Is there a faster method?

Venkatesh Raman

Theoretical Computer Science group
The Institute of Mathematical Sciences
Chennai

November 6, 2014

Outline

Computers, Computing, Science

Computers, Computing, Science

- What do we use Computers for?

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos
  - connect with friends (facebook, ...)

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos
  - connect with friends (facebook, ...)
  - search for something

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos
  - connect with friends (facebook, ...)
  - search for something
  - buy/sell/pay

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos
  - connect with friends (facebook, ...)
  - search for something
  - buy/sell/pay
  - solve our computational problems

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos
  - connect with friends (facebook, ...)
  - search for something
  - buy/sell/pay
  - solve our computational problems
- We want it to solve our problems

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos
  - connect with friends (facebook, ...)
  - search for something
  - buy/sell/pay
  - solve our computational problems
- We want it to solve our problems fast

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos
  - connect with friends (facebook, ...)
  - search for something
  - buy/sell/pay
  - solve our computational problems
- We want it to solve our problems fast
- How fast is good enough? Is there a limit?

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos
  - connect with friends (facebook, ...)
  - search for something
  - buy/sell/pay
  - solve our computational problems
- We want it to solve our problems fast
- How fast is good enough? Is there a limit?
- Clearly the time is a function of the *size* and the *complexity* of the problem (apart from the operating system, processor speed, the internet speed, the compiler speed etc)

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos
  - connect with friends (facebook, ...)
  - search for something
  - buy/sell/pay
  - solve our computational problems
- We want it to solve our problems fast
- How fast is good enough? Is there a limit?
- Clearly the time is a function of the *size* and the *complexity* of the problem (apart from the operating system, processor speed, the internet speed, the compiler speed etc)
- How fast (as a function of the size of the input) is *feasible*?

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos
  - connect with friends (facebook, ...)
  - search for something
  - buy/sell/pay
  - solve our computational problems
- We want it to solve our problems fast
- How fast is good enough? Is there a limit?
- Clearly the time is a function of the *size* and the *complexity* of the problem (apart from the operating system, processor speed, the internet speed, the compiler speed etc)
- How fast (as a function of the size of the input) is *feasible*?
- How do we measure the time as a function of the size?

Computers, Computing, Science

- What do we use Computers for?
  - see mails, videos
  - connect with friends (facebook, ...)
  - search for something
  - buy/sell/pay
  - solve our computational problems
- We want it to solve our problems fast
- How fast is good enough? Is there a limit?
- Clearly the time is a function of the *size* and the *complexity* of the problem (apart from the operating system, processor speed, the internet speed, the compiler speed etc)
- How fast (as a function of the size of the input) is *feasible*?
- How do we measure the time as a function of the size?

If we can't measure, we can't compare, we can't improve

What this talk is about?

What this talk is about?

- Discuss ONE way to measure problem complexity through Algorithms in a way independent of computers, systems, processor, compiler, ...

What this talk is about?

- Discuss ONE way to measure problem complexity through Algorithms in a way independent of computers, systems, processor, compiler, ...
- Measure the time for some well known problems/algorithms (a quick tour)

What this talk is about?

- Discuss ONE way to measure problem complexity through Algorithms in a way independent of computers, systems, processor, compiler, ...
- Measure the time for some well known problems/algorithms (a quick tour)
- Discuss limitations to get faster solutions

Outline

Find my number

Find my number

- I think of a number between 1 and 1024, can you find it

Find my number

- I think of a number between 1 and 1024, can you find it by asking me at most 10 questions?

Find my number

- I think of a number between 1 and 1024, can you find it by asking me at most 10 questions?
  at most 10 questions requiring *YES or NO answers*

Find my number

- I think of a number between 1 and 1024, can you find it by asking me at most 10 questions?
  at most 10 questions requiring *YES or NO answers*

Find my number

- I think of a number between 1 and 1024, can you find it by asking me at most 10 questions?
  at most 10 questions requiring *YES or NO answers*

- What if the number is between 1 and N?

Find my number

- I think of a number between 1 and 1024, can you find it by asking me at most 10 questions?
  at most 10 questions requiring *YES or NO answers*

- What if the number is between 1 and N? $\log_2 N$ questions.

Find my number

- I think of a number between 1 and 1024, can you find it by asking me at most 10 questions?
  at most 10 questions requiring *YES or NO answers*

- What if the number is between 1 and N? $\log_2 N$ questions.
- Can we do with fewer questions? —

Find my number

- I think of a number between 1 and 1024, can you find it by asking me at most 10 questions?
  at most 10 questions requiring *YES or NO answers*

- What if the number is between 1 and N? $\log_2 N$ questions.
- Can we do with fewer questions? — NO

Find my number and Computing – What is the connection?

