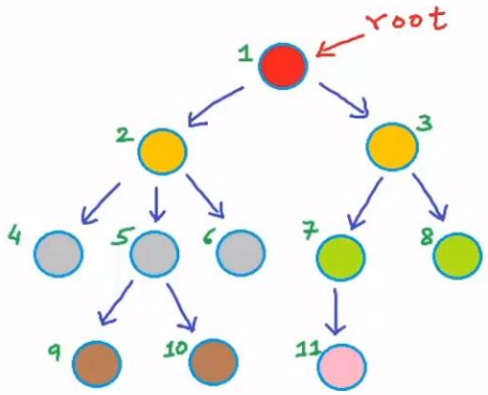
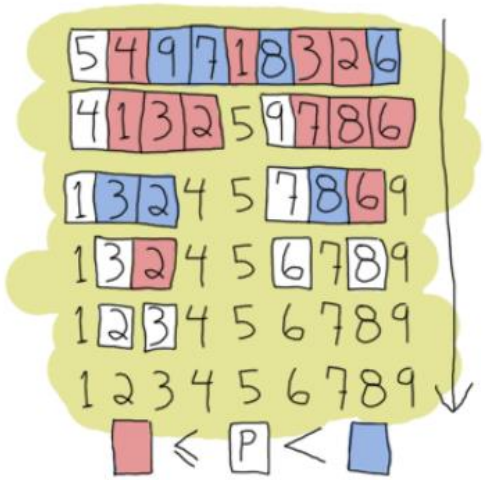
ECE 250 Data Structures & Algorithms



Devising Algorithms



Ziqiang Patrick Huang

Electrical and Computer Engineering

University of Waterloo

WATERLOO | ENGINEERING

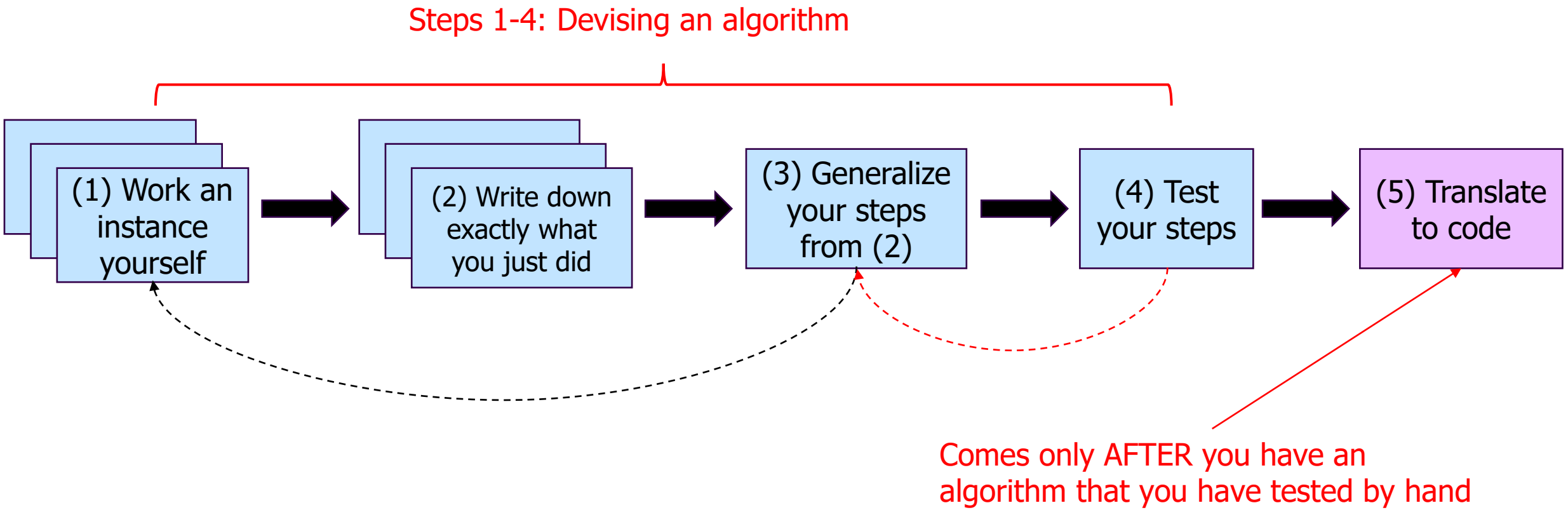
Outline

- How to come up with an algorithm?

