Recursion

RECURSIVE PROGRAM: : one that calls itself

Example:

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

RECURSION:

To compute f(N)

* compute f(o)

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

Analogy

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 $N = 0$
* $0 + 1 + 2 + 3 + \ldots + N$

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

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

Divide and Conquer

Many computations

are naturally expressed as recursive programs

``DIVIDE and CONQUER``

solve a problem by dividing into smaller ones

Ex: for a decreasing function e(int), find

smallest n within [lb,ub] such that e(n)<BOUND

```
int min_n(int lb, int ub)
  /*  e(lb) >= BOUND  &&  e(ub) < BOUND  */
  /*  ub > lb */
  {
    int m;
    if ((ub-lb) == 1) return ub;
    m = (lb + ub) / 2;
    if (e(m) < BOUND ) return min_n(lb, m);
                else return min_n(m, ub);
  }
```

Binary search

Suppose an array A contains N integers,

in increasing order

SEARCH problem: is a given integer v in A?

```
int find (int A[], int lb, int ub, int v)
  /*   A[lb] <= v  &&  A[ub] >= v   */
  /*  ub >= lb */
  {
    int m;
    if (ub == lb) return lb;
    m = (lb + ub) / 2;
    if (A[m] == v) return m;
    if (A[m] < v) return find( A, m+1, ub, v);
         else if (lb == m) return m;
                 else return find( A, lb, m-1, v);
  }
```

Number conversion

Problem: convert a given integer N to base b

(print out the result)

Solution:

```
int convert(int N)
  {
    if (N > 0)
      { convert(N / b); printf("%1d", N % b); }
  }
```

Ex: convert 876 to base 5

175*5 + 1

(35*5 + 0)*5 + 1

((7*5 + 0)*5 + 0)*5 + 1

(((1*5 + 2)*5 + 0)*5 + 0)*5 + 1

((((0*5 + 1)*5 + 2)*5 + 0)*5 + 0)*5 + 1

1*5^4 + 2*5^3 + 0*5^2 + 0*5^1 + 1*5^0

12001

Possible pitfall with recursion

Simple recursive programs
can consume excessive resources

Example: hard way to compute 2^N

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

Recomputes intermediate results

Even "natural" recursive programs may
consume excessive resources

Example 2: Fibonacci numbers

```
int fibonacci(int N)
  {
     if (N == 0) return 1;
     if (N == 1) return 1;
     return (fibonacci(N-2) + fibonacci(N-1));
  }
```

Think recursively

BUT often thinking and writing recusively

is the WAY TO GET IT RIGHT!

"Divide and conquer" -- think recursively

Write recursively

Simplify

Can you?

Is it worth it?

Effort

Correctness