CS 170: Algorithms

CS 170: Algorithms

Static Course Webpage. (cs170.org)

CS 170: Algorithms

Static Course Webpage. (cs170.org)

... (Staff)

CS 170: Algorithms

Static Course Webpage. (cs170.org)

... (Staff)

Will mostly use piazza. (Link.)

CS 170: Algorithms

Static Course Webpage. (cs170.org)

... (Staff)

Will mostly use piazza. (Link.)

Homeworks will be turned on gradescope. Latex ok, scanning handwritten stuff ok.

CS 170: Algorithms

Static Course Webpage. (cs170.org)

... (Staff)

Will mostly use piazza. (Link.)

Homeworks will be turned on gradescope. Latex ok, scanning handwritten stuff ok.

A puzzle.

Does a list have a cyle?

A puzzle.

Does a list have a cyle?

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

A puzzle.

Does a list have a cyle?

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

Mark first node.

A puzzle.

Does a list have a cyle?

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 cyle?

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 cyle?

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 cyle?

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 cyle?

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 cyle?

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 cyle?

Access to list is a pointer to the "first element."

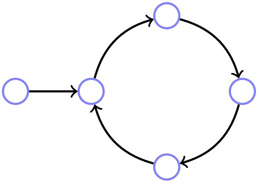
Mark first node.
While next cell not marked, go to next cell.

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 cyle?

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

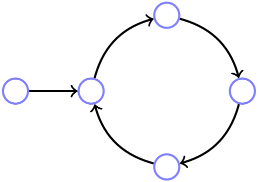
Mark first node.
While next cell not marked, go to next cell.

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 cyle?

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

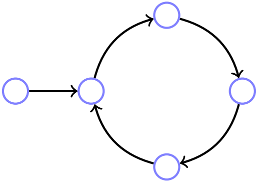
Mark first node.
While next cell not marked, go to next cell.

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 cyle?

Access to list is a pointer to the "first element."

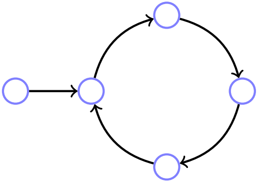
Mark first node.
While next cell not marked, go to next cell.

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 cyle?

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

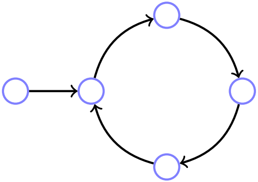
Mark first node.
While next cell not marked, go to next cell.

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 cyle?

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

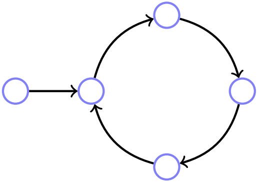
Mark first node.
While next cell not marked, go to next cell.

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 cyle?

Does a list have a cyle?

Two ptrs:

Does a list have a cyle?

Two ptrs: Step: advance ptr 1 twice,

Does a list have a cyle?

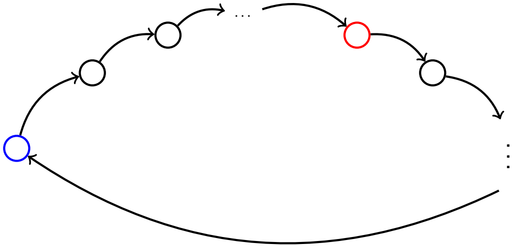
Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once.

Does a list have a cyle?

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

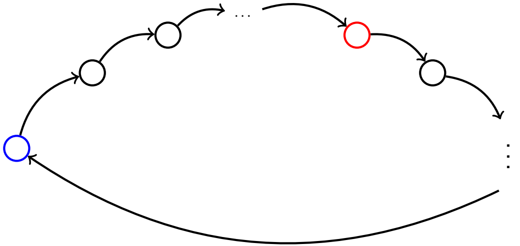
Does a list have a cyle?

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



Does a list have a cyle?

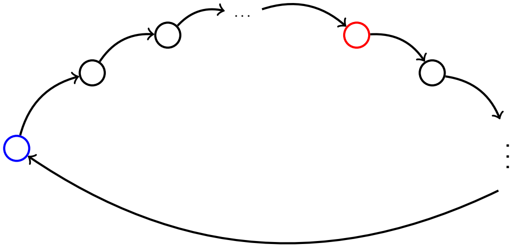
Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once.
If ever at the same place, report cycle.