How to Write a Program?

- Devising an algorithm
  - Clear set of step-by-step instructions
  - Solves any problem in a certain class of problems
  - Parameterized to identify which particular problem
- Implementing that algorithm in a programming language
  - Translating the steps into the syntax of a particular language
  - As well as testing, debugging, etc
- Many novice programmers tend to skip the first step and jump right into writing code (No time to plan! Busy schedule!)
  - Then pour countless hours into trying to fix the code

5 Steps to Writing a Program



5 Steps to Writing a Program

1. Work an instance of the problem itself
   - Pick specific values for each parameter
   - Maybe a few to get the feel of it if its hard
   - Can't do this? Either need clarification or domain knowledge

- Domain knowledge:
  - Knowledge specific to problem domain: math, physics, biology, computer architecture …
  - If you don't have the relevant domain knowledge, you can't hope to write a program about it (no matter how good your programming skill is!)
  - If you are stuck here, you should consult a source of domain expertise – a textbook, website, expert, etc.

5 Steps to Writing a Program

1. Work an instance of the problem itself
   - Maybe a few to get the feel of it if its hard
   - Can't do this? Either need domain knowledge or clarification
2. Write down exactly what you did to solve that instance
   - A clear set of instructions that anyone else could follow to reproduce your answer for the particular problem instance that you just solved
   - If an instruction is complex?
     - Abstract it out into a function

- Challenge in Step 2:
  - Easy to mentally gloss over small details, "easy" steps, or things you do implicitly

5 Steps to Writing a Program

1. Work an instance of the problem itself
   - Maybe a few to get the feel of it if its hard
   - Can't do this? Either need domain knowledge or clarification
2. Write down exactly what you did to solve that instance
   - In a level of sophistication that anyone else could follow to reproduce your answer for the particular instance that you just solved
3. Generalize your steps
   - Determine what numbers depend on parameters
   - Find repetitions

5 Steps to Writing a Program

1. Work an instance of the problem itself
   - Maybe a few to get the feel of it if its hard
   - Can't do this? Either need domain knowledge or clarification
2. Write down exactly what you did to solve that instance
   - In a level of sophistication that anyone else could follow to reproduce your answer for the particular instance that you just solved
3. Generalize your steps
   - Determine what numbers depend on parameters
   - Find repetitions
4. Test your generalized steps on another instance
   - Generalized wrong? Find it now!

5 Steps to Writing a Program

1. Work an instance of the problem itself
   - Maybe a few to get the feel of it if its hard
   - Can't do this? Either need domain knowledge or clarification
2. Write down exactly what you did to solve that instance
   - In a level of sophistication that anyone else could follow to reproduce your answer for the particular instance that you just solved
3. Generalize your steps
   - Determine what numbers depend on parameters
   - Find repetitions
4. Test your generalized steps on another instance
   - Generalized wrong? Find it now!
5. Translate generalized steps to code

No really, Plan Before You Code

- How do you build a skyscraper? (Or even a house?)
  - Option 1: start building, figure out where things go as you build …
  - Option 2: Have an architect carefully plan everything out, developing a blueprint that specifies every detail of the construction. Get it approved by the city, etc… **THEN** break ground and start building

- Many new programmers choose the analog of option 1
  - Note that is OK iff problem is very easy and can do steps 1-4 trivially in your head.

Very Basic Example: isPrime

- Let's see these steps in action on a relatively simple problem:
  - Given a number N, is N prime?

- Step1?
  - Work (at least) one instance of the problem ourselves by hand
  - Is 7 prime?
    - Note: "yes, I just know it is" is not the way to go
    - If you can only come up with "I just know it," try larger: is 94723 prime?
    - Probably won't work it all out, but might get the gist of what to do for 7.

Is 7 Prime?

- Well let's see:
  - 7 /2 = 3 R 1
  - 7 /3 = 2 R 1
  - 7 /4 = 1 R 3
  - 7 /5 = 1 R 2
  - 7 /6 = 1 R 1

  - Yes, 7 is prime

- Step 2?

Is 7 Prime?

