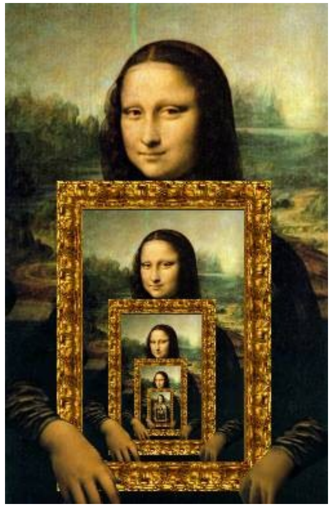
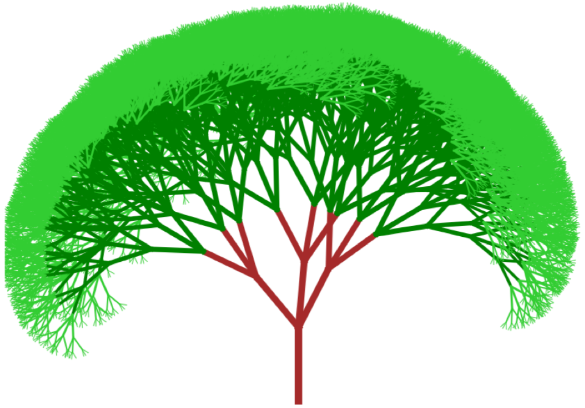
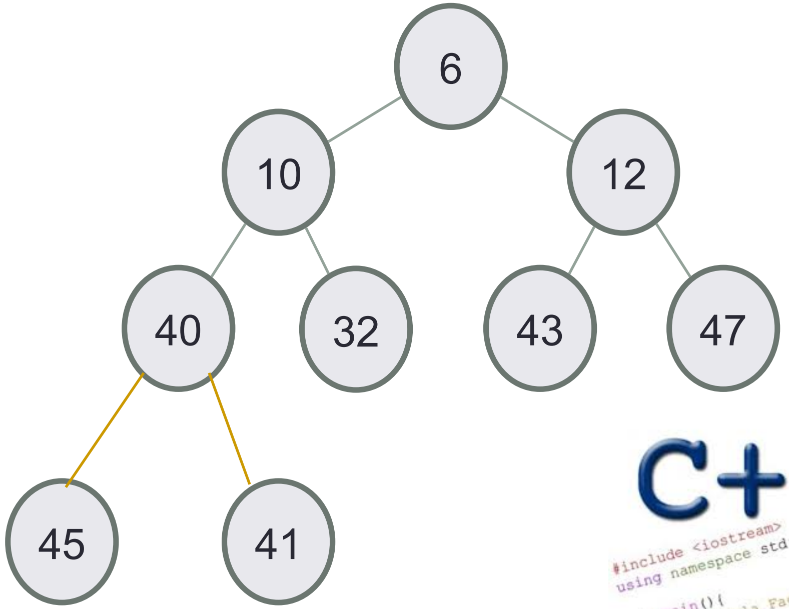
MORE ON RECURSION



Problem Solving with Computers-I





```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;

}
```

Thinking recursively !

```
int fac(int N){

    if (N <= 1)
        return 1;

    else{
        int rest = fac(N-1);
        return rest * N;
}
```

Base case

Recursive case

Human: Base case and 1 step

Computer: Everything else

Thinking recursively !

```
int fac(int N){

    if (N <= 1)
        return 1;

    else
        return fac(N-1)* N;
}
```

Base case

Recursive case

(shorter version)

