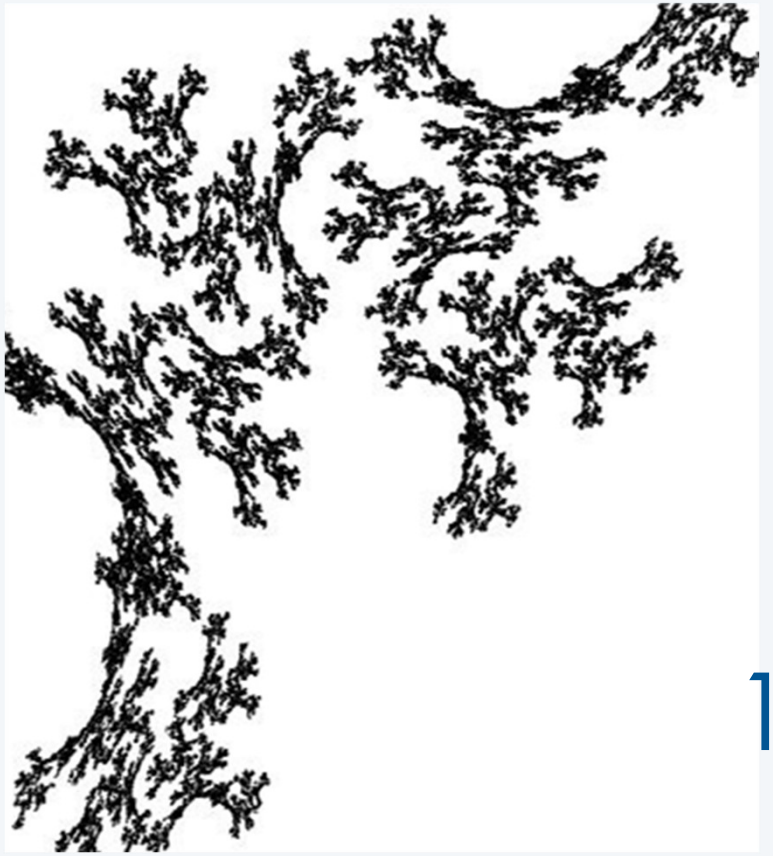


INTRODUCTION TO
DISCRETE STRUCTURES

10.1 Loop invariants

1.1–1.2

10.1 Loop invariants

- Introduction
- Examples of loop invariants
- Finding a loop invariant
- Proofs using loop invariants

Why Bugs Matter

It's fun to bring stuff into class from the headlines, so I keep my eye out for real code (i.e. bugs!) in the news. So the day before the election, something happened with the fivethirtyeight website where it changed to show Trump with a 99% percent chance of winning, which was not actually what their model had.

Turns out it was a bug. The issue is that their model weights non-poll data as some fraction, and that fraction goes to zero on the day before the election. Turns out when that happened, it drove some numbers to zero in their model which caused regular old div-zero bugs. The if-statement that was supposed to guard against it was off-by-one, kicking in on election day, when it should have been the day before the election.
I think there's two lecture stories here. One it's just an example of div-zero and off-by-one in the real world. The sort of bugs students see all the time show up in the real world just the same.

The other story is that code that has not been run for a type of case will tend to have bugs for that case. It is always a little surprising how buggy non-tested code is. Here, it's easy to see how they had run the code for hundreds of days, but this was the first time they'd run it for the day-before-election case.

We can be a little sympathetic to fivethirtyeight's testing here, like the obvious cases are election day, and non-election day. It's a little subtle that there is this other case. Still, if your code has some if-statement that kicks in at n=10, probably you should check both 9 and 10, checking both sides of the transition. When I want to make that point in lecture, I'm going to try to remember this election day story.

https://fivethirtyeight.com/features/politics-podcast-final-reflections-on-the-2020-campaign-before-election-day/

Introduction

Loop invariant is a property of a loop, that is true before and after each iteration

develop contracts (preconditions, postconditions, assertions, and loop invariants) establish the safety and correctness of imperative programs.

develop proofs of the safety and correctness of code with contracts.

develop informal termination arguments for programs with loops and recursion.

identify the difference between specification and implementation

10.1 Loop invariants

- Introduction
- Examples of loop invariants
- Finding a loop invariant
- Proofs using loop invariants

Loop invariants

A formal definition of a loop invariant is

A Boolean condition that must be true immediately before every evaluation of a loop

A Boolean condition that is true immediately after loop terminates (if the loop terminates)

Example. What is a loop invariant for the following code?

```
while (i < n)  {
      i++;
      x = x + i;
}
```

workshop

Find loop invariants for the following code segment and prove that they are indeed invariants

- p: {i=0, x=1}
- S: while (i < n) {i++; x = x * i;}

Find a post condition q for above code and prove that loop termination and loop invariant proves post-condition.

q: x = 1*2* …*n

Insert in order

How can we insert an element to an already sorted array?

