01204211 Discrete Mathematics
Lecture 1: Introduction

Jittat Fakcharoenphol

August 16, 2015

The goals of this course

There are two goals:

- To learn how to make mathematical arguments.

The goals of this course

There are two goals:

- To learn how to make mathematical arguments.
- To learn various fundamental mathematical concepts that are very useful in computer science.

What is mathematics?

What is mathematics?

Ah... that's a philosophical question.

What is mathematics?

Ah... that's a philosopical question.
IMHO, mathematics is a mean to communicate *precise* ideas.

It's like learning a new language

Why should we learn how to prove? (1)

Why should we learn how to prove? (1)

Look at this program.

```
if a > b:
    return a
else:
    return b
```

The author claims that this program takes two variables a and b and returns the larger one.

Why should we learn how to prove? (1)

Look at this program.

```
if a > b:
    return a
else:
    return b
```

The author claims that this program takes two variables a and b and returns the larger one.
Do you believe the author of the code?

Why should we learn how to prove? (1)

Look at this program.

```
if a > b:
    return a
else:
    return b
```

The author claims that this program takes two variables a and b and returns the larger one.
Do you believe the author of the code? Why?

Finding the maximum value. (1)

Now look at this program.

```
if a > b:
    if a > c:
        return a
    else:
        return c
else:
    if c > b:
        return c
    else:
        return b
```

The author claims that this program takes three variables a, b and c and returns the largest one.

Finding the maximum value. (1)

Now look at this program.

```
if a > b:
    if a > c:
        return a
    else:
        return c
else:
    if c > b:
        return c
    else:
        return b
```

The author claims that this program takes three variables a, b and c and returns the largest one.
Do you believe the author of the code?

Finding the maximum value. (1)

Now look at this program.

```
if a > b:
    if a > c:
        return a
    else:
        return c
else:
    if c > b:
        return c
    else:
        return b
```

The author claims that this program takes three variables a, b and c and returns the largest one.
Do you believe the author of the code? Why?

Finding the maximum value. (2)

Finally, look at this program.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = 0
for i = 0, 1, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

The author claims that this program takes an array A with n elements and returns the maximum element.

Finding the maximum value. (2)

Finally, look at this program.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = 0
for i = 0, 1, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

The author claims that this program takes an array A with n elements and returns the maximum element.
Do you believe the author of the code?

Finding the maximum value. (2)

Finally, look at this program.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = 0
for i = 0, 1, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

The author claims that this program takes an array A with n elements and returns the maximum element.
Do you believe the author of the code? Why?

Finding the maximum value. (2)

Finally, look at this program.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = 0
for i = 0, 1, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

The author claims that this program takes an array A with n elements and returns the maximum element.
Do you believe the author of the code? Why?
Can we try to test the code with all possible inputs?

Finding the maximum value. (3)

Let's try again.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = A[0]
for i = 1, 2, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

Do you believe the author of the code?

Finding the maximum value. (3)

Let's try again.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = A[0]
for i = 1, 2, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

Do you believe the author of the code? Why?

Finding the maximum value. (3)

Let's try again.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = A[0]
for i = 1, 2, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

Do you believe the author of the code? Why?
Can we try to test the code with all possible inputs?

Another example: testing primes (1)

A *prime* is a natural number greater than 1 that has no positive divisors other 1 and itself. E.g., 2,3,7,11 are primes.

Another example: testing primes (1)

A *prime* is a natural number greater than 1 that has no positive divisors other 1 and itself. E.g., 2,3,7,11 are primes.

```
// Input: an integer n
if n <= 1:
    return False
i = 2
while i <= n-1:
    if n is divisible by i:
        return False
    i = i + 1
return True
```

The code above checks if n is a prime number. How fast can it run?

Another example: testing primes (1)

A *prime* is a natural number greater than 1 that has no positive divisors other 1 and itself. E.g., 2,3,7,11 are primes.

```
// Input: an integer n
if n <= 1:
    return False
i = 2
while i <= n-1:
    if n is divisible by i:
        return False
    i = i + 1
return True
```

The code above checks if n is a prime number. How fast can it run?
Note that if n is a prime number, the for-loop repeats for $n-2$ times. Thus, the running time is approximately proportional to n.

Another example: testing primes (1)

A *prime* is a natural number greater than 1 that has no positive divisors other 1 and itself. E.g., 2,3,7,11 are primes.

```
// Input: an integer n
if n <= 1:
    return False
i = 2
while i <= n-1:
    if n is divisible by i:
        return False
    i = i + 1
return True
```

The code above checks if n is a prime number. How fast can it run?
Note that if n is a prime number, the for-loop repeats for $n-2$ times. Thus, the running time is approximately proportional to n.
Can we do better?