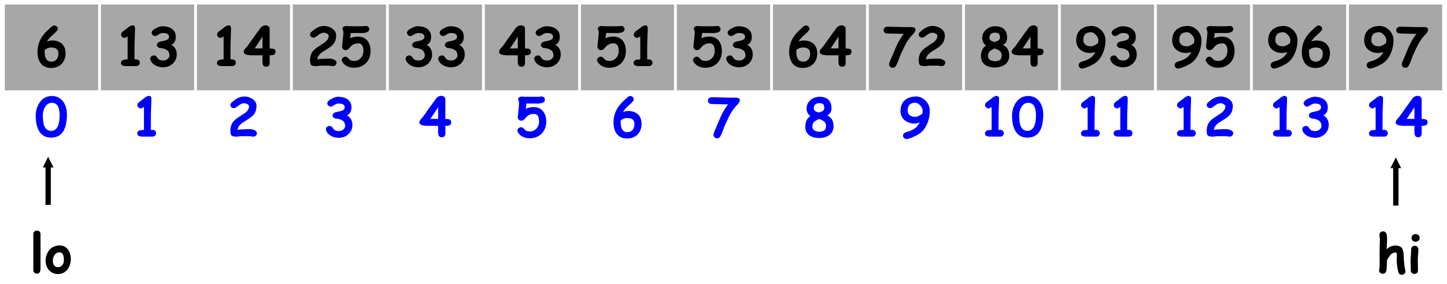
Human: Base case and 1 step

Computer: Everything else

Binary Search: Efficient search in a sorted array

- Binary search. Given `value` and sorted array `v[]`, find index `i` such that `v[i] == value`, or return -1 indicating that no such index exists.
- Invariant. Algorithm maintains `v[lo]` ≤ `value` ≤ `v[hi]`.

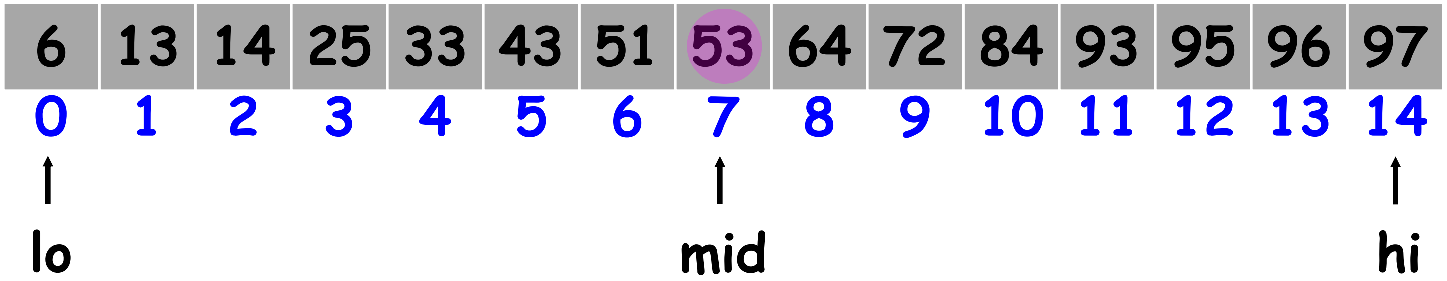
- Ex. Binary search for 33.



6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search

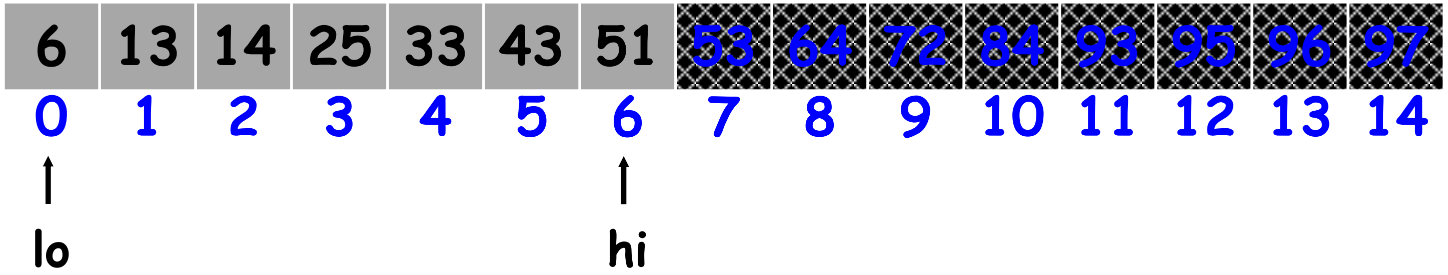
- Ex. Binary search for 33.