Correctness:

Does a list have a cyle?

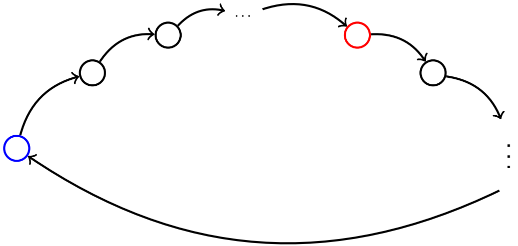
Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once.
If ever at the same place, report cycle.



Correctness:
If no cycle, slow pointer never catches fast one.

Does a list have a cyle?

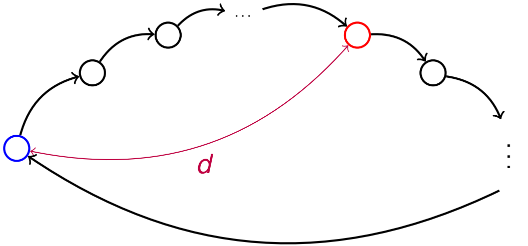
Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once.
If ever at the same place, report cycle.



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 cyle?

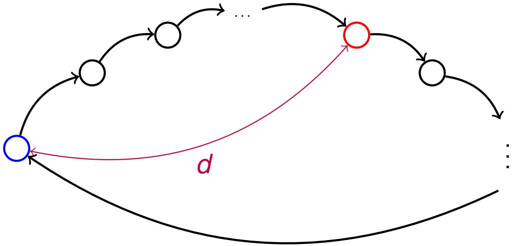
Two ptrs: Step: advance ptr 1 twice, advance ptr 2 once.
If ever at the same place, report cycle.



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 cyle?

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

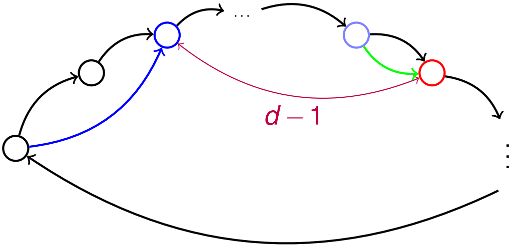


Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.
d - distance from fast ptr to slow ptr.

d decreases every step.

Does a list have a cyle?

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

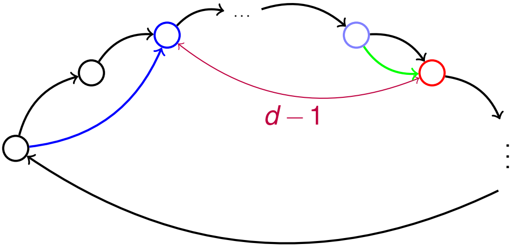


Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.
d - distance from fast ptr to slow ptr.

d decreases every step.

Does a list have a cyle?

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



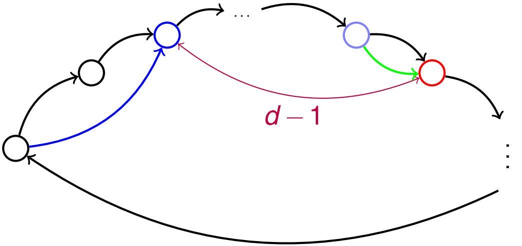
Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.
d - distance from fast ptr to slow ptr.

d decreases every step.

Runtime: n steps to cycle

Does a list have a cyle?

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



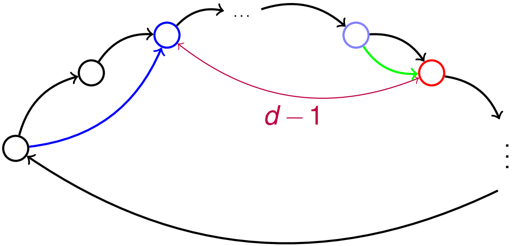
Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.
d - distance from fast ptr to slow ptr.

d decreases every step.

Runtime: n steps to cycle n steps to catch up.

Does a list have a cyle?

