}

COS 126 Lecture 9: Recursion

Recursive program: one that calls itself

MATHEMATICAL INDUCTION:

To prove S(N)

* prove S(0)
* prove S(N), assuming S(k) for all k<N

Ex:

$0 + 1 + 2 + 3 + \ldots + N = N(N+1)/2$

* trivially true for N = 0
* $0 + 1 + 2 + 3 + \ldots + N$

  $= 0 + 1 + 2 + \ldots + N-1 + N$

  $= (N-1)N/2 + N = N(N+1)/2$

RECURSION:

To compute f(N)

* compute f(0)
* compute f(N), using f(k) for k<N

Ex:

```
int f(int N)
  {
    if (N == 0) return 0;
    return f(N-1) + N;
  }
```

Divide and Conquer

Many computations are
naturally expressed as recursive programs

ITERATION
another way to write "for" loop

"DIVIDE and CONQUER"
solve a problem by dividing into smaller ones

Ex: root finding

```
float bisectr(float l, float r)
  {
    float m;
    m = (l + r) / 2;
    if ((r - l) < epsilon) return m;
    if (f(m) > 0.0)
      return bisectr(m, r);
    else
      return bisectr(l, m);
  }
```

Digression: binary search

Bisection for integer functions:

```
int bisectr(int l, int r)
   {
      int m;
      m = (l + r) / 2;
      if (r <= l) return m;
      if (f(m) > 0)
         return bisectr(m+1, r);
      else
         return bisectr(l, m-1);
   }
```

Suppose an array A has N integers, in order

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	**1**	**2**	**5**	**8**	**13**	**21**	**34**	**55**	**89**	**144**	**233**	**377**

SEARCH PROBLEM: is a given integer v in A?

SOLUTION: use above program with

```
int f(int k) { return  v - a[k]; }
```

Ex: v = 144

.	l	r	m	f(m)
.	0	12	6	+
.	7	12	9	+
.	10	12	11	-
.	10	10	10	0

Possible pitfall with recursion

Simple recursive programs
can consume excessive resources

Ex: Compute binomial coefficients

```
int f(int N, int k)
  {
    if ((k < 0) || (k > N)) return 0;
    if (N == 0) return 1;
    return f(N-1, k) + f(N-1, k-1);
  }
```

Seems to run for a long time to compute f(20, 10).

Q: Why?

A: Recomputes intermediate results

Simpler example: hard way to compute 2^N

```
int f(int N)
  {
    if (N == 0) return 1;
    return f(N-1)+f(N-1);
  }
```

Takes time proportional to 2^N (!)

Solution: DYNAMIC PROGRAMMING
save away intermediate results
see Sedgewick, section 5.3

Implementing recursion

Any function call requires

```
*Any* function call requires
     set the values of the parameters
     save the "environment"
     jump to the first instruction in the function
          execute the function
     restore the "environment"
     continue at the instruction after the call
          "return address"  (part of environment)
```

Use pushdown stack for save/restore

```
Use pushdown stack for save/restore
     call: push environment
     return: restore environment from stack
```

Ex:

		STACK (top at left)
A calls B	B	save(A)
B calls C	C	save(B) save(A)
C calls D	D	save(C) save(B) save(A)
D returns	C	save(B) save(A)
C calls E	E	save(C) save(B) save(A)
E returns	C	save(B) save(A)
C returns	B	save(A)
B returns	A	

Traveling Salesman problem

Given a set of points, find the shortest tour
connecting all the points

Recursive solution for trying all possibilities

```
visit(int N)
  {
    int i;
    if ( N == 1) { checklength(); return; }
    for (i = 1; i <= N; i++)
      {
        swap(i, N);
        visit(N-1);
        swap(i, N);
      }
  }
```

Takes N! steps

Can’t run for very large N
no computer can ever run this
to completion for N = 100 [10! > 10^150]

[stay tuned]