6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lo							mid							hi

Binary Search

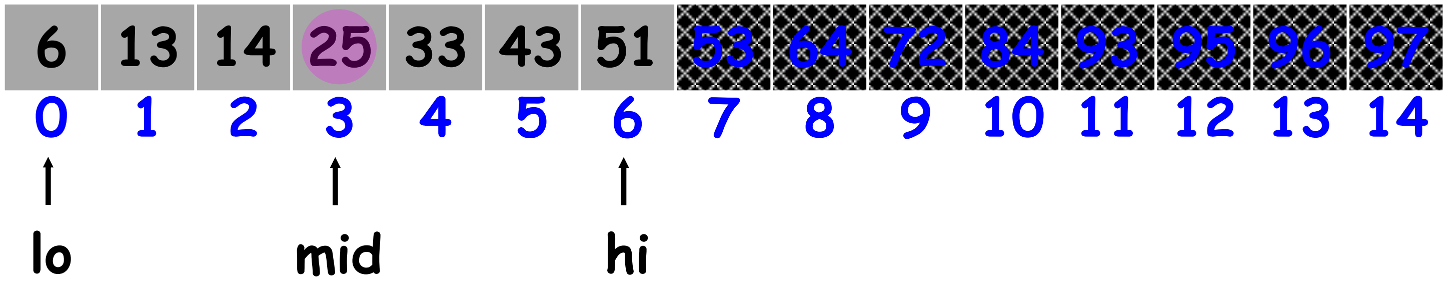
- Ex. Binary search for 33.



6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search

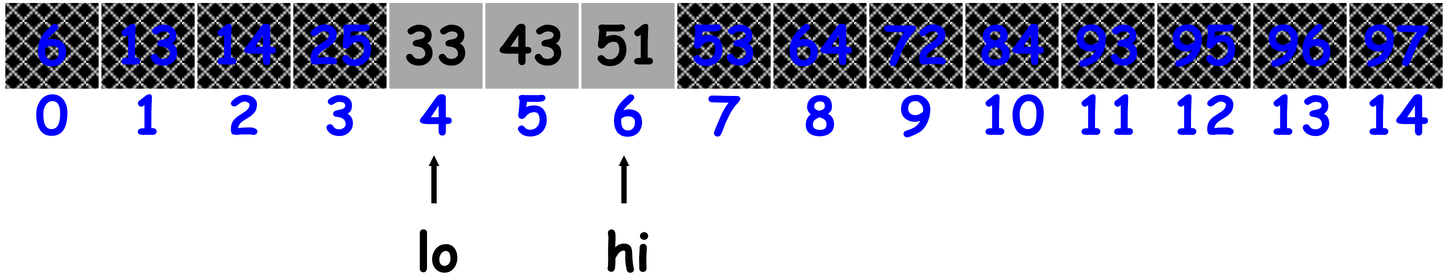
- Ex. Binary search for 33.



6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑			↑			↑								
lo			mid			hi								

Binary Search

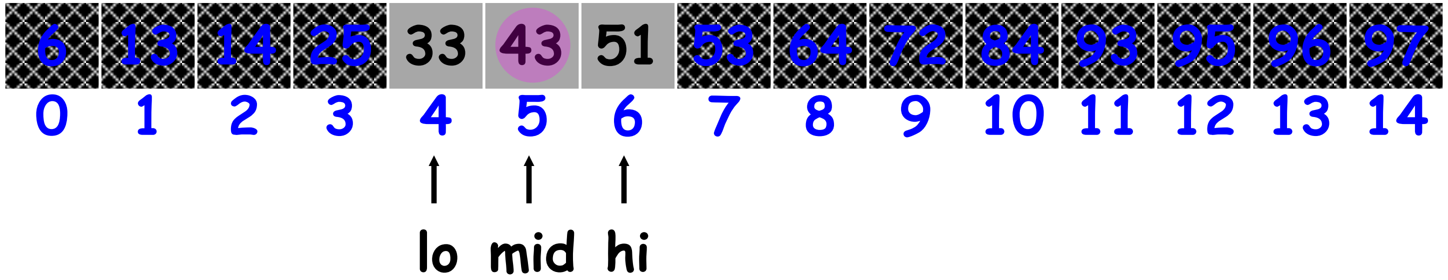
- Ex. Binary search for 33.



