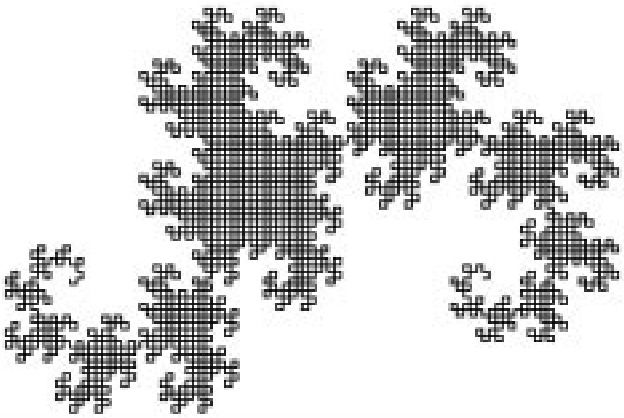
Lecture P7: Advanced Recursion



Overview

What is recursion?

- When one function calls ITSELF directly or indirectly.

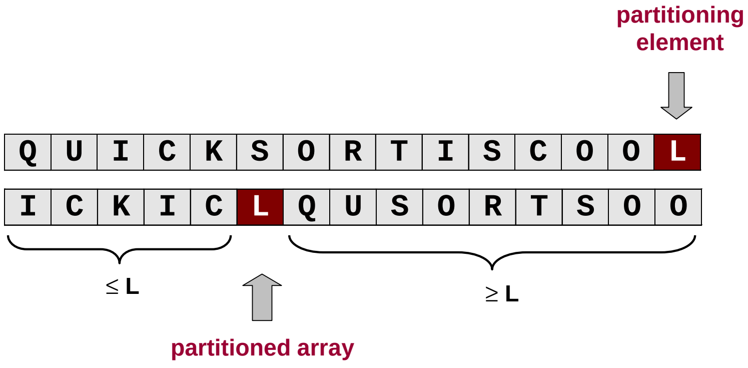
Today.

- Quicksort.
- Dragon curve.
- Travelling salesperson problem.

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`



partitioning element

Q	U	I	C	K	S	O	R	T	I	S	C	O	O	L
I	C	K	I	C	L	Q	U	S	O	R	T	S	O	O

≤ L

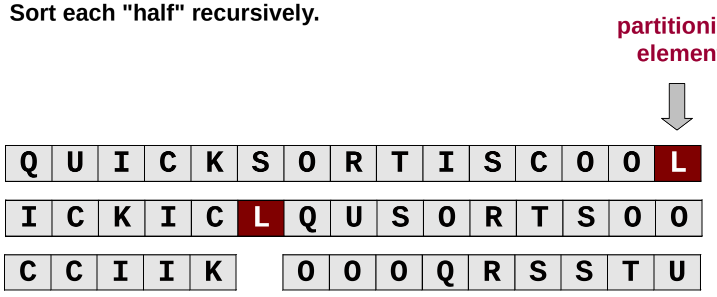
≥ L

partitioned array

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.



partitioning element

Q	U	I	C	K	S	O	R	T	I	S	C	O	O	L
I	C	K	I	C	L	Q	U	S	O	R	T	S	O	O
C	C	I	I	K		O	O	O	Q	R	S	S	T	U

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(int left, int right) {
  int m;
  if (right > left) {
    m = partition(left, right);
    quicksort(left, m - 1);
    quicksort(m + 1, right);
  }
}
```

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
  - easy with auxiliary array
  - better solution: use no extra space!

Quicksort : Implementing Partition

partition (see Sedgewick Program 7.2)

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

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

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

  swap(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( )

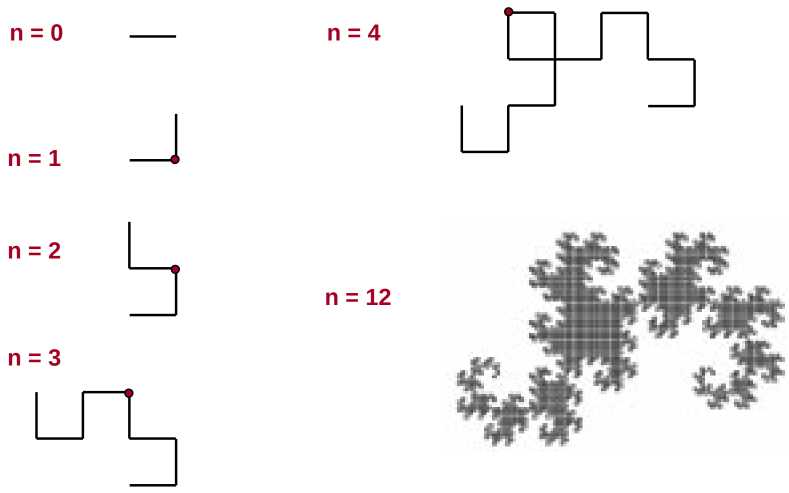
```
#define N 14
char a[] = "pseudomythical";

