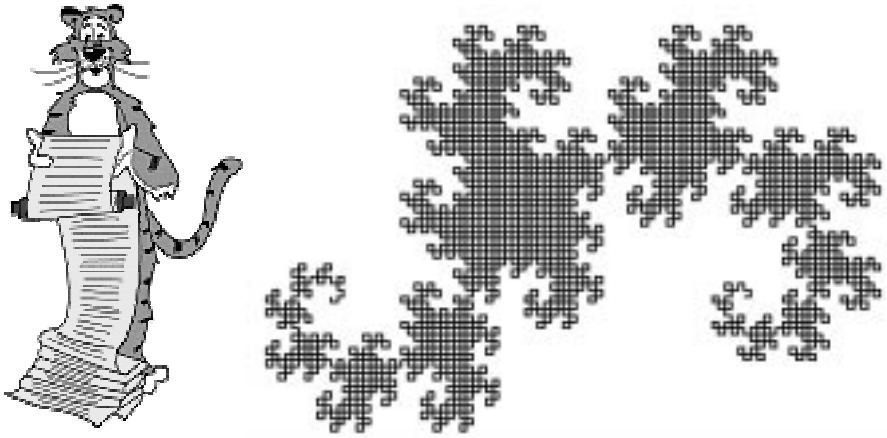
Lecture P7: Advanced Recursion



Mathematical Induction

Mathematical induction.

- Powerful and general proof technique in discrete mathematics.
- To prove a theorem true for all integers $N \geq 0$:
  - Base case: Prove it to be true for N = 0.
  - Induction step: Assuming it is true for all k < N, prove it is true for N.

Theorem: $0 + 1 + 2 + 3 + \ldots + N = N(N+1) / 2$ for all $N \geq 0$.

Proof: (by mathematical induction)

- Base case (N = 0).
  - $0 = 0(0+1) / 2$.
- Induction step.
  - Assume $0 + 1 + 2 + \ldots + k = k(k+1) / 2$ for all $0 \leq k < N$.
  - $0 + 1 + 2 + \ldots + N-1 + N = (0 + 1 + 2 + \ldots + N-1) + N$
    $= ((N-1) N /2) + N$
    $= N (N+1) / 2$

Mathematical Induction and Recursion

Mathematical induction and programming.

- Prove correctness of recursive functions.

0 + 1 + 2 + . . . + n

```
int sum(int n) {
  if (n == 0) return 0;
  else return n + sum(n-1);
}
```

- Find alternate non-recursive computation.

0 + 1 + 2 + . . . + n

```
int sum(int n) {
  return (n * (n+1)) / 2;
}
```


Overview

What is recursion?

- When one function calls ITSELF directly or indirectly.

Why learn recursion?

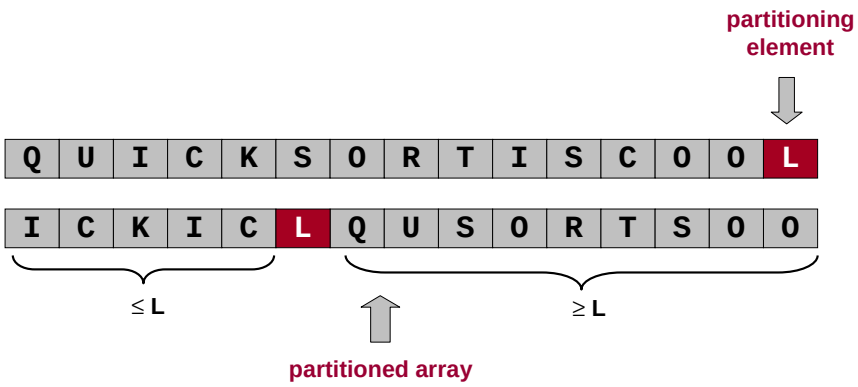
- New mode of thinking.
- Powerful programming tool to solve a problem by breaking it up into one (or more) smaller problems of similar structure.
  - *" Divide et impera "*
  - *" Veni, vidi, vici "*

Julius Caesar (100BC - 44BC)

Quicksort

Quicksort.