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



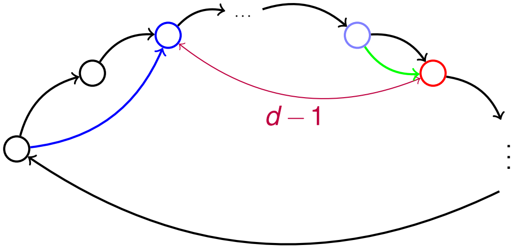
Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.
d - distance from fast ptr to slow ptr.

d decreases every step.

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

Does a list have a cyle?

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



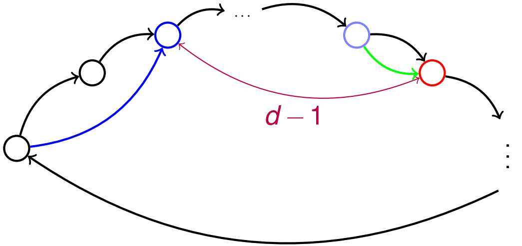
Correctness:
If no cycle, slow pointer never catches fast one.
If cycle, both pointers will enter cycle at some time.
d - distance from fast ptr to slow ptr.

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 cyle?

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...

Puzzles ..

Solutions

..are Algorithms...

which...

Puzzles ..

Solutions

..are Algorithms...

which...

..are correct...

Puzzles ..

Solutions

..are Algorithms...

which...

..are correct...and (in this class) efficient.

Puzzles ..

Solutions

..are Algorithms...

which...

..are correct...and (in this class) efficient.

Is this a useful process?

Algorithms for the Human Genome Project

Reconstruct DNA...

Algorithms for the Human Genome Project

Reconstruct DNA...

ACTGAAACTGAGTAGATA....

Algorithms for the Human Genome Project

Reconstruct DNA...

ACTGAAACTGAGTAGATA....

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

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.

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

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?

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
```

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
```

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
```

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
```

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
             AGATA
```

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
             AGATA
-------------------
ACTGAAACTGAGTAGATA
```

Page Rank.

Problem: What is good on the web?

Page Rank.

Problem: What is good on the web?

Website that pays search engine most?

Page Rank.

Problem: What is good on the web?

Website that pays search engine most? Goto.com.

Page Rank.

Problem: What is good on the web?

Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page):

Page Rank.

Problem: What is good on the web?

Website that pays search engine most? Goto.com.

Random Surfer Model (Brin-Page): Follow link, follow link,

Page Rank.

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. ε).

Page Rank.

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. ε).

Popular pages are desirable pages.

Page Rank.

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. ε).

Popular pages are desirable pages.

PageRank = popularity for random surfer.

Page Rank.

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. ε).

Popular pages are desirable pages.

PageRank = popularity for random surfer.

Sort search results by PageRank!

Page Rank.

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. ε).

Popular pages are desirable pages.

PageRank = popularity for random surfer.

Sort search results by PageRank!

Made us happier then.

Page Rank.

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. ε).

Popular pages are desirable pages.

PageRank = popularity for random surfer.

Sort search results by PageRank!

Made us happier then.

Google.

Page Rank.

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. ε).

Popular pages are desirable pages.

PageRank = popularity for random surfer.

Sort search results by PageRank!

Made us happier then.

Google.

Page Rank.

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. ε).

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.

Page Rank.

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. ε).

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

Algorithms...

Driving Directions
Airline Scheduling
Compiling
Compression
Cryptography
Optimization

Algorithms...

Driving Directions
Airline Scheduling
Compiling
Compression
Cryptography
Optimization
...(at Akamai, maintaining fastest, cheapest least loaded caches)

Algorithms...

Driving Directions
Airline Scheduling
Compiling
Compression
Cryptography
Optimization
...(at Akamai, maintaining fastest, cheapest least loaded caches)
.
.
.

Calculating: 300 BC through Middle Ages in Europe.

I – one

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

Calculating: 300 BC through Middle Ages in Europe.

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

Calculating: 300 BC through Middle Ages in Europe.

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

Calculating: 300 BC through Middle Ages in Europe.

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

Calculating: 300 BC through Middle Ages in Europe.

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

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

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

DCLXXI – 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

DCLXXI – five hundred plus a hundred plus fifty 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

DCLXXI – 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

DCLXXI – five hundred plus a hundred plus fifty plus ten plus ten..

MCDLXVIII – one thousand five hundred minus one hundred ....