Find my number and Computing – What is the connection?

- Much of our computer usage is about searching

Find my number and Computing – What is the connection?

- Much of our computer usage is about searching
- Searching a database (google, train reservation, ...)

Find my number and Computing – What is the connection?

- Much of our computer usage is about searching
- Searching a database (google, train reservation, ...)
- Searching for a solution among many possible solutions

Find my number and Computing – What is the connection?

- Much of our computer usage is about searching
- Searching a database (google, train reservation, ...)
- Searching for a solution among many possible solutions
- $\log N$ is a very slow growing function.

Find my number and Computing – What is the connection?

- Much of our computer usage is about searching
- Searching a database (google, train reservation, ...)
- Searching for a solution among many possible solutions
- $\log N$ is a very slow growing function. For example, for $N = 10^{30}$, $\log_2 N < 120$, and so logarithmic solutions are very efficient

Outline

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Example: GCD(63, 105)
Divisors of $63 = 1, 3, 7, 9, 21, 63$

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Example: GCD(63, 105)

Divisors of $63 = 1, 3, 7, 9, 21, 63$

Divisors of $105 = 1, 3, 5, 7, 15, 21, 35, 105$

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Example: GCD(63, 105)

Divisors of $63 = 1, 3, 7, 9, 21, 63$
Divisors of $105 = 1, 3, 5, 7, 15, 21, 35, 105$
$\text{GCD} = 21$

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Example: GCD(63, 105)
Divisors of $63 = 1, 3, 7, 9, 21, 63$
Divisors of $105 = 1, 3, 5, 7, 15, 21, 35, 105$
GCD $= 21$

- How many operations does this algorithm perform?

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Example: GCD(63, 105)
Divisors of $63 = 1, 3, 7, 9, 21, 63$
Divisors of $105 = 1, 3, 5, 7, 15, 21, 35, 105$
$\text{GCD} = 21$

- How many operations does this algorithm perform?
- How many divisions?

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Example: GCD(63, 105)
Divisors of $63 = 1, 3, 7, 9, 21, 63$
Divisors of $105 = 1, 3, 5, 7, 15, 21, 35, 105$
$\text{GCD} = 21$

- How many operations does this algorithm perform?
- How many divisions?
- $63 + 105$?

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Example: GCD(63, 105)
Divisors of $63 = 1, 3, 7, 9, 21, 63$
Divisors of $105 = 1, 3, 5, 7, 15, 21, 35, 105$
GCD $= 21$

- How many operations does this algorithm perform?
- How many divisions?
- $63 + 105$? $61 + 103$?

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Example: GCD(63, 105)
Divisors of $63 = 1, 3, 7, 9, 21, 63$
Divisors of $105 = 1, 3, 5, 7, 15, 21, 35, 105$
$\text{GCD} = 21$

- How many operations does this algorithm perform?
- How many divisions?
- $63 + 105$? $61 + 103$? $7 + 10$?

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Example: GCD(63, 105)
Divisors of $63 = 1, 3, 7, 9, 21, 63$
Divisors of $105 = 1, 3, 5, 7, 15, 21, 35, 105$
GCD $= 21$

- How many operations does this algorithm perform?
- How many divisions?
- $63 + 105$? $61 + 103$? $7 + 10$?
- If the numbers are m and n, then roughly $\sqrt{m} + \sqrt{n}$.

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Example: GCD(63, 105)

Divisors of $63 = 1, 3, 7, 9, 21, 63$

Divisors of $105 = 1, 3, 5, 7, 15, 21, 35, 105$

GCD $= 21$

- How many operations does this algorithm perform?
- How many divisions?
- $63 + 105$? $61 + 103$? $7 + 10$?
- If the numbers are m and n, then roughly $\sqrt{m} + \sqrt{n}$.
- Is there a faster method? –

Algorithm 1 for finding GCD of two numbers:

GCD of m and n is the **greatest common divisor** of m and n.

1. List out the divisors of both numbers
2. Identify the common ones and pick the largest one.

Example: GCD(63, 105)
Divisors of $63 = 1, 3, 7, 9, 21, 63$
Divisors of $105 = 1, 3, 5, 7, 15, 21, 35, 105$
$\text{GCD} = 21$

- How many operations does this algorithm perform?
- How many divisions?
- $63 + 105$? $61 + 103$? $7 + 10$?
- If the numbers are m and n, then roughly $\sqrt{m} + \sqrt{n}$.
- Is there a faster method? – YES.

