenberg printing press.
   Reading and writing! For everyone.

Multiply roman numbers?
Modern system.

From India, via Al Khwarizmi.
Modern system.

From India, via Al Khwarizmi.

He also described recipes for adding, multiplying, solving quadratics, computing digits of $\pi$..
Modern system.

From India, via Al Khwarizmi.
He also described recipes for adding, multiplying, solving quadratics, computing digits of $\pi$.
Algorithms!
Modern system.

From India, via Al Khwarizmi.

He also described recipes for adding, multiplying, solving quadratics, computing digits of \( \pi \).

Algorithms!

Note:
Modern system.

From India, via Al Khwarizmi.
He also described recipes for adding, multiplying, solving quadratics, computing digits of $\pi$.

Algorithms!

Note:

Mayans (base 20):
Modern system.

From India, via Al Khwarizmi.
He also described recipes for adding, multiplying, solving quadratics, computing digits of $\pi$.

Algorithms!

Note:
Mayans (base 20): dots (ones) and underlines (fives).
Modern system.

From India, via Al Khwarizmi.

He also described recipes for adding, multiplying, solving quadratics, computing digits of $\pi$.

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).

13 is “···”
Modern system.

From India, via Al Khwarizmi.

He also described recipes for adding, multiplying, solving quadratics, computing digits of \( \pi \).

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).

13 is “···”

Babylonions (base 60):
From India, via Al Khwarizmi.

He also described recipes for adding, multiplying, solving quadratics, computing digits of $\pi$.

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).

13 is “···”

Babylonions (base 60): clusters of 10 instead of digits.
From India, via Al Khwarizmi.
He also described recipes for adding, multiplying, solving quadratics, computing digits of $\pi$.

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).
  13 is “…”

Babylonions (base 60): clusters of 10 instead of digits.

Abacus
Modern system.

From India, via Al Khwarizmi.
He also described recipes for adding, multiplying, solving quadratics, computing digits of \( \pi \).

Algorithms!

Note:
Mayans (base 20): dots (ones) and underlines (fives).
13 is “···”

Babylonions (base 60): clusters of 10 instead of digits.
Abacus successive rows,
Modern system.

From India, via Al Khwarizmi.
He also described recipes for adding, multiplying, solving quadratics, computing digits of $\pi$.

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).
13 is “···”

Babylonions (base 60): clusters of 10 instead of digits.

Abacus successive rows, successive places..
Modern system.

From India, via Al Khwarizmi.
He also described recipes for adding, multiplying, solving quadratics, computing digits of $\pi$.

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).
13 is “⋯”

Babylonions (base 60): clusters of 10 instead of digits.

Abacus successive rows, successive places.

India: “invented” 0!
Modern system.

From India, via Al Khwarizmi.

He also described recipes for adding, multiplying, solving quadratics, computing digits of π..

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).

13 is “···”

Babylonions (base 60): clusters of 10 instead of digits.

Abacus successive rows, successive places..

India: “invented” 0! ... and decimal symbols.
Modern system.

From India, via Al Khwarizmi.

He also described recipes for adding, multiplying, solving quadratics, computing digits of $\pi$.

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).

13 is “···”

Babylonians (base 60): clusters of 10 instead of digits.

Abacus successive rows, successive places..

India: “invented” 0! ... and decimal symbols.

20th century. Base 2!
Modern system.

From India, via Al Khwarizmi.

He also described recipes for adding, multiplying, solving quadratics, computing digits of $\pi$.

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).

13 is “···”

Babylonions (base 60): clusters of 10 instead of digits.

Abacus successive rows, successive places.

India: “invented” 0! ... and decimal symbols.

20th century. Base 2!

The input representation for modern computers and communication.
Al Khwarizmi:

Go west! You decimal system!
Writing to propagating..

Al Khwarizmi: Go west! Young decimal system!
Al Khwarizmi: Go west! Young decimal system!

Al Khwarizmi used to be transliterated as *Algoritmi* or Algaurizin
Persian mathematician, astronomer, geographer (780-850)
Al Khwarizmi: Go west! Young decimal system!

Al Khwarizmi used to be transliterated as *Algoritmi* or Algaurizin
Persian mathematician, astronomer, geographer (780-850)
Al Khwarizmi: Go west! Young decimal system!

Al Khwarizmi used to be transliterated as *Algoritmi* or *Algaurizin*
Persian mathematician, astronomer, geographer (780-850)

..but Fibonacci popularized its use.
Al Khwarizmi: Go west! Young decimal system!

Al Khwarizmi used to be transliterated as *Algoritmi* or *Algaurizin*
Persian mathematician, astronomer, geographer (780-850)

..but Fibonacci popularized its use.

Italian mathematician (1170-1250) who traveled to learn the Hindu-Arab math.
Place value.

I love place value!!
Place value.

I love place value!!
Democratizes arithmetic.
Place value.

I love place value!!

Democratizes arithmetic. Money.
Place value.

I love place value!!
Democratizes arithmetic. Money. Helps end feudal system?
Place value.

I love place value!!

Democratizes arithmetic. Money. Helps end feudal system?

54879
Place value.

I love place value!!

Democratizes arithmetic. Money. Helps end feudal system?

54879

What does the 9 mean?
I love place value!!

Democratizes arithmetic. Money. Helps end feudal system?

54879

What does the 9 mean? 9
What does the 8 mean?
Place value.

I love place value!!

Democratizes arithmetic. Money. Helps end feudal system?

54879

What does the 9 mean? 9
What does the 8 mean? 7 hundreds.
What does the 5 mean?
Place value.