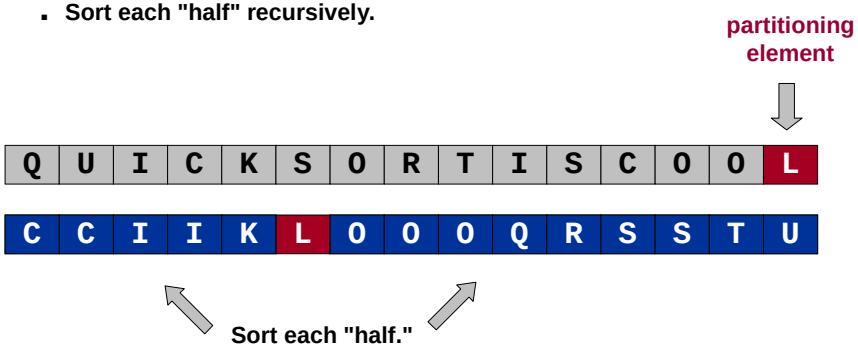
- Partition array so that:
  - some partitioning element `a[m]` is in its final position
  - no larger element to the left of `m`
  - no smaller element to the right of `m`



Quicksort

Quicksort.

- Partition array so that:
  - some partitioning element `a[m]` is in its final position
  - no larger element to the left of `m`
  - no smaller element to the right of `m`
- Sort each "half" recursively.



Quicksort

Quicksort.

- Partition array so that:
  - some partitioning element `a[m]` is in its final position
  - no larger element to the left of `m`
  - no smaller element to the right of `m`
- Sort each "half" recursively.

quicksort.c (see Sedgewick Program 7.1)

```
void quicksort(char a[], int left, int right) {
  int m;
  if (right > left) {
     m = partition(a, left, right);
     quicksort(a, left, m - 1);
     quicksort(a, m + 1, right);
  }
}
```

base case???

Quicksort

Quicksort.

- Partition array so that:
  - some partitioning element `a[m]` is in its final position
  - no larger element to the left of `m`
  - no smaller element to the right of `m`
- Sort each "half" recursively.
- How do we partition efficiently?
  - N - 1 comparisons
  - straightforward with auxiliary array
  - better solution: uses "no" extra space!

Quicksort : Implementing Partition

partition (see Sedgewick Program 7.2)

```
int partition(char a[], int left, int right) {
  int i = left-1;     /* left to right pointer */
  int j = right;      /* right to left pointer */

  while(1) {
     while (a[++i] < a[right])
        ;
     while (a[right] < a[--j])
       if (j == left)
          break;

     if (i >= j)
        break;
     swap(a, i, j);
  }

  swap(a, i, right);
  return i;
}
```

find element on left to swap

look for element on right to swap, but don't run off end

pointers cross

swap partition element

Quicksort : Implementing Partition

main( )

```
#include <stdio.h>
#define N 14

int main(void) {
   char a[] = "pseudomythical";
   printf("Before: %s\n", a);
   quicksort(a, 0, N-1);
   printf("After:  %s\n", a);
   return 0;
}
```

swap( )

```
void swap(char a[], int i, int j) {
   char t;
   t = a[i]; a[i] = a[j]; a[j] = t;
}
```


Quicksort : Performance

Quicksort vs. Insertion sort.

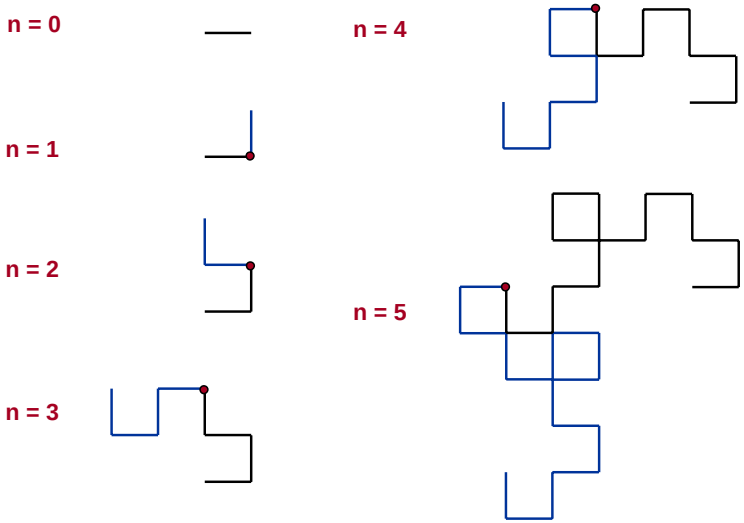
- Good algorithms more powerful than supercomputers.