Euclid's Algorithm (300 BC): given $m, n, m \leq n$

- While m does not divide n $\{$
  $r = n \mod m$
  $n = m$
  $m = r$ $\}$
- Output m.

Euclid's Algorithm (300 BC): given $m, n, m \leq n$

- While m does not divide n $\{$
  $r = n \mod m$
  $n = m$
  $m = r$ $\}$
- Output m.

Example: 63, 105

Euclid's Algorithm (300 BC): given $m, n, m \leq n$

- While m does not divide n $\{$
  $r = n \mod m$
  $n = m$
  $m = r$ $\}$
- Output m.

Example: 63, 105

- $r = 42, m = 42, n = 63$

Euclid's Algorithm (300 BC): given $m, n, m \leq n$

- While m does not divide n {
  $r = n \mod m$
  $n = m$
  $m = r$ }
- Output m.

Example: 63, 105

- $r = 42, m = 42, n = 63$
- $r = 21, m = 21, n = 42$

Euclid's Algorithm (300 BC): given $m, n, m \leq n$

- While m does not divide n $\{$
  $r = n \mod m$
  $n = m$
  $m = r$ $\}$
- Output m.

Example: 63, 105

- $r = 42, m = 42, n = 63$
- $r = 21, m = 21, n = 42$

Output 21.

Euclid's Algorithm (300 BC): given $m, n, m \leq n$

- While m does not divide n $\{$
  $r = n \mod m$
  $n = m$
  $m = r$ $\}$
- Output m.

Example: 63, 105

- $r = 42, m = 42, n = 63$
- $r = 21, m = 21, n = 42$

Output 21.
How many divisions?

Euclid's Algorithm (300 BC): given $m, n, m \leq n$

- While m does not divide n $\{$

  $r = n \mod m$

  $n = m$

  $m = r$ $\}$

- Output m.

Example: 63, 105

- $r = 42, m = 42, n = 63$
- $r = 21, m = 21, n = 42$

Output 21.
How many divisions?
Three (recall method 1 did about 17 divisions)

Another example comparing the two algorithms

GCD(1071, 17850)

- Method 1

Another example comparing the two algorithms

GCD(1071, 17850)

- Method 1
  - Divisors of 1071 = 1, 3, 7, 9, 17, 21, 51, 63, 119, 357, 1071

Another example comparing the two algorithms

GCD(1071, 17850)

- Method 1
  - Divisors of 1071 = 1, 3, 7, 9, 17, 21, 51, 63, 119, 357, 1071
  - Divisors of 17850 = 1, 2, 3, 5, 6, 7, 10, 14, 15, 17, 21, 30, 35, ... 119, ... 357, ... 17850

Another example comparing the two algorithms

GCD(1071, 17850)

- Method 1
  - Divisors of 1071 = 1, 3, 7, 9, 17, 21, 51, 63, 119, 357, 1071
  - Divisors of 17850 = 1, 2, 3, 5, 6, 7, 10, 14, 15, 17, 21, 30, 35, ... 119, ... 357, ... 17850
  - GCD = 357

Another example comparing the two algorithms

GCD(1071, 17850)

- Method 1
  - Divisors of 1071 = 1, 3, 7, 9, 17, 21, 51, 63, 119, 357, 1071
  - Divisors of 17850 = 1, 2, 3, 5, 6, 7, 10, 14, 15, 17, 21, 30, 35, ... 119, ... 357, ... 17850
  - GCD = 357

Makes more than 130 divisions.

Another example comparing the two algorithms

GCD(1071, 17850)

- Method 1
  - Divisors of 1071 = 1, 3, 7, 9, 17, 21, 51, 63, 119, 357, 1071
  - Divisors of 17850 = 1, 2, 3, 5, 6, 7, 10, 14, 15, 17, 21, 30, 35, ... 119, ... 357, ... 17850
  - GCD = 357

  Makes more than 130 divisions.
- Euclid's Algorithms execution:
  - $m = 1071, n = 17850$

Another example comparing the two algorithms

GCD(1071, 17850)

- Method 1
  - Divisors of 1071 = 1, 3, 7, 9, 17, 21, 51, 63, 119, 357, 1071
  - Divisors of 17850 = 1, 2, 3, 5, 6, 7, 10, 14, 15, 17, 21, 30, 35, ... 119, ... 357, ... 17850
  - GCD = 357

  Makes more than 130 divisions.
- Euclid's Algorithms execution:
  - $m = 1071, n = 17850$
  - $m = 714, n = 1071$

Another example comparing the two algorithms

GCD(1071, 17850)