- Well let's see:
  - 7 /2 = 3 R 1
  - 7 /3 = 2 R 1
  - 7 /4 = 1 R 3
  - 7 /5 = 1 R 2
  - 7 /6 = 1 R 1

  - Yes, 7 is prime

- Step 2?
  - Write down exactly what we did
  - May require thinking about things you did intuitively

Is 7 Prime?

- Well let's see:
  - 7 /2 = 3 R 1
  - 7 /3 = 2 R 1
  - 7 /4 = 1 R 3
  - 7 /5 = 1 R 2
  - 7 /6 = 1 R 1

<- Checked if 7 was divisible by 2, remainder of 1 meant it was not did this for 3 … 6

  - Yes, 7 is prime

- Step 2?
  - Write down exactly what we did
  - May require thinking about things you did intuitively

Step 2: Write Down What We Did

- Steps to see if 7 is prime:
  - Check if 7 is divisible by 2 (it's not)
  - Check if 7 is divisible by 3 (it's not)
  - Check if 7 is divisible by 4 (it's not)
  - Check if 7 is divisible by 5 (it's not)
  - Check if 7 is divisible by 6 (it's not)
  - Answer "yes"

- Note that these steps work for 7 (and only for 7)
  - Really boring as an algorithm: no parameters, kind of useless

Step 3: Generalize

- Steps to see if N is prime:
  - Check if 7 is divisible by 2 (it's not)
  - Check if 7 is divisible by 3 (it's not)
  - Check if 7 is divisible by 4 (it's not)
  - Check if 7 is divisible by 5 (it's not)
  - Check if 7 is divisible by 6 (it's not)
  - Answer "yes"

- What to generalize in steps to see if N is prime
  - Figure out why each number is what it is
  - Look for repetition

Step 3: Generalize

- Steps to see if N is prime:
  - Check if 7 is divisible by 2 (it's not)
  - Check if 7 is divisible by 3 (it's not)
  - Check if 7 is divisible by 4 (it's not)
  - Check if 7 is divisible by 5 (it's not)
  - Check if 7 is divisible by 6 (it's not)
  - Answer "yes"

- These are all 7
  - Are they always 7 for any value of N?
  - No: they are just 7 because we picked N=7
  - In general, want to check divisibility of N

Step 3: Generalize

- Steps to see if N is prime:
  - Check if N is divisible by 2 (it's not)
  - Check if N is divisible by 3 (it's not)
  - Check if N is divisible by 4 (it's not)
  - Check if N is divisible by 5 (it's not)
  - Check if N is divisible by 6 (it's not)
  - Answer "yes"

Step 3: Generalize

- Steps to see if N is prime:
  - Check if N is divisible by 2 (it's not)
  - Check if N is divisible by 3 (it's not)
  - Check if N is divisible by 4 (it's not)
  - Check if N is divisible by 5 (it's not)
  - Check if N is divisible by 6 (it's not)
  - Answer "yes"

- What about 2..6?
  - Do we always check exactly these?
  - No…
  - Where do we start?
  - Where do we end?

Step 3: Generalize

- Steps to see if N is prime:
  - Check if N is divisible by 2 (it's not)
  - Check if N is divisible by 3 (it's not)
  - Check if N is divisible by 4 (it's not)
  - Check if N is divisible by 5 (it's not)
  - Check if N is divisible by 6 (it's not)
  - Answer "yes"

- What about 2..6?
  - Do we always check exactly these?
  - No…
  - Where do we start? Always 2
  - Where do we end? N-1

Step 3: Generalize

- Steps to see if N is prime:
  - Count from 2 to N-1 (inclusive), for each number X that you count
    - Check if N is divisible by X (it's not)
  - Answer "yes"

- Are we done?

Step 3: Generalize

- Steps to see if N is prime:
  - Count from 2 to N-1 (inclusive), for each number X that you count
    - Check if N is divisible by X (it's not)
  - Answer "yes"

- (it's not) is not always the case
  - We mostly wrote this down as part of our thought process
  - What if it is divisible?
  - May be obvious: if so, do what you need
  - May not be obvious: Repeat steps 1 + 2 on different examples until you understand

Step 3: Generalize

- Steps to see if N is prime:
  - Count from 2 to N-1 (inclusive), for each number X that you count
    - Check if N is divisible by X
      - If so: stop and answer "no"
      - If not: (nothing special, keep going)
  - Answer "yes"

- These steps look reasonable:
  - Clear/simple/straightforward: no ambiguity, very step-by-step
  - Are they right?
    - Become more confident, but never certain: test it
    - Testing can find the presence of errors, not the absence.