I love place value!!
Democratizes arithmetic. Money. Helps end feudal system?

54879

What does the 9 mean? 9
What does the 8 mean? 7 hundreds.
What does the 5 mean? $5 \times 10^5$. 
I love place value!!

Democratizes arithmetic. Money. Helps end feudal system?

54879

What does the 9 mean? 9
What does the 8 mean? 7 hundreds.
What does the 5 mean? $5 \times 10^5$.

This is amazing.
I love place value!!

Democratizes arithmetic. Money. Helps end feudal system?

54879

What does the 9 mean? 9
What does the 8 mean? 7 hundreds.
What does the 5 mean? $5 \times 10^5$.

This is amazing.

How many decimal digits in a number between a million and two million?
Place value.

I love place value!!
Democratizes arithmetic. Money. Helps end feudal system?

54879
What does the 9 mean? 9
What does the 8 mean? 7 hundreds.
What does the 5 mean? $5 \times 10^5$.

This is amazing.

How many decimal digits in a number between a million and two million?

7.
Place value.

I love place value!!
Democratizes arithmetic. Money. Helps end feudal system?

54879
What does the 9 mean? 9
What does the 8 mean? 7 hundreds.
What does the 5 mean? $5 \times 10^5$.

This is amazing.

How many decimal digits in a number between a million and two million?

7.

Nice!!!
Place value.

I love place value!!
Democratizes arithmetic. Money. Helps end feudal system?
54879
What does the 9 mean? 9
What does the 8 mean? 7 hundreds.
What does the 5 mean? $5 \times 10^5$.
This is amazing.
How many decimal digits in a number between a million and two million?
  7.
Nice!!!
A million is $10^6$. One more is 7.
Place value.

I love place value!!

Democratizes arithmetic. Money. Helps end feudal system?

54879

What does the 9 mean? 9
What does the 8 mean? 7 hundreds.
What does the 5 mean? $5 \times 10^5$.

This is amazing.

How many decimal digits in a number between a million and two million?

7.

Nice!!!

A million is $10^6$. One more is 7. $6 = \log_{10} \text{(million)}$
Place value.

I love place value!!

Democratizes arithmetic. Money. Helps end feudal system?

54879

What does the 9 mean? 9
What does the 8 mean? 7 hundreds.
What does the 5 mean? $5 \times 10^5$.

This is amazing.

How many decimal digits in a number between a million and two million?

7.

Nice!!!

A million is $10^6$. One more is 7. $6 = \log_{10} \text{(million)}$
Place value.

I love place value!!
Democratizes arithmetic. Money. Helps end feudal system?

54879
What does the 9 mean? 9
What does the 8 mean? 7 hundreds.
What does the 5 mean? $5 \times 10^5$.

This is amazing.

How many decimal digits in a number between a million and two million?

7.

Nice!!

A million is $10^6$. One more is 7. $6 = \log_{10} (\text{million})$

$N$ in decimal takes
Place value.

I love place value!!
Democratizes arithmetic. Money. Helps end feudal system?

54879
What does the 9 mean? 9
What does the 8 mean? 7 hundreds.
What does the 5 mean? $5 \times 10^5$.

This is amazing.

How many decimal digits in a number between a million and two million?

7.

Nice!!!

A million is $10^6$. One more is 7. $6 = \log_{10} \text{(million)}$

$N$ in decimal takes $\lceil \log_{10} N \rceil$ digits.
Fibonacci numbers.

\[ F_0 = 0, \ F_1 = 1. \]
Fibonacci numbers.

\[ F_0 = 0, \quad F_1 = 1. \]
\[ F_n = F_{n-1} + F_{n-2}. \]
Fibonacci numbers.

\[ F_0 = 0, \quad F_1 = 1. \]
\[ F_n = F_{n-1} + F_{n-2}. \]

```python
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

Correct? Implements definition! Run time.

\[ T(n) = T(n-1) + T(n-2) + 2T(n) \geq F_n \]
Fibonacci numbers.

\[ F_0 = 0, \ F_1 = 1. \]
\[ F_n = F_{n-1} + F_{n-2}. \]

def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)

Correct?
Fibonacci numbers.

\[ F_0 = 0, \quad F_1 = 1. \]
\[ F_n = F_{n-1} + F_{n-2}. \]

def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)

Correct? Implements definition!
Fibonacci numbers.

\[ F_0 = 0, \ F_1 = 1. \]
\[ F_n = F_{n-1} + F_{n-2}. \]

```python
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

Correct? Implements definition!

Run time.
Fibonacci numbers.

\[ F_0 = 0, \quad F_1 = 1. \]

\[ F_n = F_{n-1} + F_{n-2}. \]

def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)

Correct? Implements definition!