- Method 1
  - Divisors of 1071 = 1, 3, 7, 9, 17, 21, 51, 63, 119, 357, 1071
  - Divisors of 17850 = 1, 2, 3, 5, 6, 7, 10, 14, 15, 17, 21, 30, 35, ... 119, ... 357, ... 17850
  - GCD = 357

  Makes more than 130 divisions.
- Euclid's Algorithms execution:
  - $m = 1071, n = 17850$
  - $m = 714, n = 1071$
  - $m = 357, n = 714$

Another example comparing the two algorithms

GCD(1071, 17850)

- Method 1
  - Divisors of 1071 = 1, 3, 7, 9, 17, 21, 51, 63, 119, 357, 1071
  - Divisors of 17850 = 1, 2, 3, 5, 6, 7, 10, 14, 15, 17, 21, 30, 35, ... 119, ... 357, ... 17850
  - GCD = 357

  Makes more than 130 divisions.
- Euclid's Algorithms execution:
  - $m = 1071, n = 17850$
  - $m = 714, n = 1071$
  - $m = 357, n = 714$
  - Output 357.

Another example comparing the two algorithms

GCD(1071, 17850)

- Method 1
  - Divisors of 1071 = 1, 3, 7, 9, 17, 21, 51, 63, 119, 357, 1071
  - Divisors of 17850 = 1, 2, 3, 5, 6, 7, 10, 14, 15, 17, 21, 30, 35, ... 119, ... 357, ... 17850
  - GCD = 357

  Makes more than 130 divisions.
- Euclid's Algorithms execution:
  - $m = 1071, n = 17850$
  - $m = 714, n = 1071$
  - $m = 357, n = 714$
  - Output 357.
    Makes 4 divisions

Another example comparing the two algorithms

GCD(1071, 17850)

- Method 1
  - Divisors of 1071 = 1, 3, 7, 9, 17, 21, 51, 63, 119, 357, 1071
  - Divisors of 17850 = 1, 2, 3, 5, 6, 7, 10, 14, 15, 17, 21, 30, 35, ... 119, ... 357, ... 17850
  - GCD = 357

  Makes more than 130 divisions.

- Euclid's Algorithms execution:
  - $m = 1071, n = 17850$
  - $m = 714, n = 1071$
  - $m = 357, n = 714$
  - Output 357.
    Makes 4 divisions

Questions: Why is Euclid's algorithm correct? Will it always terminate? Why is it fast? How fast?

Analysis of Euclid's Algorithm

Algorithm:

While m does not divide n $\{\ r = n \mod m; n = m; m = r\ \}$
Output m.

Analysis of Euclid's Algorithm

Algorithm:
While m does not divide n $\{\ r = n \mod m; n = m; m = r\ \}$
Output m.

- **Correctness**: $GCD(m, n) = GCD(n \mod m, m)$

Analysis of Euclid's Algorithm

Algorithm:
While m does not divide n $\{\ r = n \mod m; n = m; m = r\ \}$
Output m.

- **Correctness**: $GCD(m, n) = GCD(n \mod m, m)$
  Proof: If d divides m and n, d divides $n - m$.

Analysis of Euclid's Algorithm

Algorithm:
While m does not divide n $\{\ r = n \mod m; n = m; m = r\ \}$
Output m.

- **Correctness**: $GCD(m, n) = GCD(n \mod m, m)$
  Proof: If d divides m and n, d divides $n - m$.
- **Termination**: After the first step, the new m, let's call it
  - m' becomes $n \mod m < m$ and the new n, let's call it
  - $n' = m < n$

  and so the pair of numbers progressively decrease.

Analysis of Euclid's Algorithm

Algorithm:
While m does not divide n { $r = n \mod m; n = m; m = r$ }
Output m.

- **Correctness**: $GCD(m, n) = GCD(n \mod m, m)$
  Proof: If d divides m and n, d divides $n - m$.
- **Termination**: After the first step, the new m, let's call it
  - m' becomes $n \mod m < m$ and the new n, let's call it
  - $n' = m < n$

  and so the pair of numbers progressively decrease.
- **Time (# of divisions)**: How much do these numbers decrease by?.

Analysis of Euclid's Algorithm continued

- One can show that $m' + n' \leq 2(m+n)/3$. So, after the first step,
$$(m' + n') \leq (2/3)(m+n).$$
After the second step,

Analysis of Euclid's Algorithm continued

- One can show that $m' + n' \leq 2(m+n)/3$. So, after the first step,
$$(m' + n') \leq (2/3)(m+n).$$
After the second step,
$$(m' + n') \leq (2/3)^2(m+n)$$
and so after the k-th step

Analysis of Euclid's Algorithm continued

- One can show that $m' + n' \leq 2(m+n)/3$. So, after the first step,
$$(m' + n') \leq (2/3)(m+n).$$
After the second step,
$$(m' + n') \leq (2/3)^2(m+n)$$
and so after the k-th step
$$(m' + n') \leq (2/3)^k(m+n)$$
and so after the $(\log_{3/2}(m+n))$-th step $(m' + n')$ become less than 1 if the loop hasn't terminated already.

