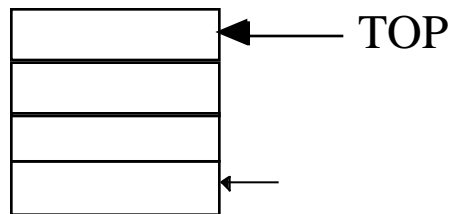# Section 3.4
Recursive Algorithms

A recursive algorithm is one which calls <u>itself</u> to solve "smaller" versions of an input problem.

How it works:

• The current status of the algorithm is placed on a *stack*.

A *stack* is a data structure from which entries can be added and deleted only from one end.

- like the plates in a cafeteria:

PUSH: put a 'plate' on the stack.

POP: take a 'plate' off the stack.

When an algorithm calls itself, the current *activation* is suspended in time and its parameters are **PUSH**ed on a stack.

The set of parameters need to restore the algorithm to its current activation is called an *activation record*.

Example:

> **procedure** *factorial* (n)
> /* make the procedure idiot proof */
>   **if** n < 0 **return** 'error'
>   **if** n = 0 **then return** 1
> **else**
>   **return** n *factorial* (n-1)

The operating system supplies all the necessary facilities to produce:

factorial (3): **PUSH** 3 on stack and call

    factorial (2): **PUSH** 2 on stack and call

        factorial (1): **PUSH** 1 on stack and call

            factorial (0): return 1

        POP 1 from stack and return (1) (1)

    POP 2 from the stack and return (2) [(1) (1)]

POP 3 from the stack and return (3) [(2) [(1) (1)]]

———————————————————

Complexity:

Let f(n) be the number of multiplications required to compute factorial (n).

$$f(0) = 0: \text{the } \textit{initial condition}$$

$$f(n) = 1 + f(n\text{-}1): \text{the } \textit{recurrence equation}$$
_____

Example:

A recursive procedure to find the max of a <u>nonvoid</u> list.

Assume we have a built-in functions called

    • Length which returns the number of elements in a list

    • Max which returns the larger of two values

    • Listhead which returns the first element in a list

Max requires one comparison.

```
procedure maxlist (list)
/* strip off head of list and pass the remainder */

if Length(list) = 1 then
    return Listhead(list)
        else
    return Max( Listhead(list), maxlist
        (remainder of list))
```

The recurrence equation for the number of comparisons required for a list of length n, f(n), is

- $f(1) = 0$
- $f(n) = 1 + f(n-1)$

_____

Example:

If we assume the length is a power of 2:

    • We divide the list in half and find the maximum of each half.

    • Then find the Max of the maximum of the two halves.

> **procedure** _maxlist2_ (list)
> /* a _divide and conquer_ approach */
>
>   **if** Length (list) = 1 **then**
> **return** Listhead(list)
>   **else**
>   a = _maxlist_ (fist half of list)
>   b = _maxlist_ (second half of list)
>   **return** Max{a, b}

Recurrence equation for the number of comparisons required for a list of length n, f(n), is

- $f(1) = 0$
- $f(n) = 2\, f(n/2) + 1$

• There are two calls to *maxlist* each of which requires f(n/2) operations to find the max.

• There is one comparison required by the Max function.

If n = 16:

Divide list in half

Divide list in half   Divide list in half

X X|X X|X X|X X|X X|X X|X X|X X
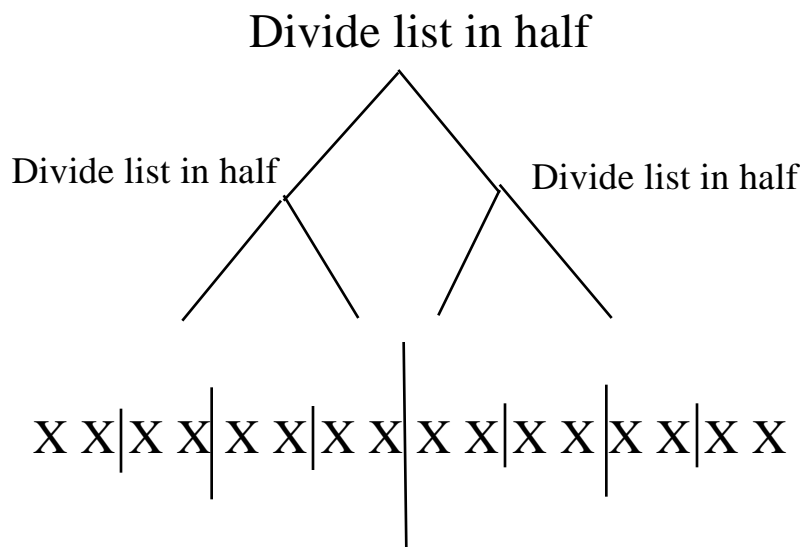
f(16) = 2 f(8) + 1
f(8) = 2 f(4) + 1
f(4) = 2 f(2) + 1
f(2) = 2 f(1) + 1

So

f(2) = 1,
f(4) = 2 (1) + 1 = 3
f(8) = 2 (3) + 1 = 7
f(16) = 2(7) + 1 = 15
f(n)?