Run time.

\[ T(n) = T(n-1) + T(n-2) + 2 \]
Fibonacci numbers.

\[ F_0 = 0, \quad F_1 = 1. \]
\[ F_n = F_{n-1} + F_{n-2}. \]

```python
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

Correct? Implements definition!

Run time.

\[ T(n) = T(n-1) + T(n-2) + 2 \]
\[ T(n) \geq F_n \]
Fibonacci algorithm and numbers!

\[ F_n = F_{n-1} + F_{n-2} \]
Fibonacci algorithm and numbers!

\[ F_n = F_{n-1} + F_{n-2} = F_{n-2} + F_{n-3} + F_{n-2} \]
Fibonacci algorithm and numbers!

\[ F_n = F_{n-1} + F_{n-2} = F_{n-2} + F_{n-3} + F_{n-2} \geq 2F_{n-2} \]
Fibonacci algorithm and numbers!

\[ F_n = F_{n-1} + F_{n-2} = F_{n-2} + F_{n-3} + F_{n-2} \geq 2F_{n-2} \]

By induction, we get \( F_n \geq 2^{n/2} \).
Fibonacci algorithm and numbers!

\[ F_n = F_{n-1} + F_{n-2} = F_{n-2} + F_{n-3} + F_{n-2} \geq 2F_{n-2} \]

By induction, we get \( F_n \geq 2^{n/2} \).

From book.. \( F_n \approx 2^{0.694n} \).
Fibonacci algorithm and numbers!

\[ F_n = F_{n-1} + F_{n-2} = F_{n-2} + F_{n-3} + F_{n-2} \geq 2F_{n-2} \]

By induction, we get \( F_n \geq 2^{n/2} \).

From book.. \( F_n \approx 2^{0.694n} \).

\[ T(n) \geq 2^{n/2} \]
Fibonacci algorithm and numbers!

\[ F_n = F_{n-1} + F_{n-2} = F_{n-2} + F_{n-3} + F_{n-2} \geq 2F_{n-2} \]

By induction, we get \( F_n \geq 2^{n/2} \).

From book.. \( F_n \approx 2^{0.694n} \).

\[ T(n) \geq 2^{n/2} \]

From book \( T(n) \geq 2^{0.694n} \).
Fibonacci algorithm and numbers!

\[ F_n = F_{n-1} + F_{n-2} = F_{n-2} + F_{n-3} + F_{n-2} \geq 2F_{n-2} \]

By induction, we get \( F_n \geq 2^{n/2} \).
   From book.. \( F_n \approx 2^{0.694n} \).

\[ T(n) \geq 2^{n/2} \]
   From book \( T(n) \geq 2^{0.694n} \)

For \( n = 100 \), this is around \( 2^{64} \) operations, (more than a thousand years or so on a fast computer.)
Fibonacci algorithm and numbers!

\[ F_n = F_{n-1} + F_{n-2} = F_{n-2} + F_{n-3} + F_{n-2} \geq 2F_{n-2} \]

By induction, we get \( F_n \geq 2^{n/2} \).

From book.. \( F_n \approx 2^{0.694n} \).

\[ T(n) \geq 2^{n/2} \]

From book \( T(n) \geq 2^{0.694n} \)

For \( n = 100 \), this is around \( 2^{64} \) operations, (more than a thousand years or so on a fast computer.)

Exponential algorithm. Bad.
Fibonacci algorithm and numbers!

\[ F_n = F_{n-1} + F_{n-2} = F_{n-2} + F_{n-3} + F_{n-2} \geq 2F_{n-2} \]

By induction, we get \( F_n \geq 2^{n/2} \).

From book. \( F_n \approx 2^{0.694n} \).

\( T(n) \geq 2^{n/2} \)

From book \( T(n) \geq 2^{0.694n} \)

For \( n = 100 \), this is around \( 2^{64} \) operations, (more than a thousand years or so on a fast computer.)

Exponential algorithm. Bad. Grows very fast.
Fibonacci algorithm and numbers!

\[ F_n = F_{n-1} + F_{n-2} = F_{n-2} + F_{n-3} + F_{n-2} \geq 2F_{n-2} \]

By induction, we get \( F_n \geq 2^{n/2} \).

From book.. \( F_n \approx 2^{0.694n} \).

\[ T(n) \geq 2^{n/2} \]

From book \( T(n) \geq 2^{0.694n} \)

For \( n = 100 \), this is around \( 2^{64} \) operations, (more than a thousand years or so on a fast computer.)

Exponential algorithm. Bad. Grows very fast.

Can we do better?
Better Algorithm.

```python
def fib(n):
    if n <= 1:
        return n
    else:
        a = [0, 1]
        for i in range(2, n+1):
            a.append(a[i-1]+a[i-2])
        return a[n]
```

$O(n)$ operations!

Maybe.

Let's try it!

Oops: doubling the size more than doubled the runtime!
Better Algorithm.

def fib(n):
    if n <= 1:
        return n
    else:
        a = [0,1]
        for i in range(2,n+1):
            a.append(a[i-1]+a[i-2])
        return a[n]
def fib(n):
    if n <= 1:
        return n
    else:
        a = [0,1]
        for i in range(2,n+1):
            a.append(a[i-1]+a[i-2])
        return a[n]

$O(n)$ operations!
Better Algorithm.

```python
def fib(n):
    if n <= 1:
        return n
    else:
        a = [0, 1]
        for i in range(2, n+1):  # O(n)
            a.append(a[i-1]+a[i-2])
        return a[n]
```

$O(n)$ operations! Maybe.
def fib(n):
    if n <= 1:
        return n
    else:
        a = [0, 1]
        for i in range(2, n+1):
            O(n)
            a.append(a[i-1]+a[i-2])
        return a[n]

$O(n)$ operations! Maybe.
Let’s try it!
def fib(n):
    if n <= 1:
        return n
    else:
        a = [0,1]
        for i in range(2,n+1):  # O(n)
            a.append(a[i-1]+a[i-2])
        return a[n]

O(n) operations! Maybe.
Let’s try it!
Oops:
Better Algorithm.