Add them?

1448 + 671 =

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

DCLXXI – five hundred plus a hundred plus fifty plus ten plus ten..

MCDLXVIII – one thousand five hundred minus one hundred ....

Add them?

1448 + 671 = 2019

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

DCLXXI – five hundred plus a hundred plus fifty plus ten plus ten..

MCDLXVIII – one thousand five hundred minus one hundred ....

Add them?

1448 + 671 = 2019

671 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

DCLXXI – five hundred plus a hundred plus fifty plus ten plus ten..

MCDLXVIII – one thousand five hundred minus one hundred ....

Add them?

1448 + 671 = 2019

671 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

DCLXXI – five hundred plus a hundred plus fifty plus ten plus ten..

MCDLXVIII – one thousand five hundred minus one hundred ....

Add them?

1448 + 671 = 2019

671 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

DCLXXI – five hundred plus a hundred plus fifty plus ten plus ten..

MCDLXVIII – one thousand five hundred minus one hundred ....

Add them?

1448 + 671 = 2019

671 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 π..

Modern system.

From India, via Al Khwarizmi.

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

Algorithms!

Modern system.

From India, via Al Khwarizmi.

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

Algorithms!

Note:

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):

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).

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 “$\underline{\underline{\cdot\;\cdot\;\cdot}}$”

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 “$\underline{\underline{\cdots}}$”

Babylonions (base 60):

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 “$\underline{\underline{\cdots}}$”

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

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 "$\underline{\underline{\cdot\cdot\cdot}}$"

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 π..

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).
13 is “$\underline{\underline{\cdot\cdot\cdot}}$”

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 π..

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).
13 is "$\underline{\underline{\cdots}}$"

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 π..

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).
    13 is “$\underline{\underline{\cdots}}$”

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 “$\underline{\underline{\cdots}}$”

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 π..

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).
  13 is “$\underline{\underline{\cdot\cdot\cdot}}$”

Babylonions (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 π..

Algorithms!

Note:

Mayans (base 20): dots (ones) and underlines (fives).
13 is “$\underline{\underline{\cdots}}$”

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.

Writing to propagating..

Al Khwarizmi:

Writing to propagating..

Al Khwarizmi: Go west! Young decimal system!

Writing to propagating..

Al Khwarizmi: Go west! Young decimal system!

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

Writing to propagating..

Al Khwarizmi: Go west! Young decimal system!

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

Writing to propagating..

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.

Writing to propagating..

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.

Fibonacci numbers.

$F_0 = 0$, $F_1 = 1$.

Fibonacci numbers.

$F_0 = 0$, $F_1 = 1$.

$F_n = F_{n-1} + F_{n-2}$.

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)
```

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.

$F_0 = 0$, $F_1 = 1$.

$F_n = F_{n-1} + F_{n-2}$.

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

Correct? Implements definition!

Fibonacci numbers.

$F_0 = 0$, $F_1 = 1$.

$F_n = F_{n-1} + F_{n-2}$.

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

Correct? Implements definition!

Run time.

Fibonacci numbers.

$F_0 = 0$, $F_1 = 1$.

$F_n = F_{n-1} + F_{n-2}$.

```
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.

$F_0 = 0$, $F_1 = 1$.

$F_n = F_{n-1} + F_{n-2}$.

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

Correct? Implements definition!

Run time.

$T(n) = T(n-1) + T(n-2) + 2$

$T(n) \geq F_n$

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.

Better Algorithm.

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

Better Algorithm.

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

$O(n)$ operations!

Better Algorithm.

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

$O(n)$ operations! Maybe.

Better Algorithm.

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

$O(n)$ operations! Maybe.

Let's try it.

From demo: Size matters.

How many bits in the representation of F_n?

From demo: Size matters.

How many bits in the representation of F_n?

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

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?

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?

$\log_2 F_n$

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?

$\log_2 F_n \approx 0.6294n$

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 $F_{n-1} + F_{n-2}$?

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 $F_{n-1} + F_{n-2}$?

$O(n)$.

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.

cn^2 runtime.

Demo: polynomial.

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

cn^2 runtime.

$c(2n)^2$

Demo: polynomial.

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

cn^2 runtime.

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

Demo: polynomial.