Step 4: Test Your Generalized Steps

- Try out your steps on other values
  - Make sure you can get some "yes" answers and some "no" answers
  - Are there any corner cases?
    - Where your algorithm has to do something special for a special value?
    - Test those explicitly
  - Get at least "statement coverage"
    - Should test every step at least once

- For this, what should we test with?
  - Yes answers: 5, 13
  - No answers: 6, 25
  - Corner cases: 0, 1, 2, negative numbers?

Step 4: Test with 0

- Steps to see if N is prime:
  - Count from 2 to N-1 (inclusive), for each number X that you count
    - Check if N is divisible by X
      - If so: stop and answer "no"
      - If not: (nothing special, keep going)
  - Answer "yes"

- Let's see: 0 is NOT prime (infinite divisors)
  - Count from 2 to -1?
    - We mean "count up by one" (even though we didn't say it)…
    - This is how we were counting when we wrote our steps
    - So nothing in this range
  - Get answer of "yes" - oops.

Step 4: Fix Our Generalized Steps

- Steps to see if N is prime:
  - If N is less than or equal to 1, stop and answer "no"
  - Count from 2 to N-1 (inclusive), for each number X that you count
  - Check if N is divisible by X
    - If so: stop and answer "no"
    - If not: (nothing special, keep going)
  - Answer "yes"

- Fix algorithm:
  - All primes are > 1
  - So we can answer "no" immediately for <= 1

- Are we done?
  - What about N = 2.76 or N="hello world" or N=false?

Step 5: Translate to Code

- **Steps to see if N is prime:**
  - If N is less than or equal to 1, stop and answer “no”
  - Count from 2 to N-1 (inclusive), for each number X that you count
  - Check if N is divisible by X
    - If so: stop and answer “no”
    - If not: (nothing special, keep going)
  - Answer “yes”

- Language dependent:
  - Let’s do C first

Step 5: Translate to Code

```
int isPrime (int N) {
```

- **If N is less than or equal to 1, stop and answer "no"**
- Count from 2 to N-1 (inclusive), for each number X that you count
- Check if N is divisible by X
  - If so: stop and answer "no"
  - If not: (nothing special, keep going)
- Answer "yes"

```
}
```

- Each step should translate into one line of code

Step 5: Translate to Code

```
int isPrime (int N) {
    if (n <= 1) { return 0; }
```

- **Count from 2 to N-1 (inclusive), for each number X that you count**
- Check if N is divisible by X
  - If so: stop and answer "no"
  - If not: (nothing special, keep going)
- Answer "yes"

```
}
```

- Conditional decisions are if statements
  - Or if-else

Step 5: Translate to Code

```
int isPrime (int N) {
   if (n <= 1) { return 0; }
   for (int X = 2; X <= N-1; X++) {
```

- **Check if N is divisible by X**
  - If so: stop and answer "no"
  - If not: (nothing special, keep going)

```
   }
```

- Answer "yes"

```
}
```

- Counting is a for loop
  - Be careful whether you mean < or <=

Step 5: Translate to Code

```
int isPrime (int N) {
    if (n <= 1) { return 0; }
    for (int X = 2; X <= N-1; X++) {
        if (isDivisibleBy(N, X)) {
```

- **stop and answer "no"**

```
        }
        else {
```

- (nothing special, keep going)

```
        }
    }
```

- Answer "yes"

```
}
```

- Good place for abstraction:
  - Pull `isDivisibleBy` out into a separate function, write when done

Step 5: Translate to Code

```
int isPrime (int N) {
    if (n <= 1) { return 0; }
    for (int X = 2; X <= N-1; X++) {
        if (isDivisibleBy(N, X)) {
            return 0;
        }
        else {
```

- **(nothing special, keep going)**

```
        }
    }
```

- Answer “yes”

```
}
```

- Giving the answer back is returning a value

Step 5: Translate to Code

```
int isPrime (int N) {
   if (n <= 1) { return 0; }
   for (int X = 2; X <= N-1; X++) {
      if (isDivisibleBy(N, X)) {
         return 0;
      }
      else {
      }
   }
```