```python
def fib(n):
    if n <= 1:
        return n
    else:
        a = [0, 1]
        for i in range(2, n+1):  # O(n)
            a.append(a[i-1] + a[i-2])
        return a[n]
```

$O(n)$ operations! Maybe.

Let’s try it!

Oops:
  doubling the size more than doubled the runtime!
How many bits in the representation of $F_n$?
From demo: Size matters.

How many bits in the representation of $F_n$?
Remember $F_n \approx 2^{0.694n}$. 
How many bits in the representation of $F_n$?
Remember $F_n \approx 2^{0.694n}$.
About how many bits in $F_n$?
How many bits in the representation of $F_n$?

Remember $F_n \approx 2^{0.694n}$.

About how many bits in $F_n$?

$\log_2 F_n$
From demo: Size matters.

How many bits in the representation of $F_n$?
Remember $F_n \approx 2^{0.694n}$.
About how many bits in $F_n$?
$\log_2 F_n \approx 0.6294n$
How many bits in the representation of \( F_n \)?

Remember \( F_n \approx 2^{0.694n} \).

About how many bits in \( F_n \)?

\[ \log_2 F_n \approx 0.6294n \]
How many bits in the representation of $F_n$?

Remember $F_n \approx 2^{0.694n}$.

About how many bits in $F_n$?

$\log_2 F_n \approx 0.6294n$

How long does it take to compute $F_{n-1} + F_{n-2}$?
How many bits in the representation of $F_n$?

Remember $F_n \approx 2^{0.694n}$.

About how many bits in $F_n$?
\[ \log_2 F_n \approx 0.6294 n \]

How long does it take to compute $F_{n-1} + F_{n-2}$?

$O(n)$. 

From demo: Size matters.

How many bits in the representation of $F_n$?
Remember $F_n \approx 2^{0.694n}$.

About how many bits in $F_n$?

$$\log_2 F_n \approx 0.6294n$$

How long does it take to compute $F_{n-1} + F_{n-2}$?

$O(n)$.

How long does Fib take?
From demo: Size matters.

How many bits in the representation of $F_n$?

Remember $F_n \approx 2^{0.694n}$.

About how many bits in $F_n$?

$$\log_2 F_n \approx 0.6294n$$

How long does it take to compute $F_{n-1} + F_{n-2}$?

$O(n)$. 

How long does Fib take?

$n$ additions.
From demo: Size matters.

How many bits in the representation of $F_n$?
Remember $F_n \approx 2^{0.694n}$.
About how many bits in $F_n$?
\[
\log_2 F_n \approx 0.6294n
\]

How long does it take to compute $F_{n-1} + F_{n-2}$?
$O(n)$.

How long does Fib take?
$n$ additions.
At most $O(n^2)$. 
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.
Example: $cn^2$ runtime.
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.

Example: \( cn^2 \) runtime.

Calculation: \( c(2n)^2 \)
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.

Example: $cn^2$ runtime.

Calculation: $c(2n)^2 = 4cn^2$. 
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.

Example: $cn^2$ runtime.

Calculation: $c(2n)^2 = 4cn^2$.

Polynomial time algorithm has runtime $O(n^k)$ for a constant $k$. 
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.
Example: $cn^2$ runtime.

Calculation: $c(2n)^2 = 4cn^2$.

Polynomial time algorithm has runtime $O(n^k)$ for a constant $k$.
Calculation: $(\alpha n)^k = \alpha^k n^k$. 
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.

Example: \( cn^2 \) runtime.

Calculation: \( c(2n)^2 = 4cn^2 \).

Polynomial time algorithm has runtime \( O(n^k) \) for a constant \( k \).

Calculation: \( (\alpha n)^k = \alpha^k n^k \).

Scaling input by \( \alpha \) grows runtime bound by \( \alpha^k \).
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.

Example: $cn^2$ runtime.

Calculation: $c(2n)^2 = 4cn^2$.

Polynomial time algorithm has runtime $O(n^k)$ for a constant $k$.

Calculation: $(\alpha n)^k = \alpha^k n^k$.

Scaling input by $\alpha$ grows runtime bound by $\alpha^k$.

Doubling size, scales runtime by a constant for polynomial time algorithm.
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.
Example: $cn^2$ runtime.

Calculation: $c(2n)^2 = 4cn^2$.

Polynomial time algorithm has runtime $O(n^k)$ for a constant $k$.
Calculation: $(\alpha n)^k = \alpha^k n^k$.

Scaling input by $\alpha$ grows runtime bound by $\alpha^k$.

Doubling size, scales runtime by a constant for polynomial time algorithm.

Not true for exponential algorithms.
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.

Example: $cn^2$ runtime.

Calculation: $c(2n)^2 = 4cn^2$.

Polynomial time algorithm has runtime $O(n^k)$ for a constant $k$.

Calculation: $(\alpha n)^k = \alpha^k n^k$.

Scaling input by $\alpha$ grows runtime bound by $\alpha^k$.

Doubling size, scales runtime by a constant for polynomial time algorithm.

Not true for exponential algorithms. Squares runtime!
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.

Example: $cn^2$ runtime.

Calculation: $c(2n)^2 = 4cn^2$.

Polynomial time algorithm has runtime $O(n^k)$ for a constant $k$.

Calculation: $(\alpha n)^k = \alpha^k n^k$.

Scaling input by $\alpha$ grows runtime bound by $\alpha^k$.

Doubling size, scales runtime by a constant for polynomial time algorithm.

Not true for exponential algorithms. Squares runtime!

Calculation: $2^{2n} = (2^n)^2$. 
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.

Example: \( cn^2 \) runtime.

Calculation: \( c(2n)^2 = 4cn^2 \).

Polynomial time algorithm has runtime \( O(n^k) \) for a constant \( k \).

Calculation: \( (\alpha n)^k = \alpha^k n^k \).

Scaling input by \( \alpha \) grows runtime bound by \( \alpha^k \).

Doubling size, scales runtime by a constant for polynomial time algorithm.

Not true for exponential algorithms. Squares runtime!

Calculation: \( 2^{2n} = (2^n)^2 \).

From: \( 2^{ab} = (2^a)^b \)
Demo: polynomial.

Doubling size, made fast fib grow by factor of roughly four.
Example: \( cn^2 \) runtime.
Calculation: \( c(2n)^2 = 4cn^2 \).

Polynomial time algorithm has runtime \( O(n^k) \) for a constant \( k \).
Calculation: \( (\alpha n)^k = \alpha^k n^k \).

Scaling input by \( \alpha \) grows runtime bound by \( \alpha^k \).
Doubling size, scales runtime by a constant for polynomial time algorithm.

Not true for exponential algorithms. Squares runtime!
Calculation: \( 2^{2n} = (2^n)^2 \).
From: \( 2^{ab} = (2^a)^b = (2^b)^a \).
Asymptotic Analysis.

Used $O(n)$ for number of additions, rather than $n - 2$. 
Asymptotic Analysis.

Used $O(n)$ for number of additions, rather than $n - 2$. Why?
Asymptotic Analysis.

Used $O(n)$ for number of additions, rather than $n - 2$.

Why?

61a, 61b..
Asymptotic Analysis.

Used $O(n)$ for number of additions, rather than $n - 2$.

Why?

61a, 61b..

Recursive fib has faster inner loop than iterative fib.
Asymptotic Analysis.

Used $O(n)$ for number of additions, rather than $n - 2$.
Why?
61a, 61b..
Recursive fib has faster inner loop than iterative fib.
Does it matter?
Asymptotic Analysis.

Used $O(n)$ for number of additions, rather than $n - 2$.

Why?

61a, 61b..

Recursive fib has faster inner loop than iterative fib.

Does it matter?

$2^{.694n}$ versus $O(n^2)$. 
Asymptotic Analysis.

Used $O(n)$ for number of additions, rather than $n - 2$.
Why?
61a, 61b..
Recursive fib has faster inner loop than iterative fib.
Does it matter?
$2^{.694n}$ versus $O(n^2)$.
For $2^{.694n}$, doubling $n$, squares run time.