	Insertion Sort			*Quicksort*		
computer	thousand	million	billion	thousand	million	billion
home pc	instant	2 hour	310 years	instant	0.3 sec	6 min
super	instant	1 sec	1.6 weeks	instant	instant	instant

Stay tuned: Lecture T5.

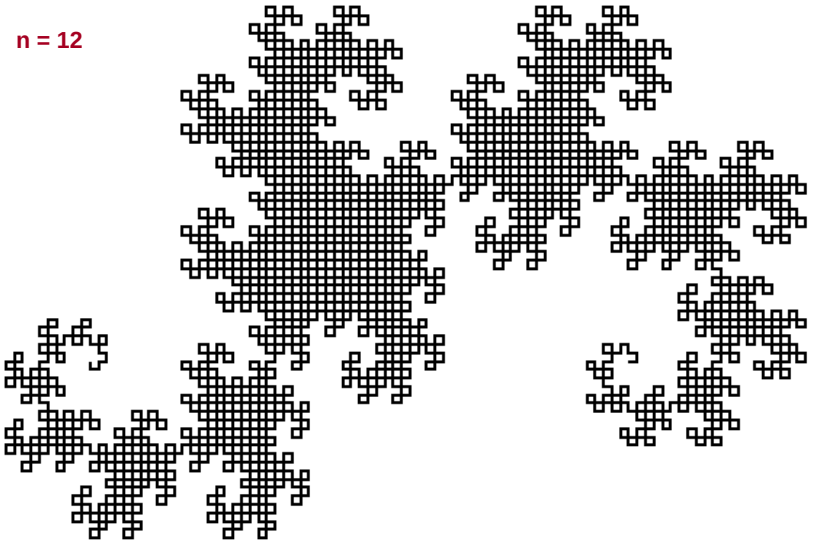
Dragon (Jurassic Park) Curve

Fold a wire in half n times. Unfold to right angles.



Dragon (Jurassic Park) Curve

n = 12



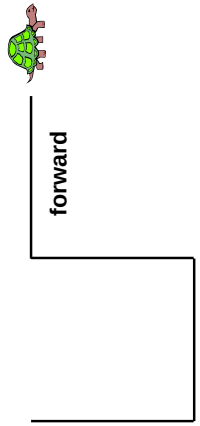
Drawing a Dragon Curve

Use simple "turtle graphics."

- F: move turtle forward one step (pen down).
- L: turn left 90°.
- R: turn right 90°.

Example.

- F L F L F R F



Drawing a Dragon Curve

Use simple "turtle graphics."

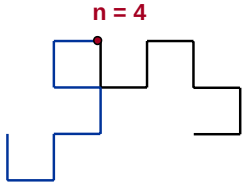
- F: move turtle forward one step (pen down).
- L: turn left 90°.
- R: turn right 90°.

Example.

- dragon(0): F
- dragon(1): F L F
- dragon(2): F L F L F R F
- dragon(3): F L F L F R F L F L F R F R F
- dragon(4): F L F L F R F L F L F R F R F L F L F L F R F R F L F R F R F

dragon(3) nogard(3)

n = 4



"inverted" dragon(3):
reverse string, switch L and R

Recursive Dragon Curve Program

A dragon curve of order n is:

- Dragon curve of order n-1.
- Move left.
- Inverted dragon curve of order n-1.
  - backwards, switch L and R

dragon( )

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

drawing in PostScript

```
void F(void) {
  printf("10 0 rlineto\n");
}

void L(void) {
  printf("90 rotate\n");
}

void R(void) {
  printf("-90 rotate\n");
}
```

Need implementation of `nogard( )`.