- **Answer "yes"**

```
}
```

- Doing nothing is no statements

Step 5: Translate to Code

```
int isPrime (int N) {
   if (n <= 1) { return 0; }
   for (int X = 2; X <= N-1; X++) {
      if (isDivisibleBy(N, X)) {
         return 0;
      }
      else {
      }
   }
   return 1;
}
```

- Answering “yes” is returning ‘true’ (1)

Clean Up

```
int isPrime (int n) {
   if (n <= 1) { return 0; }
   for (int n = 2; x <= n-1; n++) {
      if (isDivisibleBy(n, x)) {
         return 0;
      }
   }
   return 1;
}
```

- Lowercase variables. Remove empty else statement.

Other languages?

- C/C++/Java: almost identical

isPrime: C

```
int isPrime (int n) {
   if (n <= 1) { return 0; }
   for (int n = 2; x <= n-1; n++) {
      if (isDivisibleBy(n, x)) {
         return 0;
      }
      else {
      }
   }
   return 1;
}
```

isPrime: C++

```
bool isPrime (int n) {
   if (n <= 1) { return false; }
   for (int n = 2; x <= n-1; n++) {
      if (isDivisibleBy(n, x)) {
         return false;
      }
   }
   return true;
}
```

- C++ has an separate boolean type (called bool)
  - And literals for true and false

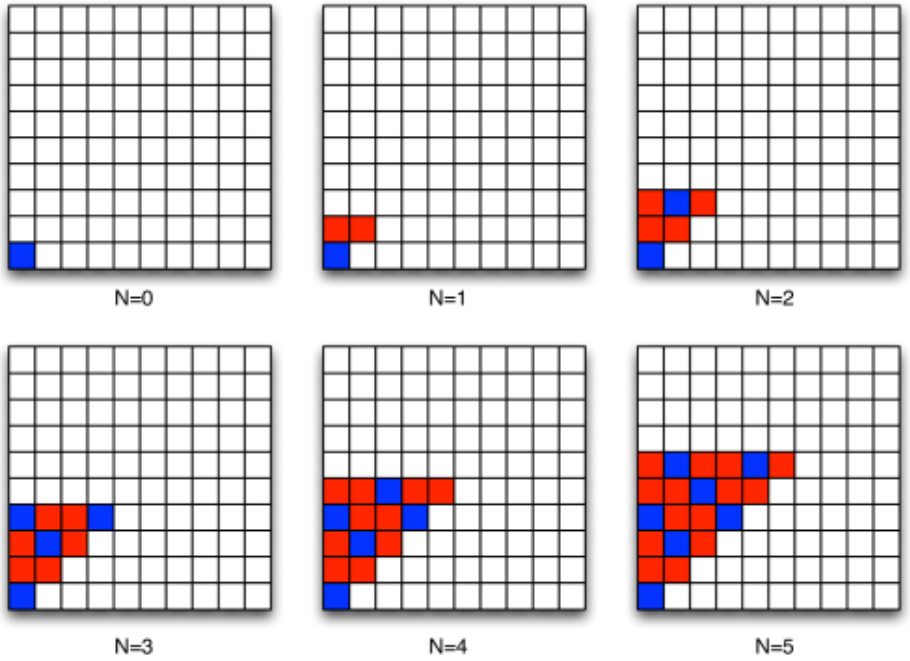
isPrime: Java

```
boolean isPrime (int n) {
   if (n <= 1) { return false; }
   for (int n = 2; x <= n-1; n++) {
      if (isDivisibleBy(n, x)) {
         return false;
      }
   }
   return true;
}
```

- Java: looks like C++, but spells out boolean

Exercise for You: Pattern of Squares on a Grid

- We have an algorithm that is parameterized over one integer N and produces a pattern of red and blue square on a grid that starts all white. The output of the algorithm for N = 0 to N = 5 is as follows:



Your task: Walk through Steps 1-4 to devise the algorithm

Patrick's advice: Do not skip it!

Next up

- Algorithm analysis (Big-O)

Acknowledgement

- This slide builds on the hard work of the following amazing instructors:
  - Andrew Hilton (Duke)