Asymptotic Analysis.

Used $O(n)$ for number of additions, rather than $n - 2$.
Why?
61a, 61b..
Recursive fib has faster inner loop than iterative fib.
Does it matter?
$2^{.694n}$ versus $O(n^2)$.
For $2^{.694n}$, doubling $n$, squares run time.
   Calculation: $2^{.694(2n)} = (2^{.694n})^2$.
   From: $2^{ab} = (2^b)^a$. 
Asymptotic Analysis.

Used $O(n)$ for number of additions, rather than $n - 2$.

Why?

61a, 61b.

Recursive fib has faster inner loop than iterative fib.

Does it matter?

$2^{.694n}$ versus $O(n^2)$.

For $2^{.694n}$, doubling $n$, squares run time.

Calculation: $2^{.694(2n)} = (2^{.694n})^2$.

From: $2^{ab} = (2^b)^a$.

For $O(n^2)$, doubling $n$, multiplies run time by four.
Asymptotic Analysis.

Used $O(n)$ for number of additions, rather than $n - 2$.

Why?

61a, 61b..

Recursive fib has faster inner loop than iterative fib.

Does it matter?

$2^{.694n}$ versus $O(n^2)$.

For $2^{.694n}$, doubling $n$, squares run time.

Calculation: $2^{.694(2n)} = (2^{.694n})^2$.

From: $2^{ab} = (2^b)^a$.

For $O(n^2)$, doubling $n$, multiplies run time by four.

Calculation: $c(2n)^2 = 4 \times cn^2$
Asymptotic Analysis.

Used $O(n)$ for number of additions, rather than $n - 2$.

Why?

61a, 61b.

Recursive fib has faster inner loop than iterative fib.

Does it matter?

$2^{.694n}$ versus $O(n^2)$.

For $2^{.694n}$, doubling $n$, squares run time.

Calculation: $2^{.694(2n)} = (2^{.694n})^2$.

From: $2^{ab} = (2^b)^a$.

For $O(n^2)$, doubling $n$, multiplies run time by four.

Calculation: $c(2n)^2 = 4 \times cn^2$
Refreshing Asymptotic Notation.

Ignore constant factors.
Refreshing Asymptotic Notation.

Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$
Ignore constant factors.

\(2n^2\) asymptotically same as \(4n^2\)
both are \(O(n^2)\)
Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$
both are $O(n^2)$

$4 \log n$ asymptotically same as $100 \log n$
Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$
both are $O(n^2)$

$4 \log n$ asymptotically same as $100 \log n$
both are $O(\log n)$
Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$
  both are $O(n^2)$

$4\log n$ asymptotically same as $100\log n$
  both are $O(\log n)$

Example?
Refreshing Asymptotic Notation.

Ignore constant factors.

\[ 2n^2 \text{ asymptotically same as } 4n^2 \]
both are \( O(n^2) \)

\[ 4 \log n \text{ asymptotically same as } 100 \log n \]
both are \( O(\log n) \)

Example? Binary search.
Refreshing Asymptotic Notation.

Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$
   both are $O(n^2)$

$4\log n$ asymptotically same as $100\log n$
   both are $O(\log n)$

Example? Binary search.

Ignore smaller order terms.
Refreshing Asymptotic Notation.

Ignore constant factors.

\( 2n^2 \) asymptotically same as \( 4n^2 \)
both are \( O(n^2) \)

\( 4 \log n \) asymptotically same as \( 100 \log n \)
both are \( O(\log n) \)

Example? Binary search.

Ignore smaller order terms.

\( 2n^2 + 100 \)
Refreshing Asymptotic Notation.

Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$
  both are $O(n^2)$

$4\log n$ asymptotically same as $100\log n$
  both are $O(\log n)$

Example? Binary search.

Ignore smaller order terms.

$2n^2 + 100$ is $O(n^2)$
Refreshing Asymptotic Notation.

Ignore constant factors.

\(2n^2\) asymptotically same as \(4n^2\)
both are \(O(n^2)\)

\(4\log n\) asymptotically same as \(100\log n\)
both are \(O(\log n)\)
Example? Binary search.

Ignore smaller order terms.

\(2n^2 + 100\) is \(O(n^2)\)
\(2n^2 + 1000\log n\)
Refreshing Asymptotic Notation.

Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$
  both are $O(n^2)$

$4\log n$ asymptotically same as $100\log n$
  both are $O(\log n)$

Example? Binary search.

Ignore smaller order terms.

$2n^2 + 100$ is $O(n^2)$
$2n^2 + 1000\log n$ is $O(n^2)$
Refreshing Asymptotic Notation.

Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$
  both are $O(n^2)$

$4\log n$ asymptotically same as $100\log n$
  both are $O(\log n)$

Example? Binary search.

Ignore smaller order terms.

$2n^2 + 100$ is $O(n^2)$

$2n^2 + 1000\log n$ is $O(n^2)$

Upper bound.

$n^2$
Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$
  both are $O(n^2)$

$4 \log n$ asymptotically same as $100 \log n$
  both are $O(\log n)$

Example? Binary search.

Ignore smaller order terms.

$2n^2 + 100$ is $O(n^2)$
$2n^2 + 1000 \log n$ is $O(n^2)$

Upper bound.

$n^2$ is $O(n^3)$. 

Formally, for positive functions $g, f$ from integers to reals, $g(n) = O(f(n))$, if there is a constant $c$ where $g(n) \leq cf(n)$.
Refreshing Asymptotic Notation.

Ignore constant factors.

\(2n^2\) asymptotically same as \(4n^2\)
both are \(O(n^2)\)

\(4\log n\) asymptotically same as \(100\log n\)
both are \(O(\log n)\)
Example? Binary search.

Ignore smaller order terms.

\(2n^2 + 100\) is \(O(n^2)\)
\(2n^2 + 1000\log n\) is \(O(n^2)\)

Upper bound.
\(n^2\) is \(O(n^3)\).
\(\log n\)
Refreshing Asymptotic Notation.

Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$
both are $O(n^2)$

$4\log n$ asymptotically same as $100\log n$
both are $O(\log n)$
Example? Binary search.

Ignore smaller order terms.

$2n^2 + 100$ is $O(n^2)$
$2n^2 + 1000\log n$ is $O(n^2)$

Upper bound.

$n^2$ is $O(n^3)$.
$\log n$ is $O(n)$. 
Refreshing Asymptotic Notation.

Ignore constant factors.

\[ 2n^2 \text{ asymptotically same as } 4n^2 \]
both are \( O(n^2) \)

\[ 4 \log n \text{ asymptotically same as } 100 \log n \]
both are \( O(\log n) \)

Example? Binary search.

Ignore smaller order terms.

\[ 2n^2 + 100 \text{ is } O(n^2) \]
\[ 2n^2 + 1000 \log n \text{ is } O(n^2) \]

Upper bound.

\[ n^2 \text{ is } O(n^3) \]
\[ \log n \text{ is } O(n) \]

Formally, for positive functions \( g, f \) from integers to reals,
\[ g(n) = O(f(n)) \text{ , if there is a constant } c \text{ where } g(n) \leq cf(n). \]
More asymptotic notation.

$\Omega$ notation.

Formally, for positive functions $g, f$ from integers to reals, $g(n) = \Omega(f(n))$, if there is a constant $c$ where $g(n) \geq cf(n)$.
More asymptotic notation.

Ω notation.
A “lower bound”.
More asymptotic notation.

$\Omega$ notation.

A “lower bound”.

$2n^2$ is $\Omega(n^2)$ ...
More asymptotic notation.

$\Omega$ notation.

A “lower bound”.

$2n^2$ is $\Omega(n^2)$ ...and $\Omega(n)$...
More asymptotic notation.

Ω notation.
A “lower bound”.

$2n^2$ is $\Omega(n^2)$ ...and $\Omega(n)$...
More asymptotic notation.

Ω notation.
A “lower bound”.
2n^2 is Ω(n^2) ... and Ω(n)...

Formally, for positive functions g, f from integers to reals,
g(n) = Ω(f(n)), if there is a constant c where g(n) ≥ cf(n)
More asymptotic notation.

Ω notation.
A “lower bound”.
$2n^2$ is $\Omega(n^2)$ ...and $\Omega(n)$...

Formally, for positive functions $g, f$ from integers to reals, $g(n) = \Omega(f(n))$, if there is a constant $c$ where $g(n) \geq cf(n)$

$g(n) = \Theta(f(n))$ if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$. 
Al Khwarizmi: Arithmetic.

Addition:

Place value. Algorithm: add places.