Analysis of Euclid's Algorithm continued

- One can show that $m' + n' \leq 2(m+n)/3$. So, after the first step,

$$(m' + n') \leq (2/3)(m+n).$$

After the second step,

$$(m' + n') \leq (2/3)^2(m+n)$$

and so after the k-th step

$$(m' + n') \leq (2/3)^k(m+n)$$

and so after the $(\log_{3/2}(m+n))$-th step $(m' + n')$ become less than 1 if the loop hasn't terminated already.

- So Euclid's algorithm performs $(\log_{3/2}(m+n))$ divisions while the naive algorithm makes $\sqrt{m} + \sqrt{n}$ divisions.

Analysis of Euclid's Algorithm continued

- One can show that $m' + n' \leq 2(m+n)/3$. So, after the first step,
$$(m' + n') \leq (2/3)(m+n).$$
After the second step,
$$(m' + n') \leq (2/3)^2(m+n)$$
and so after the k-th step
$$(m' + n') \leq (2/3)^k(m+n)$$
and so after the $(\log_{3/2}(m+n))$-th step $(m' + n')$ become less than 1 if the loop hasn't terminated already.
- So Euclid's algorithm performs $(\log_{3/2}(m+n))$ divisions while the naive algorithm makes $\sqrt{m} + \sqrt{n}$ divisions.
- For 10 digit numbers, *i.e.* when $m+n$ is about 10^{10}, Method 1 makes 10^5 divisions while Euclid makes about 60 divisions.

Analysis of Euclid's Algorithm continued

- One can show that $m' + n' \leq 2(m+n)/3$. So, after the first step,
$$(m' + n') \leq (2/3)(m+n).$$
After the second step,
$$(m' + n') \leq (2/3)^2(m+n)$$
and so after the k-th step
$$(m' + n') \leq (2/3)^k(m+n)$$
and so after the $(\log_{3/2}(m+n))$-th step $(m' + n')$ become less than 1 if the loop hasn't terminated already.
- So Euclid's algorithm performs $(\log_{3/2}(m+n))$ divisions while the naive algorithm makes $\sqrt{m} + \sqrt{n}$ divisions.
- For 10 digit numbers, $i.e.$ when $m+n$ is about 10^{10}, Method 1 makes 10^5 divisions while Euclid makes about 60 divisions. which can make a difference between 0.5 second and 0.000000006 second in modern computers.

Can we solve it faster?

NOT really because $\log m + \log n$ is actually the SIZE of the input (integers m and n).

Can we solve it faster?

NOT really because $\log m + \log n$ is actually the SIZE of the input (integers m and n).
Any algorithm must at least see its entire input.

Outline

Find maximum

Given a list L of n items, find the maximum using comparisons.

Find maximum

Given a list L of n items, find the maximum using comparisons.

- $Max = L[1]$
  for $i = 2$ to n
  if $L[i] > Max$, $Max := L[i]$
  Output Max

Find maximum

Given a list L of n items, find the maximum using comparisons.

- $Max = L[1]$
  for $i = 2$ to n
  if $L[i] > Max$, $Max := L[i]$
  Output Max
- Makes $n - 1$ comparisons.

Find maximum

Given a list L of n items, find the maximum using comparisons.

- $Max = L[1]$
  for $i = 2$ to n
  if $L[i] > Max$, $Max := L[i]$
  Output Max
- Makes $n - 1$ comparisons.
- Can we do better?

Find maximum

Given a list L of n items, find the maximum using comparisons.

- $Max = L[1]$
  for $i = 2$ to n
  if $L[i] > Max$, $Max := L[i]$
  Output Max
- Makes $n - 1$ comparisons.
- Can we do better? No

Outline

Arrange in increasing order

Given a list of n iterms, arrange them in increasing order.

Arrange in increasing order

Given a list of n iterms, arrange them in increasing order.

- By repeatedly finding the maximum, one can sort in about n^2 comparisons.

Arrange in increasing order

Given a list of n iterms, arrange them in increasing order.

- By repeatedly finding the maximum, one can sort in about n^2 comparisons.
- Can be improved to $O(n \log n)$ – a number of methods (Mergesort, Heapsort, ...)

Arrange in increasing order

Given a list of n iterms, arrange them in increasing order.

