WHAT DOES THIS CODE DO?

```cpp
int message () {
    int x = 1;
    cout << "we are in message: " << x << "!" << endl;
    message();
    return [x];
}
```

A function is recursive if it calls itself.

EXAMPLE: FACTORIAL FUNCTION

The factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n.

\[ n! = n \times (n-1) \times \ldots \times 2 \times 1 \]

Note: 0! = 1

Factorial is an example of a mathematical function that can be defined recursively.

\[ (n-1)! = (n-1) \times \ldots \times 2 \times 1 \]

Note: (n-1)! = (n-1)\times\ldots\times2\times1

n! = n \times (n-1)!
RECURSIVE FACTORIAL IMPLEMENTATION

```c
// Compute n!  // i.e. the product of the first n integers
int factorial(int n)
{
    int result;
    if (n <= 1) {
        result = 1;
    } else {
        result = n * factorial(n - 1);
    }
    return result;
}
```

**Trace:**

```
factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
1
2 * 1
3 * 2
4 * 6
24
```

WHAT MAKES RECURSION WORK?

A recursive definition has two parts

- One or more recursive cases where the function calls itself
- One or more base cases that return a result without a recursive call

**Rules:**

- There must be at least one base case
- Every recursive case must make progress towards a base case

SEARCH REVISITED

Is there a more efficient way than linear search to search for target x in the sorted array?

Search(A, 8, 45)

<table>
<thead>
<tr>
<th>Initial array</th>
<th>Sorted array</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 12 -5 6 142 21 -17 45</td>
<td>-17 -5 3 6 12 21 45 142</td>
</tr>
</tbody>
</table>

RECURSIVE BINARY SEARCH

**Key Idea:**

Do a little bit of work, and make recursive call to do the rest

**Details:**

- Look at midpoint, and decide which half of the range is of interest
- Use binary search again (recursively) to find value in reduced range

We can also think of this as a divide-and-conquer strategy

(...but not all recursion is based on divide-and-conquer)
**EXAMPLE**

Search(A, 8, 45)

```
| -17 | -5 | 3 | 6 | 12 | 21 | 45 | 142 |
```

```
12 21 45 142
```

```
45 142
```

**BINARY SEARCH CODE**

```c
int BinarySearch(int A[], int first, int last, int target)
{
    int middle;
    if (first > last) return -1;
    middle = (first + last )/2;
    if [A[middle] == target] return middle;
    if [target > A[middle]]
        return binarySearch(A, middle+1, last, target);
    else
        return binarySearch(A, first, middle-1, target);
}
```

*Where are the base cases?*

*How does the call make progress?*

**ADD DEBUG STATEMENTS**

```c
int BinarySearch(int A[], int first, int last, int target)
{
    int middle;
    middle = (first + last )/2;
    cout << "Searching A[ " << first << " .. " << last << " ]" << endl;
    if (first > last) {
        cout << "First is larger than Last -- item not found!\n" ;
        return -1;
    }
    if [A[middle] == target] {
        cout << "Middle " << middle << " found!\n" ;
        return middle;
    }
    if [target > A[middle]]
        return binarySearch(A, middle+1, last, target);
    else
        return binarySearch(A, first, middle-1, target);
}
```

**TRACE EXECUTION**

```
BinarySearch(A, 8, 45)
```

```
17 -5 3 6 12 21 45 142
```

```
12 21 45 142
```

```
45 142
```

*What is the maximum number of calls possible?*

**SORTING REVISITED**

Can we sort more efficiently than bubble sort?

Merge Sort is a recursive sorting procedure that employs a 
divide-and-conquer technique

*Base case(s)?*

*How does the call make progress?  What gets returned at the 
end of each call?*

**MERGE SORT**

To sort an array of n elements:

If n<2 then the array is already sorted,
Stop now.

Otherwise,
Sort the left half of the array.
Sort the right half of the array.
Merge the now-sorted left and right halves.
MERGE SORT ALGORITHM

```c
void merge_sort (int A[], int low, int hi)
{
    // base case
    if your array has only one element
        return
    // recursive case
    else
        // Sort the 2 halves of the array as follows:
        set mid = (lo + hi) / 2
        merge_sort (A, lo, mid)
        merge_sort (A, mid + 1, hi)
        // combine the two halves
        merge (A, lo, mid, mid + 1, hi)
}
```

MERGING TWO SORTED ARRAYS

Efficient procedure

```
  -5  3  12  45
-17  6  21 142
```

OUR MERGE PROBLEM

Need temporary storage
Need 3 indices to point to correct locations in arrays

```
A
  | -5 3 12| 45 -17 | 6 21 | 142 |
  |  lo1  |     |     |  hi1 |
  |  hi2  |     |     |  lo2 |
  |  leftIndex | rightIndex |
  | spare |
```

MERGE ALGORITHM

```c
void merge(int A[], int low1, int hi1, int low2, int hi2, int spare[])
{
    initialize the three indices
    // Initially need to grab numbers from both arrays
    while there are still more elements to look at in both halves of the array:
        update the spare array with the smallest element from either half of the array
        update indices
    // At this point, all of the elements in one half of the array
    // have been copied into spare.
    if there are more elements in the lower half of A:
        copy the elements in the lower half of A into spare and update indices
    else if there are more elements in the upper half of A:
        copy the elements in the upper half of A into spare and update indices
    // Now, all of the elements of A are in sorted order in array spare.
    copy the elements of spare into A.
}
```

EFFICIENCY OF SORTING ALGORITHMS

Good algorithms are better than supercomputers

```
| Insert | Select | Bubble | Shell | Merge | Heap | Quick | RQ
|-------|--------|--------|-------|-------|------|-------|---
 inserted sorted sorted sorted sorted sorted sorted sorted sorted
 sorted inserted inserted inserted inserted inserted inserted inserted
 sorted sorted inserted inserted inserted inserted inserted inserted
 sorted sorted sorted inserted inserted inserted inserted inserted inserted
 sorted sorted sorted sorted inserted inserted inserted inserted inserted
 sorted sorted sorted sorted sorted inserted inserted inserted inserted
 sorted sorted sorted sorted sorted sorted inserted inserted inserted
```

TOWERS OF HANOI

```
```

See sorting-algorithms.com
How would you describe this fractal pattern?
How could you generate it?

SUMMARY
A function is recursive if it calls itself

Popular recursive algorithms:
   Binary search
   Merge sort