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

cn^2 runtime.

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

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.

cn^2 runtime.

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

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

Scaling input by c grows runtime bound by c^k.

Demo: polynomial.

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

cn^2 runtime.

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

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

Scaling input by c grows runtime bound by c^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.

cn^2 runtime.

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

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

Scaling input by c grows runtime bound by c^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.

cn^2 runtime.

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

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

Scaling input by c grows runtime bound by c^k.

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

Not true for exponential algorithms. Squares runtime!

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.

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

Refreshing Asymptotic Notation.

Ignore constant factors.

Refreshing Asymptotic Notation.

Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$

Refreshing Asymptotic Notation.

Ignore constant factors.

$2n^2$ asymptotically same as $4n^2$

both are $O(n^2)$

Refreshing Asymptotic Notation.

Ignore constant factors.

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

$4 \log n$ asymptotically same as $100 \log n$

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 $O(\log n)$

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 $O(\log n)$

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 $O(\log n)$

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 $O(\log n)$

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 $O(\log n)$

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 $O(\log n)$

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 $O(\log n)$

Ignore smaller order terms.

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

Upper bound.
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 $O(\log n)$

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 $O(n^3)$.

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 $O(\log n)$

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 $O(\log n)$

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.

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

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

More asymptotic notation.

Ω notation.

More asymptotic notation.

Ω notation.

A “lower bound”.

More asymptotic notation.

Ω notation.

A “lower bound”.

$2n^2$ is $\Omega(n^2)$ ...

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 $\Omega(n^2)$ ...and $\Omega(n)$...

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 $g(n) \geq cf(n)$

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: $O(n)$

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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
	0	4	4	6	9	4	4	8	0

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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
1	0	4	4	6	9	4	4	8	0

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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

Time: $O(n)$

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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

Time: $O(n)$

Can we do better?

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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
1	0	4	4	6	9	4	4	8	0

Time: $O(n)$

Can we do better?

Need to look at the numbers to add them...

Al Khwarizmi: Arithmetic.

Addition: $O(n)$

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

Time: $O(n)$

Can we do better?

Need to look at the numbers to add them... optimal.

More Al Khwarizmi's: algorithms.

Addition: $O(n)$

More Al Khwarizmi's: algorithms.

Addition: $O(n)$

Multiplication:

$$
\begin{array}{rccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
\end{array}
$$

More Al Khwarizmi's: algorithms.

Addition: $O(n)$

Multiplication:

$$
\begin{array}{rccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9
\end{array}
$$

More Al Khwarizmi's: algorithms.

Addition: $O(n)$

Multiplication:

$$
\begin{array}{rrrrrrrrrrr}
 & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
 & \times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
 & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
9 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 &
\end{array}
$$

More Al Khwarizmi's: algorithms.

Addition: $O(n)$

Multiplication:

$$
\begin{array}{ccccccccccc}
 & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
 & \times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
 & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
9 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & \\
. & . & . & . & . & . & . & . & . & . & .
\end{array}
$$

More Al Khwarizmi's: algorithms.

Addition: $O(n)$

Multiplication:

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

More Al Khwarizmi's: algorithms.

Addition: $O(n)$

Multiplication:

$$
\begin{array}{ccccccccccccc}
 & & & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
 & & & \times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
 & & & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
 & & 9 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & \\
 & & . & . & . & . & . & . & . & . & . & . & . \\
 & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}
\qquad \updownarrow\, n
$$

More Al Khwarizmi's: algorithms.

Addition: $O(n)$

Multiplication:

$$
\begin{array}{ccccccccccccc}
 & & & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
 & & & \times & 9 & 2 & 1 & 2 & 3 & 7 & 6 & 9 & 1 \\
\hline
 & & & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
 & & 9 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & \\
 & & . & . & . & . & . & . & . & . & . & . & . \\
 & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}
\qquad \updownarrow n
$$

Time: $O(n^2)$

Multiplication

Multiplication: $O(n^2)$.

Multiplication

Multiplication: $O(n^2)$.

Is 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!

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!

Quick Thoughts.

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

Quick Thoughts.

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

Complexity or runtimes in terms of size of representation.

Quick Thoughts.

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

Complexity or runtimes in terms of size of representation.

Asymptotic analysis.