Recursive Problem Solving

Recursion

def'n: A recursive program is one that calls itself
- must have termination condition (so that program can stop calling self)
- each recursive call must be on a smaller problem (if only a little) than the callee's
  - must be smaller by at least some constant

Relatives of Recursion

Closely related to:
- recursive mathematical functions
- inductive proofs
- recursive invariants
- notion of trees
Two ‘Flavors’ of Recursion

- **Little Steps**
  - given the problem in terms of \( n \)
  - recursively solve in terms of \( n -1 \)
  - until termination condition is reached

- **Divide and Conquer**
  - divide the problem into subproblems (typically by dividing solution space in half)
  - conquer subproblems by solving them recursively

---

### Find Maximum in Unsorted List

#### Iterative

```c
int findmax(int a[], int left, int right)
{
    max = a[left];
    for (int i = left+1; i <= right; i++) {
        if (a[i] > max) max = a[i];
    }
    return(max);
}
```

---

### Find Maximum in Unsorted List

#### Recursive: Little Steps

```c
int findmaxR2(int a[], int left, int right)
{
    if (left == right) return a[left];
    return max (a[left],
                findmaxR2(a, left+1, right));
}
```
Find Maximum in Unsorted List

Recursive: Divide and Conquer

```c
int findmaxR(int a[], int left, int right)
    if (left == right)  return a[left]
    int mid = (left+right)/2
    return max(findmaxR(a, left, mid), findmaxR(a, mid+1, right))
```

```c
int max(int i, int j)
    if (i > j) return i
    else return j
```

FindMax Performance Analysis

- Iterative
- Recursive: Little Steps
- Recursive: Divide and Conquer

Definitions

Linear Recursion: function makes 1 recursive call each time it is invoked

Higher Order Recursion: function makes more than one recursive call

Binary Recursion: exactly two recursive calls
Recursion: Divide and Conquer

Divide domain into (typically same-sized) subdomains, continue algorithm in each subdomain

Examples
- Recursive FindMax in Unsorted List, V.1
- Recursive Binary Search of Sorted List

Binary Search

Start with *sorted* list
- Divide the set of items into two parts
- Determine to which of the two parts the search key belongs
- Continue search on that part

Iterative

```c
int search(int a[], int v, int left, int right) {
    while (right >= left) {
        int mid = (left + right)/2;
        if (v == a[mid]) return mid;
        if (v < a[mid])
            right = mid - 1;
        else
            left = mid + 1;
    }
    return -1;
}
```
Binary Search

Recursive

```c
int searchR(int a[], int v, int left, int right)
    if (left > right) return -1;
    int mid = (left + right)/2;
    if (v == a[mid]) return mid;
    if (v < a[mid])
        return searchR(a[], v, left, mid - 1);
    else
        return searchR(a[], v, mid + 1, right);
}
```

Recursion: Little Steps

Given the problem in terms of $n$
recursively solve in terms of $n-1$
until termination condition is reached

Examples
- Recursive FindMax in Unsorted List, V.2
- Fibonacci

Fibonacci Sequence

Definition
- $F(0) = 0; F(1) = 1$
- $F(n) = F(n-1) + F(n-2)$

Find $F(3)$
- $F(3) = F(2) + F(1); F(2) = F(1) + F(0)$
  \[ \Rightarrow \]
- $F(3) = (F(1) + F(0)) + F(1) = (1 + 0) + 1$
Recursive Implementation

Fibonacci Sequence

```c
int FindFib(int i)
{
    if (i < 1) return 0;
    if (i == 1) return 1;
    return FindFib(i-1) + FindFib(i-2);
}
```

- Spectacularly inefficient recursive algorithm
- That is, the recursive algorithm is exponential (O(1.618^i))

Summary: Recursive Methods

- Recursive functions call themselves
- Two flavors of recursive functions:
  - Divide and Conquer
  - Little Steps
- Need tool to analyze complexity
  -> Recurrence Relations