- By repeatedly finding the maximum, one can sort in about n^2 comparisons.
- Can be improved to $O(n \log n)$ – a number of methods (Mergesort, Heapsort, ...)
- Using a decision tree, one can show that this can not be (asymptotically) improved.

Outline

Multiplying two *nxn* matrices

Given two matrices A and B, each of dimension nxn, compute AB.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ & \end{bmatrix}$$

Multiplying two *nxn* matrices

Given two matrices A and B, each of dimension nxn, compute AB.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ & \end{bmatrix}$$

- Standard multiplication algorithm takes n^3 (scalar) multiplications.
  $(C[i,j] = \sum_{k=1}^{n} A[i,k] * B[k,j]$, there are n^2 product values, each requires about n multiplications)

Multiplying two *nxn* matrices

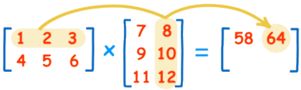
Given two matrices A and B, each of dimension nxn, compute AB.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ & \end{bmatrix}$$

- Standard multiplication algorithm takes n^3 (scalar) multiplications.
  ($C[i,j] = \sum_{k=1}^{n} A[i,k] * B[k,j]$, there are n^2 product values, each requires about n multiplications)
- Using Divide and Conquer and algebraic techniques, one can improve this to $n^{2.236}$.

Multiplying two *nxn* matrices

Given two matrices A and B, each of dimension *nxn*, compute AB.



- Standard multiplication algorithm takes n^3 (scalar) multiplications.
  ($C[i,j] = \sum_{k=1}^{n} A[i,k] * B[k,j]$, there are n^2 product values, each requires about n multiplications)
- Using Divide and Conquer and algebraic techniques, one can improve this to $n^{2.236}$.
- Can we solve it faster? We don't know, the search is on. Improving the bound is an open problem.

Outline

Multiplying two n digit numbers

Multiplying two n digit numbers

- Given two n digit numbers a and b, compute ab.

Multiplying two n digit numbers

- Given two n digit numbers a and b, compute ab.
- Standard multiplication algorithm takes n^2 (single digit) multiplications.

Multiplying two n digit numbers

- Given two n digit numbers a and b, compute ab.
- Standard multiplication algorithm takes n^2 (single digit) multiplications.
- **Example:**

$$\begin{array}{rl} 435876 & \times \\ 275683 & \\ \hline \end{array}$$

Multiplying two n digit numbers

- Given two n digit numbers a and b, compute ab.
- Standard multiplication algorithm takes n^2 (single digit) multiplications.
- **Example:**

$$
\begin{array}{rl}
435876 & \times \\
275683 & \\
\hline
\end{array}
$$

- Using Divide and Conquer and algebraic techniques, this can be further improved to $O(n \log n \log \log n)$.

Multiplying two n digit numbers

- Given two n digit numbers a and b, compute ab.
- Standard multiplication algorithm takes n^2 (single digit) multiplications.
- **Example:** $\begin{array}{lr} 435876 & \times \\ 275683 & \\ \hline \end{array}$
- Using Divide and Conquer and algebraic techniques, this can be further improved to $O(n \log n \log \log n)$. Improving this further is an open problem.

Divide and Conquer algorithm for integer multiplication

Divide and Conquer algorithm for integer multiplication

- An idea for a better method:

Divide and Conquer algorithm for integer multiplication

- An idea for a better method:
  $435876 = 435x10^3 + 876$
  $275683 = 275x10^3 + 683$

Divide and Conquer algorithm for integer multiplication

- An idea for a better method:
  $435876 = 435x10^3 + 876$
  $275683 = 275x10^3 + 683$
  The product $=$
  $(435x275)10^6 + (876x683) + 10^3(435x683 + 876x275)$.

Divide and Conquer algorithm for integer multiplication

- An idea for a better method:
  $435876 = 435x10^3 + 876$
  $275683 = 275x10^3 + 683$
  The product $=$
  $(435x275)10^6 + (876x683) + 10^3(435x683 + 876x275)$.
- In general, multiplying a pair of n digit numbers is broken into 4 multiplications of pairs of $n/2$ digit numbers.