```
1 2 3 4 5 6 7 8 9
+ 9 2 1 2 3 7 6 9 1
```

Correctness: See how many ones, if more than 10, add to 10's. And so on.

Time: $O(n)$

Can we do better? Need to look at the numbers to add them... must be optimal.
Al Khwarizmi: Arithmetic.

Addition: $O(n)$
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value.
Addition: $O(n)$

Place value. Algorithm: add places.

Correctness: See how many ones, if more than 10, add to 10’s. And so on.

Time: $O(n)$

Can we do better? Need to look at the numbers to add them... must be optimal.
Addition: $O(n)$

Place value. Algorithm: add places.
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{ccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
4 & 4 & 8 & 0
\end{array}
\]
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{ccccccc}
0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
9 & 4 & 4 & 8 & 0 \\
\end{array}
\]

Correctness: See how many ones, if more than 10, add to 10’s. And so on.

Time: $O(n)$

Can we do better? Need to look at the numbers to add them... must be optimal.
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccccc}
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
4 & 6 & 9 & 4 & 4 & 8 & 0 \\
\end{array}
\]
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
4 & 4 & 6 & 9 & 4 & 4 & 8 & 0 \\
\end{array}
\]
Al Khwarizmi: Arithmetic.

Addition: \( O(n) \)

<table>
<thead>
<tr>
<th>Place value</th>
<th>Algorithm: add places.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 0 0 1 1 1 1</td>
<td>1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>+ 9 2 1 2 3 7 6 9 1</td>
<td>+ 9 2 1 2 3 7 6 9 1</td>
</tr>
<tr>
<td>0 4 4 6 9 4 4 8 0</td>
<td>0 4 4 6 9 4 4 8 0</td>
</tr>
</tbody>
</table>
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccccc}
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 0 & 4 & 4 & 6 & 9 & 4 & 4 & 8 & 0 \\
\end{array}
\]
Addition: $O(n)$

Place value. Algorithm: add places.

