COMPUTER SCIENCE 1: STARTING COMPUTING CSCI 1300

Ioana Fleming / Vipra Gupta
Spring 2018
Lecture 23

Announcements

- Rec 9 due on 3/17
- Hmwk 7 (Project 2)
  - Part I – due on 3/18
  - Part II – due on 3/25
- Hmwk 8 (Project 3)
  - Proposal – due 3/19
  - Classes & Code Skeleton – due 4/8
  - Final deliverables – due 4/22

Agenda

- Today:
  - Sorting algorithms

Activity

- Write a program that receives three integers as input and outputs the numbers in increasing order.

- Do not use loop / array!

Strategy 1: Compare Each to All

- This looks like a three-way decision, where we need to execute *one* of the following:

```
max = x1;
max = x2;
max = x3;
```

- All we need to do now is preface each one of these with the right condition!

Strategy 1: Compare Each to All

- Let's look at the case where x1 is the largest.

```
if (x1 >= x2 >= x3)
  max = x1;
```

- Is this syntactically correct?

Strategy 1: Compare Each to All

```
if (x1 >= x2 && x1 >= x3)
    max = x1;
else if (x2 >= x1 && x2 >= x3)
       max = x2;
else
    max = x3;
```

- We're comparing each possible value against all the others to determine which one is largest.

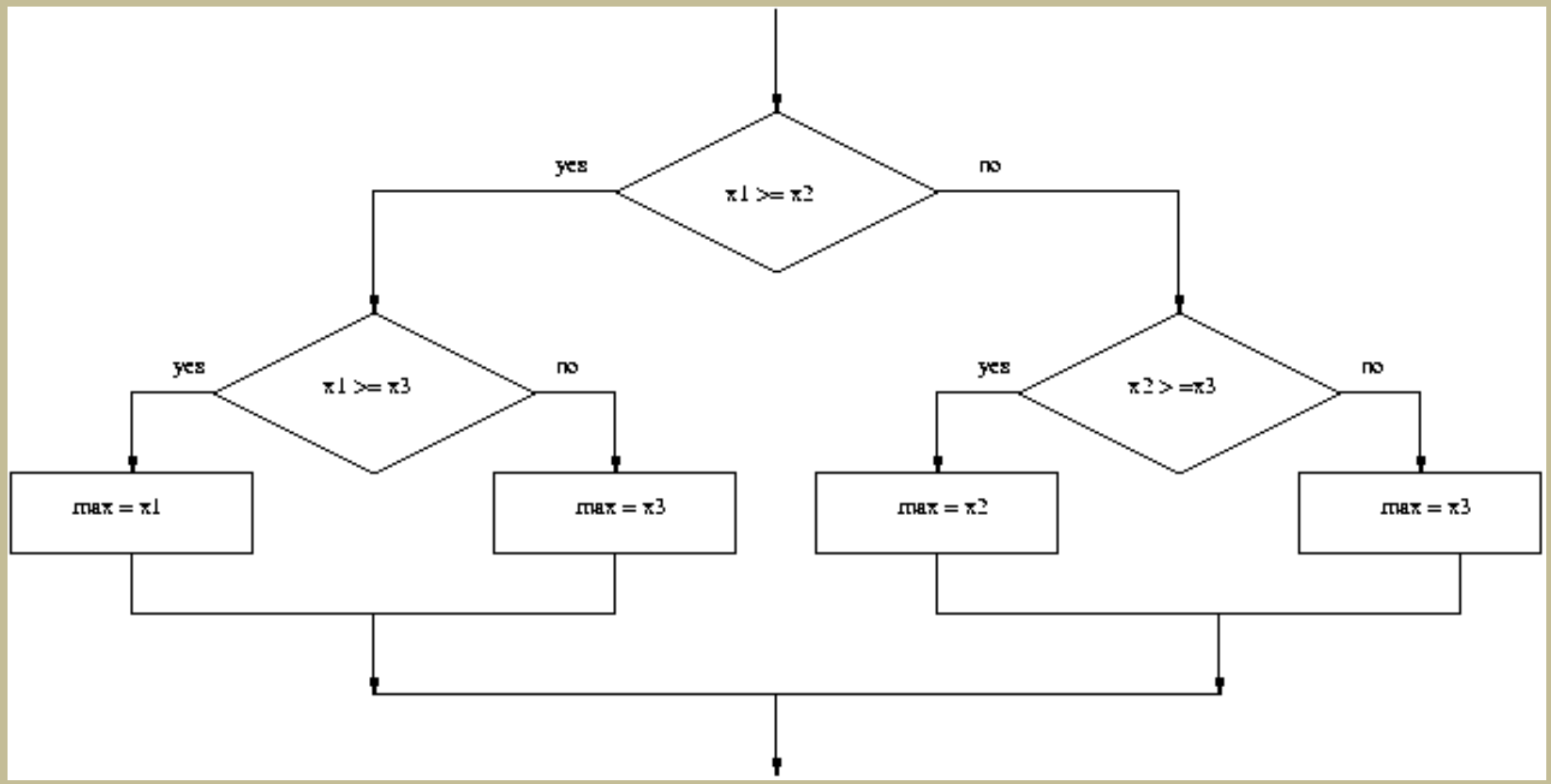
Strategy 1: Compare Each to All

- What would happen if we were trying to find the max of five values?
- We would need four Boolean expressions, each consisting of four conditions *and*-ed together.
- Yuck!