int main(void) {
   printf("%s", a);
   quicksort(0, N-1);
   printf("%s", a);
   return 0;
}
```

swap( )

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

Dragon Curve

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



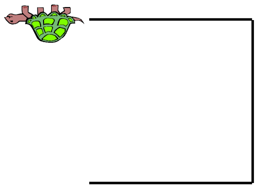
Drawing a Dragon Curve

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



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)

"backwards" 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.
- Dragon curve of order n-1 backwards.

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

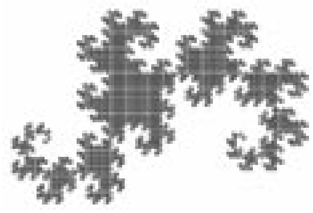
To get nogard(n):

- dragon(2): F L F L F R F
- nogard(2): F L F R F R F

- dragon(3): $\underbrace{\text{F L F L F R F}}_{\text{dragon(2)}}\ \text{L}\ \underbrace{\text{F L F R F R F}}_{\text{nogard(2)}}$
- nogard(3): $\underbrace{\text{F L F L F R F}}_{\text{dragon(2)}}\ \text{R}\ \underbrace{\text{F L F R F R F}}_{\text{nogard(2)}}$

nogard( )

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



Enumerating All Permutations

Enumerate all permutations of a set of elements.

- N elements ⇒ N! possibilities
- If elements named a, b, c, then 6 possible permutations are: abc, acb, bac, bca, cab, cba.

Key idea: permutations of abcde are one of the followig:

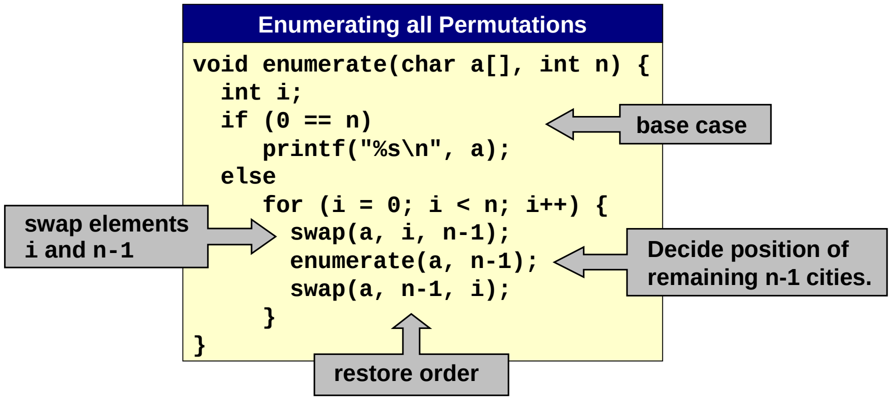
- End with a preceded by one of 4! permutations of bcde.
- End with b preceded by one of 4! permutations of acde.
- End with c preceded by one of 4! permutations of abde.
- End with d preceded by one of 4! permutations of abce.
- End with e preceded by one of 4! permutations of abcd.

Reduces enumerating permutations of N elements to enumerating permutations of N-1 elements.

Enumerating All Permutations

Recursive solution for trying all permutations:

- Array `a[]` to store current permutation.
- Initially `a[] = "abcde"`



Enumerating all Permutations

```
void enumerate(char a[], int n) {
  int i;
  if (0 == n)
     printf("%s\n", a);
  else
     for (i = 0; i < n; i++) {
       swap(a, i, n-1);
       enumerate(a, n-1);
       swap(a, n-1, i);
     }
}
```

base case

swap elements i and n-1

Decide position of remaining n-1 cities.

restore order

Enumerating All Permutations

Recursive solution for trying all permutations:

Enumerating all Permutations

```
#include <stdio.h>

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

void enumerate() { . . . }

int main(void) {
  char a[] = "abcde";
  enumerate(a, 5);
  return 0;
}
```

Unix

```
% a.out
bca
cba
cab
acb
bac
abc
```

Application: Traveling Salesperson Problem

▶

Given N points, find shortest tour connecting them.

- **Brute force: try all N! possible permutations.**

Recursive solution for finding best TSP tour.

- **Store coordinates of points in `a[]`.**
- **Replace `printf()` with `checklength()`.**
- **Takes N! steps.**
- **No computer can run this for N ≥ 100.**
  - **100! > 10^{150}.**

Is there an efficient way to do this computation?