```
  1 0 0 0 0 1 1 1 1 1
  1 2 3 4 5 6 7 8 9
+  9 2 1 2 3 7 6 9 1
```

1 0 4 4 6 9 4 4 8 0
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 0 & 4 & 4 & 6 & 9 & 4 & 4 & 8 & 0 \\
\end{array}
\]

Correctness: See how many ones, if more than 10, add to 10's. And so on.

Time: $O(n)$

Can we do better? Need to look at the numbers to add them... must be optimal.
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 0 & 4 & 4 & 6 & 9 & 4 & 4 & 8 & 0 \\
\end{array}
\]
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value.  Algorithm: add places.

\[
\begin{array}{cccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 0 & 4 & 4 & 6 & 9 & 4 & 4 & 8 & 0 \\
\end{array}
\]
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 0 & 4 & 4 & 6 & 9 & 4 & 4 & 8 & 0 \\
\end{array}
\]
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 0 & 4 & 4 & 6 & 9 & 4 & 4 & 8 & 0
\end{array}
\]

Correctness:
Al Khwarizmi: Arithmetic.

Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccccc}
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 0 & 4 & 4 & 6 & 9 & 4 & 4 & 8 & 0 \\
\end{array}
\]

Correctness:
See how many ones, if more than 10, add to 10’s.
Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 0 & 4 & 4 & 6 & 9 & 4 & 4 & 8 & 0 \\
\end{array}
\]

Correctness:
See how many ones, if more than 10, add to 10’s. And so on.
Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccccc}
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 0 & 4 & 4 & 6 & 9 & 4 & 4 & 8 & 0
\end{array}
\]

Correctness:
See how many ones, if more than 10, add to 10’s.
And so on.

Time: $O(n)$
Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 0 & 4 & 4 & 6 & 9 & 4 & 4 & 8 & 0 \\
\end{array}
\]

Correctness:
See how many ones, if more than 10, add to 10’s.
And so on.

Time: $O(n)$

Can we do better?
Addition: $O(n)$

Place value. Algorithm: add places.

\[
\begin{array}{cccccccccc}
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
+ & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 0 & 4 & 4 & 6 & 9 & 4 & 4 & 8 & 0
\end{array}
\]

Correctness:
See how many ones, if more than 10, add to 10’s. And so on.

Time: $O(n)$

Can we do better?

Need to look at the numbers to add them...
Addition: $O(n)$

Place value. Algorithm: add places.