Another example: testing primes (2)

Consider the following code.

```
// Input: an integer n
if n <= 1:
    return False
let s = square root of n
i = 2
while i <= s:
    if n is divisible by i:
        return False
    i = i + 1
return True
```

How fast can it run?

Another example: testing primes (2)

Consider the following code.

```
// Input: an integer n
if n <= 1:
    return False
let s = square root of n
i = 2
while i <= s:
    if n is divisible by i:
        return False
    i = i + 1
return True
```

How fast can it run? Note that $s = \sqrt{n}$; therefore, it takes time approximately proportional to $\sqrt{n}$ to run.

Another example: testing primes (2)

Consider the following code.

```
// Input: an integer n
if n <= 1:
    return False
let s = square root of n
i = 2
while i <= s:
    if n is divisible by i:
        return False
    i = i + 1
return True
```

How fast can it run? Note that $s = \sqrt{n}$; therefore, it takes time approximately proportional to $\sqrt{n}$ to run.
Ok, it should be faster. **But is it correct?**

Informal arguments (1)

- Let's try to argue that the faster algorithm works correctly.

Informal arguments (1)

- Let's try to argue that the faster algorithm works correctly.
- Note that if n is a prime number, the algorithm answers correctly. (Why?)

Informal arguments (1)

- Let's try to argue that the faster algorithm works correctly.
- Note that if n is a prime number, the algorithm answers correctly. (Why?)
- Therefore, let's consider the case when n is not prime (i.e., n is a composite).

Informal arguments (1)

- Let's try to argue that the faster algorithm works correctly.
- Note that if n is a prime number, the algorithm answers correctly. (Why?)
- Therefore, let's consider the case when n is not prime (i.e., n is a composite).
- If that's the case, n has some positive divisor which is not 1 or n. Let's call this number a.

Informal arguments (1)

- Let's try to argue that the faster algorithm works correctly.
- Note that if n is a prime number, the algorithm answers correctly. (Why?)
- Therefore, let's consider the case when n is not prime (i.e., n is a composite).
- If that's the case, n has some positive divisor which is not 1 or n. Let's call this number a.
- Now, if $2 \leq a \leq \sqrt{n}$, at some point during the execution of the algorithm, $i = a$ and i should divides n; thus the algorithm correctly returns `False`.

Informal arguments (1)

- Let's try to argue that the faster algorithm works correctly.
- Note that if n is a prime number, the algorithm answers correctly. (Why?)
- Therefore, let's consider the case when n is not prime (i.e., n is a composite).
- If that's the case, n has some positive divisor which is not 1 or n. Let's call this number a.
- Now, if $2 \leq a \leq \sqrt{n}$, at some point during the execution of the algorithm, $i = a$ and i should divides n; thus the algorithm correctly returns `False`.
- Are we done?

Informal arguments (2)

- Recall that we are left with the case that (1) n is not prime and (2) its positive divisor a is larger than $\sqrt{n}$.

Informal arguments (2)

- Recall that we are left with the case that (1) n is not prime and (2) its positive divisor a is larger than $\sqrt{n}$.
- Let $b = n/a$. Since n and a are positive integers and a divides n, b is also a positive integer.

Informal arguments (2)

- Recall that we are left with the case that (1) n is not prime and (2) its positive divisor a is larger than $\sqrt{n}$.
- Let $b = n/a$. Since n and a are positive integers and a divides n, b is also a positive integer.
- Note that if we can argue that $2 \leq b \leq \sqrt{n}$, we are done. (why?)

Informal arguments (2)

- Recall that we are left with the case that (1) n is not prime and (2) its positive divisor a is larger than $\sqrt{n}$.
- Let $b = n/a$. Since n and a are positive integers and a divides n, b is also a positive integer.
- Note that if we can argue that $2 \leq b \leq \sqrt{n}$, we are done. (why?)
- How can we do that?

Informal arguments (3): quick break

- **Original goal:** To show that the faster algorithm is correct.

Informal arguments (3): quick break

- **Original goal:** To show that the faster algorithm is correct.
- **Current (sub) goal:** Consider a positive composite n and its positive divisor a, where $a > \sqrt{n}$. Let $b = n/a$. We want to show that $2 \leq b \leq \sqrt{n}$.