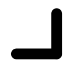
Recursion: Dragon Curve

Fold a strip of paper in half n times

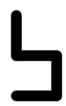
unfold to right angles

n = 0

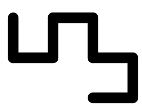
n = 1



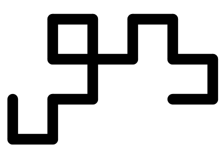
n = 2



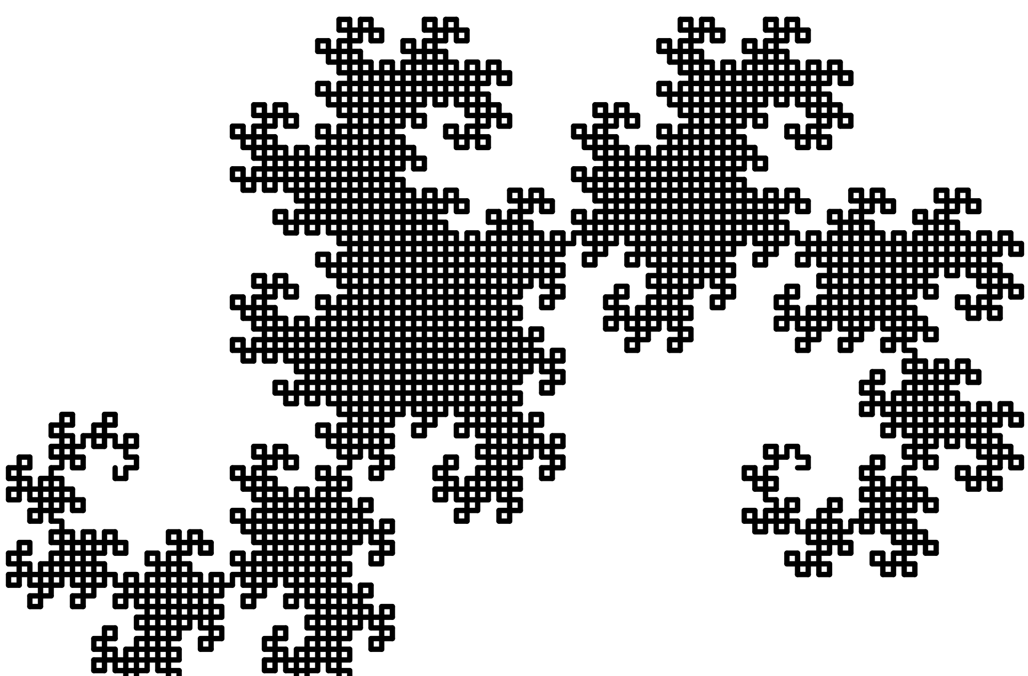
n = 3



n = 4



n = 12



Drawing a Dragon Curve

Use simplest turtle graphics

F: move forward one step (pen down)

L: turn left

R: turn right

n = 0

F

n = 1

F L F

n = 2

F L F L F R F

L L R

n = 3

F L F L F R F L F L F R F R F

L L R L L R R

n = 4

L L R L L R R L L L R R L R R

Recursive program for dragon curve

A dragon curve of order n is

* dragon curve of order n−1
* L
* dragon curve of order n−1, backwards

```
dragon(int n)
{
   if (n == 0) { F(); return; }
   dragon(n-1);
   L();
   nogard(n-1);
}
```

Still need implementation of "nogard"

Backwards dragon curve

To get "nogard" string from "dragon" string
* reverse the string
* switch L and R

```
dragon:  F L F L F R F
reverse: F R F L F L F
nogard:  F L F R F R F
```

```
nogard(int n)
{
   if (n == 0) { F(); return; }
   dragon(n-1);
   R();
   nogard(n-1);
}
```

Note: easy to make this program nonrecursive
(replace recursive call with jump to beginning)

Alternate "dragon"

Replace call to "nogard"
    by nonrecursive version

```
dragon(int n)
{
   int k;
   if (n == 0)  { F(); return; }
   dragon(n-1);
   L();
   for (k = n-2; k >= 0; k--)
      {
         dragon(k);
         R();
      }
   F();
}
```

F L F L F R F L F L F R F R F

F L F L F R F L F L F R F R F

Points out self-similarities in curve

Nonrecursive dragon curve

```
.
.                          D(3)
.
.             D(2)                       N(2)
.
.     D(1)          N(1)          D(1)          N(1)
.
.  D(0)   N(0)   D(0)   N(0)   D(0)   N(0)   D(0)   N(0)
.
.   F  L  F  L  F  R  F  L  F  L  F  R  F  R  F
```

To write down the whole dragon curve sequence

* first, put "F" in every other space
* put "L", "R" (alternating)
  in every other remaining space
* continue until done

```
.   F     F     F     F     F     F     F     F
.   F  L  F     F  R  F     F  L  F     F  R  F
.   F  L  F  L  F  R  F     F  L  F  R  F  R  F
.   F  L  F  L  F  R  F  L  F  L  F  R  F  R  F
```

Like Towers of Hanoi (see Sedgewick, section 5.2)
    requires too much storage (how much?)
    "ruler function" connects to binary numbers
    Details? [challenge for the bored]