```
  1 0 0 0 0 1 1 1 1 1
  1 2 3 4 5 6 7 8 9
+ 9 2 1 2 3 7 6 9 1
```

Correctness:
See how many ones, if more than 10, add to 10’s.
And so on.

Time: $O(n)$

Can we do better?

Need to look at the numbers to add them... must be optimal.
More Al Khwarizmi’s: algorithms.

Addition: $O(n)$
More Al Khwarizmi’s: algorithms.

Addition: $O(n)$

Multiplication:

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\end{array}
\]
More Al Khwarizmi’s: algorithms.

Addition: $O(n)$

Multiplication:

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9
\end{array}
\]
More Al Khwarizmi’s: algorithms.

Addition: $O(n)$

Multiplication:

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
9 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\
\end{array}
\]
More Al Khwarizmi’s: algorithms.

Addition: $O(n)$

Multiplication:

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
9 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 2 \\
9 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 2 \\
\ldots & \\ \\
\end{array}
\]
More Al Khwarizmi’s: algorithms.

Addition: $O(n)$

Multiplication:

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
9 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\
\end{array}
\]
More Al Khwarizmi’s: algorithms.

Addition: $O(n)$

Multiplication:

\[
\begin{array}{cccccccccccc}
& 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
& 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
& 9 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\
\end{array}
\]

Time: $O(n^2)$
More Al Khwarizmi’s: algorithms.

Addition: $O(n)$

Multiplication:

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
9 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\
\end{array}
\]

Time: $O(n^2)$
Multiplication

Multiplication: $O(n^2)$. 

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!

$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes. 
(b) No.

No.

We can do better!

What ?!?!?

Really!

What does Python do?

Let's see.

Runtime: $2$ times as large increases by a factor of $3$.

Note: $2 \log_2 3 = 3$.

Runtime: $O(n \log_2 3)$.

How?

Next time (read ahead!!!)
Multiplication

Multiplication: $O(n^2)$. Is the best possible?

Since every digit in $x$ must multiply every digit in $y$ at least once, we have $\Theta(n^2)$ such pairs. Is this the best possible?

(a) Yes.
(b) No.

No.

We can do better!

What?!!

Really!

What does Python do?

Let's see.

Runtime: 2 times as large increases by a factor of 3

Note: $2 \log_2 3 = 3$.

Runtime: $O(n \log_2 3)$.

How?

Next time (read ahead!!!)
Multiplication

Multiplication: $O(n^2)$.  
Is the best possible?  
Every digit in $x$ must multiply every digit in $y$ at least once!

Is this the best possible?  
(a) Yes.  
(b) No.  
No.  
We can do better!  
What?!!!  
Really!  
What does python do?  
Let's see.  
Runtime:  
2 times as large increases by a factor of 3  
Note: $2 \log_2 3 = 3$.  
Runtime: $O(n \log_2 3)$.  
How?  
Next time (read ahead!!!)
Multiplication

Multiplication: $O(n^2)$. Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once! $\Theta(n^2)$ such pairs.
Multiplication

Multiplication: $O(n^2)$.

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!
$\Theta(n^2)$ such pairs.

Is this the best possible?
**Multiplication**

Multiplication: $O(n^2)$.

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!

$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes.

(b) No.
Multiplication

Multiplication: $O(n^2)$.

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!

$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes.

(b) No.

No.
Multiplication

Multiplication: $O(n^2)$.

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!

$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes.

(b) No.

No. We can do better!
Multiplication

Multiplication: $O(n^2)$.

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!

$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes.

(b) No.

No. We can do better!
What ?!?!
Multiplication

Multiplication: \( O(n^2) \).

Is the best possible?

Every digit in \( x \) must multiply every digit in \( y \) at least once!
\( \Theta(n^2) \) such pairs.

Is this the best possible?

(a) Yes.

(b) No.

No. We can do better!
What ?!?!?
Really!
Multiplication

Multiplication: $O(n^2)$.

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!

$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes.

(b) No.

No. We can do better!
What ?!?! Really!
Multiplication

Multiplication: $O(n^2)$.  
Is the best possible?  

Every digit in $x$ must multiply every digit in $y$ at least once!  
$\Theta(n^2)$ such pairs.  

Is this the best possible?  

(a) Yes.  
(b) No.  

No. We can do better!  
What ?!?!!  
Really!  

What does python do?
Multiplication

Multiplication: $O(n^2)$.

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!

$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes.

(b) No.

No. We can do better!

What ?!?!

Really!

What does python do? Let’s see.
Multiplication

Multiplication: $O(n^2)$. 

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!  
$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes.

(b) No.

No. We can do better!

What ?!?!

Really!

What does python do? Let’s see.

Runtime:

2 times as large increases by a factor of 3
Multiplication

Multiplication: $O(n^2)$.

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!

$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes.

(b) No.

No. We can do better!
What ?!?!?
Really!

What does python do? Let’s see.

Runtime:

2 times as large increases by a factor of 3

Note: $2^{\log_2 3} = 3$. 
Multiplication

Multiplication: $O(n^2)$.

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!

$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes.

(b) No.

No. We can do better!

What ?!?!?

Really!

What does python do? Let’s see.

Runtime:

- 2 times as large increases by a factor of 3
- Note: $2^{\log_2 3} = 3$.

Runtime: $O(n^{\log_2 3})$. 
Multiplication

Multiplication: $O(n^2)$. 

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!

$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes.
(b) No.

No. We can do better!
What ?!?!?
Really!

What does python do? Let’s see.

Runtime:
  2 times as large increases by a factor of 3
Note: $2^{\log_2 3} = 3$.

Runtime: $O(n^{\log_2 3})$.
How?
Multiplication

Multiplication: $O(n^2)$.

Is the best possible?

Every digit in $x$ must multiply every digit in $y$ at least once!

$\Theta(n^2)$ such pairs.

Is this the best possible?

(a) Yes.

(b) No.

No. We can do better!

What ?!?!?

Really!

What does python do? Let’s see.

Runtime:

2 times as large increases by a factor of 3

Note: $2^\log_2 3 = 3$.

Runtime: $O(n^{\log_2 3})$.

How? Next time (read ahead!!!)
Quick Thoughts.

Big (historic) idea: representation as digits or bits.
Quick Thoughts.

Big (historic) idea: representation as digits or bits.
Algorithms for addition/multiplication/fibonacci numbers.
Quick Thoughts.

Big (historic) idea: representation as digits or bits.
Algorithms for addition/multiplication/fibonacci numbers.
Complexity or runtimes in terms of size of representation.
Quick Thoughts.

Big (historic) idea: representation as digits or bits.
Algorithms for addition/multiplication/fibonacci numbers.
Complexity or runtimes in terms of size of representation.
Asymptotic analysis.