Strategy 2: Decision Tree

- We can avoid the redundant tests of the previous algorithm using a *decision tree* approach.
- Suppose we start with `x1 >= x2`. This knocks either `x1` or `x2` out of contention to be the max.
- If the conidition is true, we need to see which is larger, `x1` or `x3`.

Strategy 2: Decision Tree



Strategy 2: Decision Tree

```
if (x1 >= x2)
{
   if (x1 >= x3)
      max = x1;
   else
      max = x3;
}
else
{
   if (x2 >= x3)
      max = x2;
   else
      max = x3;
}
```

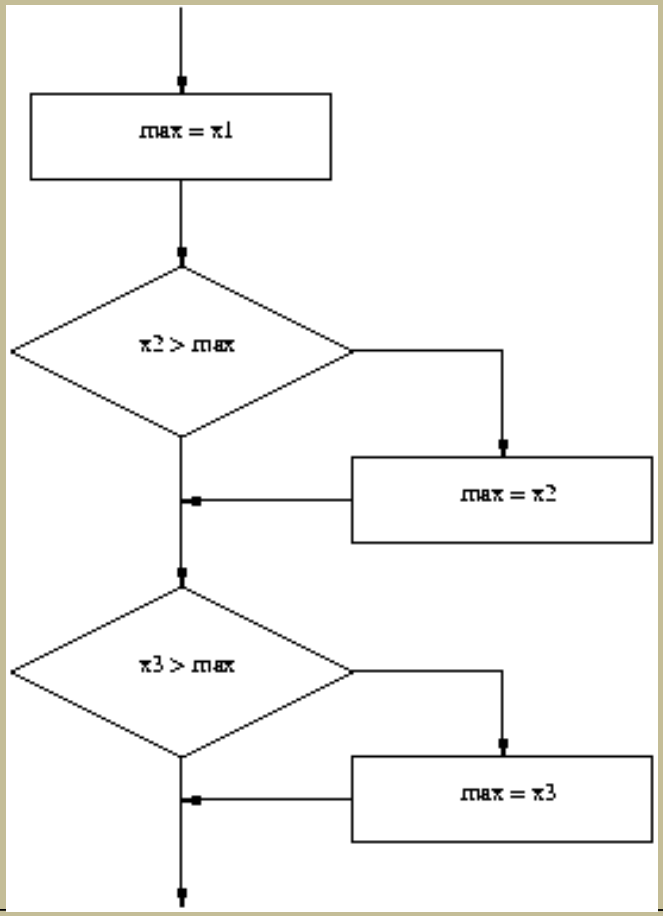
Strategy 2: Decision Tree

- This approach makes exactly two comparisons, regardless of the ordering of the original three variables.
- However, this approach is more complicated than the first. To find the max of four values you'd need `if-else`s nested three levels deep with eight assignment statements.

Strategy 3: Sequential Processing

- How would you solve the problem?
- You could probably look at three numbers and just *know* which is the largest. But what if you were given a list of a hundred numbers?
- One strategy is to scan through the list looking for a big number. When one is found, mark it, and continue looking. If you find a larger value, mark it, erase the previous mark, and continue looking.

Strategy 3: Sequential Processing



Strategy 3: Sequential Processing

```
max = x1;
if (x2 > max)
    max = x2;
if (x3 > max)
    max = x3;
```

Strategy 3: Sequential Programming

- This process is repetitive and lends itself to using a loop.
- We prompt the user for a number, we compare it to our current max, if it is larger, we update the max value, repeat.

Some Lessons

- There's usually more than one way to solve a problem.
  - Don't rush to code the first idea that pops out of your head. Think about the design and ask if there's a better way to approach the problem.
  - Your first task is to find a correct algorithm. After that, strive for clarity, simplicity, efficiency, scalability, and elegance.

Some Lessons

- Be the computer.
  - One of the best ways to formulate an algorithm is to ask yourself how you would solve the problem.
  - This straightforward approach is often simple, clear, and efficient enough.

Some Lessons

- Generality is good.
  - Consideration of a more general problem can lead to a better solution for a special case.
  - If the max of *n* program is just as easy to write as the max of three, write the more general program because it's more likely to be useful in other situations.

Sorting

Getting data into order is something that is often needed.

An alphabetical listing.

A list of grades in descending order.

Sorting Algorithms

- The basic sorting problem is to take a list and rearrange it so that the values are in increasing (or nondecreasing) order.

Visualizing sorting algorithms:

https://visualgo.net/en/sorting