Divide and Conquer algorithm for integer multiplication

- An idea for a better method:
  $435876 = 435x10^3 + 876$
  $275683 = 275x10^3 + 683$
  The product $=$
  $(435x275)10^6 + (876x683) + 10^3(435x683 + 876x275)$.
- In general, multiplying a pair of n digit numbers is broken into 4 multiplications of pairs of $n/2$ digit numbers. I.e.
  $T(n) = 4T(n/2) + O(n)$

Divide and Conquer algorithm for integer multiplication

- An idea for a better method:
  $435876 = 435x10^3 + 876$
  $275683 = 275x10^3 + 683$
  The product $=$
  $(435x275)10^6 + (876x683) + 10^3(435x683 + 876x275)$.
- In general, multiplying a pair of n digit numbers is broken into 4 multiplications of pairs of $n/2$ digit numbers. I.e.
  $T(n) = 4T(n/2) + O(n)$
- which unfortunately solves again to $T(n) = O(n^2)$.
  Here is an improved idea.

Divide and Conquer algorithm for integer multiplication

- An idea for a better method:
  $435876 = 435x10^3 + 876$
  $275683 = 275x10^3 + 683$
  The product $=$
  $(435x275)10^6 + (876x683) + 10^3(435x683 + 876x275)$.
- In general, multiplying a pair of n digit numbers is broken into 4 multiplications of pairs of $n/2$ digit numbers. I.e.
  $T(n) = 4T(n/2) + O(n)$
- which unfortunately solves again to $T(n) = O(n^2)$.
  Here is an improved idea.
- The product $= (435x275)x10^6 + (876x683) +$
  $10^3((435 + 876)(275 + 683) - (435x275) - (876x683))$

Divide and Conquer algorithm for integer multiplication

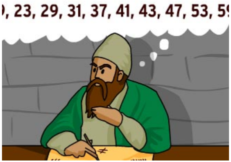
- An idea for a better method:
  $435876 = 435x10^3 + 876$
  $275683 = 275x10^3 + 683$
  The product $=$
  $(435x275)10^6 + (876x683) + 10^3(435x683 + 876x275)$.
- In general, multiplying a pair of n digit numbers is broken into 4 multiplications of pairs of $n/2$ digit numbers. I.e.
  $T(n) = 4T(n/2) + O(n)$
- which unfortunately solves again to $T(n) = O(n^2)$.
  Here is an improved idea.
- The product $= (435x275)x10^6 + (876x683) +$
  $10^3((435 + 876)(275 + 683) - (435x275) - (876x683))$
- I.e. $T(n) = 3T(n/2) + O(n)$

Divide and Conquer algorithm for integer multiplication

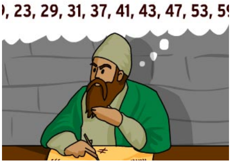
- An idea for a better method:
  $435876 = 435x10^3 + 876$
  $275683 = 275x10^3 + 683$
  The product $=$
  $(435x275)10^6 + (876x683) + 10^3(435x683 + 876x275)$.
- In general, multiplying a pair of n digit numbers is broken into 4 multiplications of pairs of $n/2$ digit numbers. I.e. $T(n) = 4T(n/2) + O(n)$
- which unfortunately solves again to $T(n) = O(n^2)$. Here is an improved idea.
- The product $= (435x275)x10^6 + (876x683) + 10^3((435 + 876)(275 + 683) - (435x275) - (876x683))$
- I.e. $T(n) = 3T(n/2) + O(n)$
- which results in a HUGE improvement ($O(n^{1.7})$).

Outline

Checking whether a number is prime

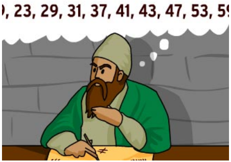


Checking whether a number is prime



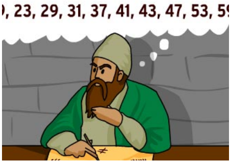
- For $i = 2$ to $\sqrt{n}$
  check if i divides n, if so, it is not prime,
  else (no factor till $\sqrt{n}$), it is prime.

Checking whether a number is prime



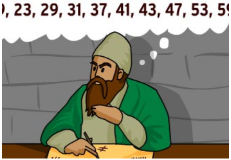
- For $i = 2$ to $\sqrt{n}$
  check if i divides n, if so, it is not prime,
  else (no factor till $\sqrt{n}$), it is prime.
- Takes $\sqrt{n}$ divisions.

Checking whether a number is prime



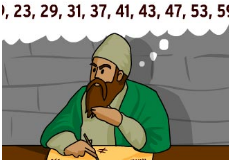
- For $i = 2$ to $\sqrt{n}$
  check if i divides n, if so, it is not prime,
  else (no factor till $\sqrt{n}$), it is prime.
- Takes $\sqrt{n}$ divisions.
- Recently (about 10 years ago), this has been improved to $O(\log^5 n)$ divisions using techniques from algebraic number theory.