We call this method insertInOrder

What is a loop invariants for insertInOrder code?

```
insertInOrder(a, first, last, item) {
   i = last
   while (i>first && item < a[i])
         { a[i+1]=a[i]; i-- ; }
   a[i] = item;
}
```

10.1 Loop invariants

- Introduction
- Examples of loop invariants
- Finding a loop invariant
- Proofs using loop invariants

Finding a loop invariant

Finding trivial loop invariants are easy, but less useful

Finding a good loop invariant requires some work to establish a relationship between variables that changes within the loop.

What variables change within this while loop?

```
int g (int x) {
  int i = 0;
  int j = 0;
  int k = 0;
  while (j < x) {
    j = j+k+3*i+1;
    k = k+6*i+3;
    i = i+1;
  }
  return i;
}
```

Finding a loop invariant

Consider the function foo given below where a is an array of ints

```
void foo(int[] a) {
  int i = 0, c = 1, n = a.length;
  while (i < n) {
      a[i] = c;
      c +=2;
      i += 1;
  }
}
```

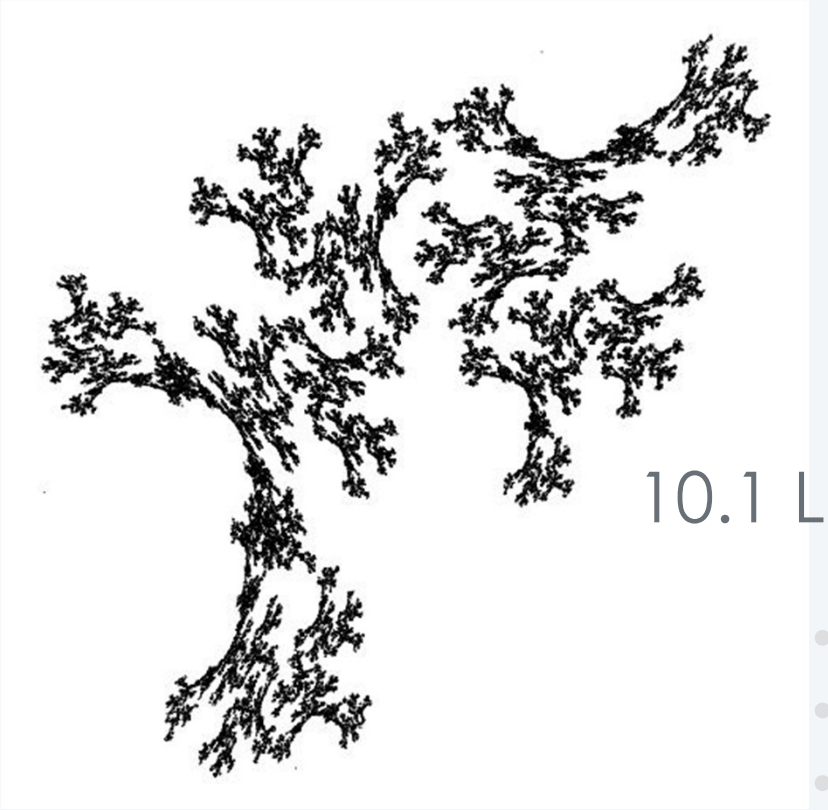
a. Create a table of values for a.length=5
b. Find all loop invariants in the above code

Loop invariants for Insertion sort

The code for insertion sort is given below.

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while
```

a. Find all possible loop invariants for the inner and outer loops



10.1 Loop invariants

- Introduction
- Examples of loop invariants
- Finding a loop invariant
- Proofs using loop invariants

Finding a loop invariant

Consider the body of the function foo given below where a is an array of ints

```
int i=0, c=1, n=a.length;
 while ( i < n) {
     a[i] += c;
     c += 2;
     i += 1;
 }
```

a. Find all loop invariants in the above code
b. Prove that they are indeed loop invariants
c. Show that the above code terminates

Loop invariants for Insertion sort

The code for bubble sort is given below.

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while
```

a. Find all possible loop invariants for the inner and outer loops (see slides before)
b. Formally prove that they are indeed loop invariants
c. What is the termination condition and prove that both the inner and outer loops must terminate.

Examples of loop invariants

```
int i = 0, c=0, k=1, m=6;
int n = a.length;
while (i < n) {
   a[i] = c;
   c = c + k;
   k = k + m;
   m = m + 6;
   i = i + 1;
}
```

- Find all loop invariants
- What does the program do?

INTRODUCTION TO DISCRETE STRUCTURES

10.1 Loop invariants

- Introduction
- Examples of loop invariants
- Finding a loop invariant
- Proofs using loop invariants

INTRODUCTION TO DISCRETE STRUCTURES

10.1 Loop invariants

1.1–1.2