Drawing a Dragon Curve

Observation: `nogard(n)` is same as `dragon(n)`, except middle move is R not L

Justification: by definition, `nogard(n)` is inverted `dragon(n)`.

- `dragon(n)` = `dragon(n-1)` L `nogard(n-1)`
- `nogard(n)` = `dragon(n-1)` R `nogard(n-1)`

inverted nogard(n-1)

inverted dragon(n-1)

nogard( )

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

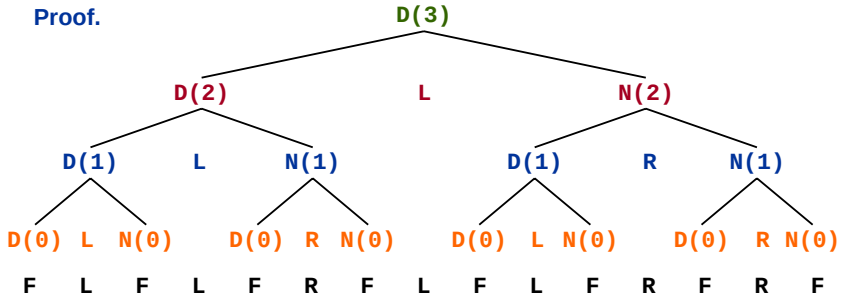

Nonrecursive Dragon Curve

To write down the whole dragon curve sequence:

1. Put F in every other space.
2. Put L, R (alternating) in every other remaining space.
3. Repeat Step 2 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

Proof.



Drawback: requires excessive memory.

Sequential Dragon Curve

The dragon curve and binary integers.

- The k^{th} turtle turn (ignore Fs) depends on the bit to the left of the rightmost 1 in the binary representation of k.
  - L if bit = 0, R if bit = 1

Proof: (by induction on order of curve)

- Base case: dragon(1) = F L F. ✓
- Assume true for dragon(n), and consider dragon(n+1).
- Recall: only difference between top and bottom halves is their middle moves.

Dec	Bin						D(n)	L	N(n)	D(n+1)
1	0	0	0	0	1	L				
2	0	0	0	1	0	L				
3	0	0	0	1	1	R				
4	0	0	1	0	0	L		L		
5	0	0	1	0	1	L				
6	0	0	1	1	0	R				
7	0	0	1	1	1	R				
8	0	1	0	0	0	L		L		L
9	0	1	0	0	1	L				
10	0	1	0	1	0	L				
11	0	1	0	1	1	R				
12	0	1	1	0	0	R	D(n)	R	N(n)	N(n+1)
13	0	1	1	0	1	L				
14	0	1	1	1	0	R				
15	0	1	1	1	1	R				

Consequence: simple iterative algorithm that requires little storage.

Sequential Dragon Curve

The dragon curve and binary integers.

- The k^{th} turtle turn (ignore Fs) depends on the bit to the left of the rightmost 1 in the binary representation of k.

dragon-nonrecursive.c

```
void dragon(int m) {
   int k;
   F();
   for (k = 1; k <= m-1; k++) {
      if (k & ((k^(k-1))+1))
         R();
      else
         L();
      F();
   }
}
```

m need not be power of 2

check bit to the left of rightmost 1

Dec	Bin					
1	0	0	0	0	1	L
2	0	0	0	1	0	L
3	0	0	0	1	1	R
4	0	0	1	0	0	L
5	0	0	1	0	1	L
6	0	0	1	1	0	R
7	0	0	1	1	1	R
8	0	1	0	0	0	L
9	0	1	0	0	1	L
10	0	1	0	1	0	L
11	0	1	0	1	1	R
12	0	1	1	0	0	R
13	0	1	1	0	1	L
14	0	1	1	1	0	R
15	0	1	1	1	1	R