Checking whether a number is prime



- For $i = 2$ to $\sqrt{n}$
  check if i divides n, if so, it is not prime,
  else (no factor till $\sqrt{n}$), it is prime.
- Takes $\sqrt{n}$ divisions.
- Recently (about 10 years ago), this has been improved to $O(\log^5 n)$ divisions using techniques from algebraic number theory. Major recent contribution from India (Agrawal, Kayal, Saxena, IITKanpur)

Checking whether a number is prime

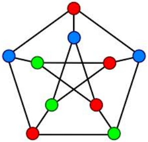


- For $i = 2$ to $\sqrt{n}$
  check if i divides n, if so, it is not prime,
  else (no factor till $\sqrt{n}$), it is prime.
- Takes $\sqrt{n}$ divisions.
- Recently (about 10 years ago), this has been improved to $O(\log^5 n)$ divisions using techniques from algebraic number theory. Major recent contribution from India (Agrawal, Kayal, Saxena, IITKanpur)
- Manindra Agrawal won several awards including Bhatnagar award, Goedel prize, Infosys award

Outline

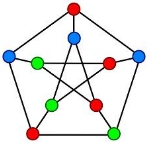
Graph Algorithms

- A number of real world optimization problems can be modeled as a graph theoretic problem such as
  - Traveling Salesperson Problem, Graph Coloring
  - Shortest/Longest Paths, Games and Puzzles
- and hence Graph Algorithms form a huge area in the study of algorithms.



Graph Algorithms

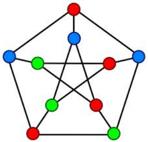
- A number of real world optimization problems can be modeled as a graph theoretic problem such as
  - Traveling Salesperson Problem, Graph Coloring
  - Shortest/Longest Paths, Games and Puzzles
- and hence Graph Algorithms form a huge area in the study of algorithms.



For many of them, the best known method takes exponential time (2^n steps) and

Graph Algorithms

- A number of real world optimization problems can be modeled as a graph theoretic problem such as
  - Traveling Salesperson Problem, Graph Coloring
  - Shortest/Longest Paths, Games and Puzzles
- and hence Graph Algorithms form a huge area in the study of algorithms.



For many of them, the best known method takes exponential time (2^n steps) and whether they have polynomial time (say n^5) algorithms is a million dollar open problem.

Summary

Problem	Time	Can we do better?
Find number	$\log n$	NO
GCD	$O(\log n)$	NO
Find max	$n - 1$	NO
Integer Multiplication	$O(n \log n \log \log n)$	OPEN
Sorting	$O(n \log n)$	NO
Matrix Multiplication	$O(n^{2.236})$	OPEN
Primality	$O(\log^5 n)$	OPEN
TSP	c^n (for some $c < 2$)	OPEN

Outline

Conclusions

Conclusions

- Efficient algorithms form the fundamental core behind a software (For example, Google, Airline scheduling – Makemytrip, ...).

Conclusions

- Efficient algorithms form the fundamental core behind a software (For example, Google, Airline scheduling – Makemytrip, ...).
- Either by understanding the problem or by using sophisticated algorithmic techniques, one can design better algorithms.

Conclusions

- Efficient algorithms form the fundamental core behind a software (For example, Google, Airline scheduling – Makemytrip, ...).
- Either by understanding the problem or by using sophisticated algorithmic techniques, one can design better algorithms.
- Better algorithms can make a difference between a software running in seconds versus minutes or hours or months or years.

Conclusions

- Efficient algorithms form the fundamental core behind a software (For example, Google, Airline scheduling – Makemytrip, ...).
- Either by understanding the problem or by using sophisticated algorithmic techniques, one can design better algorithms.
- Better algorithms can make a difference between a software running in seconds versus minutes or hours or months or years.
- There is a theory of non existence of efficient algorithms for some problems – and even this is useful

Conclusions

- Efficient algorithms form the fundamental core behind a software (For example, Google, Airline scheduling – Makemytrip, ...).
- Either by understanding the problem or by using sophisticated algorithmic techniques, one can design better algorithms.
- Better algorithms can make a difference between a software running in seconds versus minutes or hours or months or years.
- There is a theory of non existence of efficient algorithms for some problems – and even this is useful in areas like cryptography (for password protection etc).

Conclusions

- Efficient algorithms form the fundamental core behind a software (For example, Google, Airline scheduling – Makemytrip, ...).
- Either by understanding the problem or by using sophisticated algorithmic techniques, one can design better algorithms.
- Better algorithms can make a difference between a software running in seconds versus minutes or hours or months or years.
- There is a theory of non existence of efficient algorithms for some problems – and even this is useful in areas like cryptography (for password protection etc).
- Deep and interesting mathematics are behind designing and analysing efficient algorithms.

Thank you