6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
				lo		hi								

Binary Search

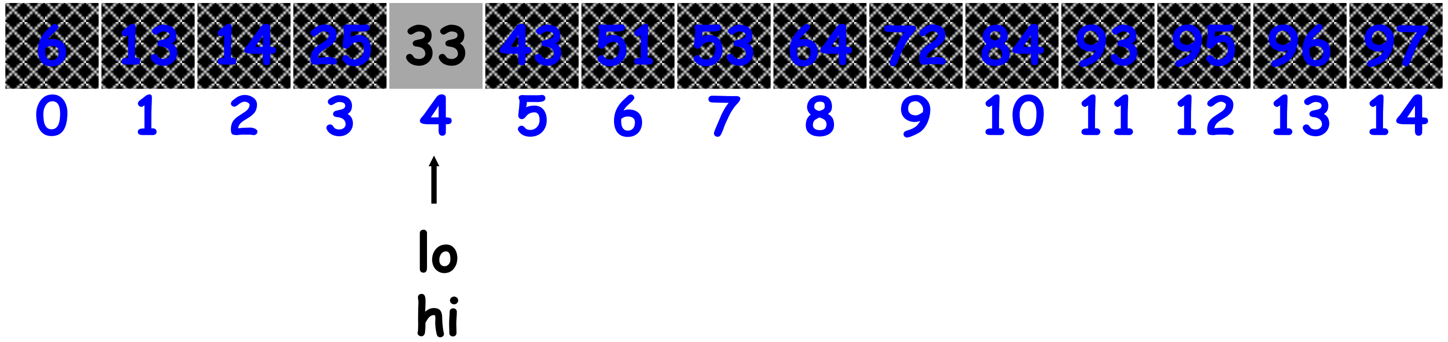
- Ex. Binary search for 33.



6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
				lo	mid	hi								

Binary Search

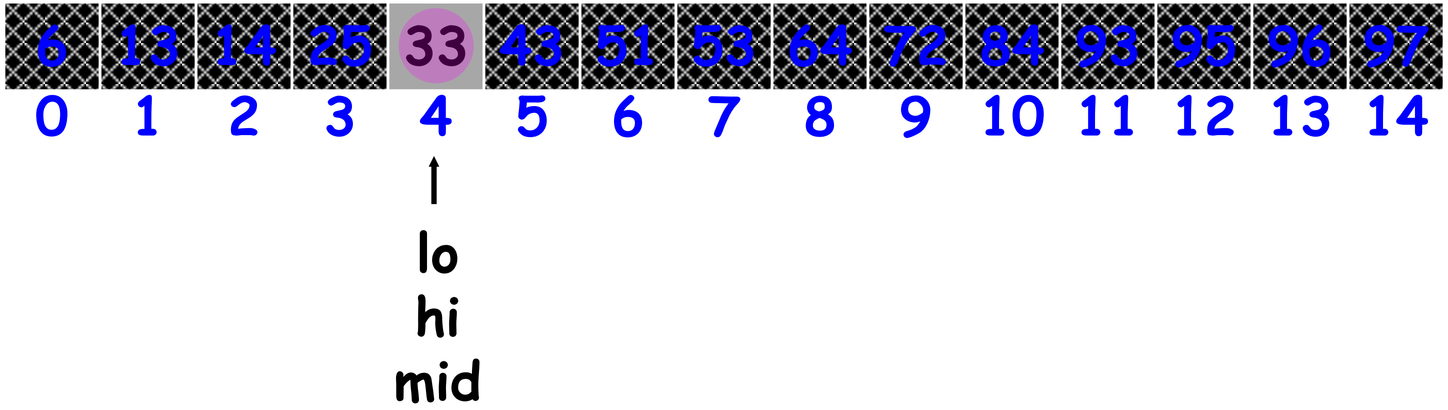
- Ex. Binary search for 33.



6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search

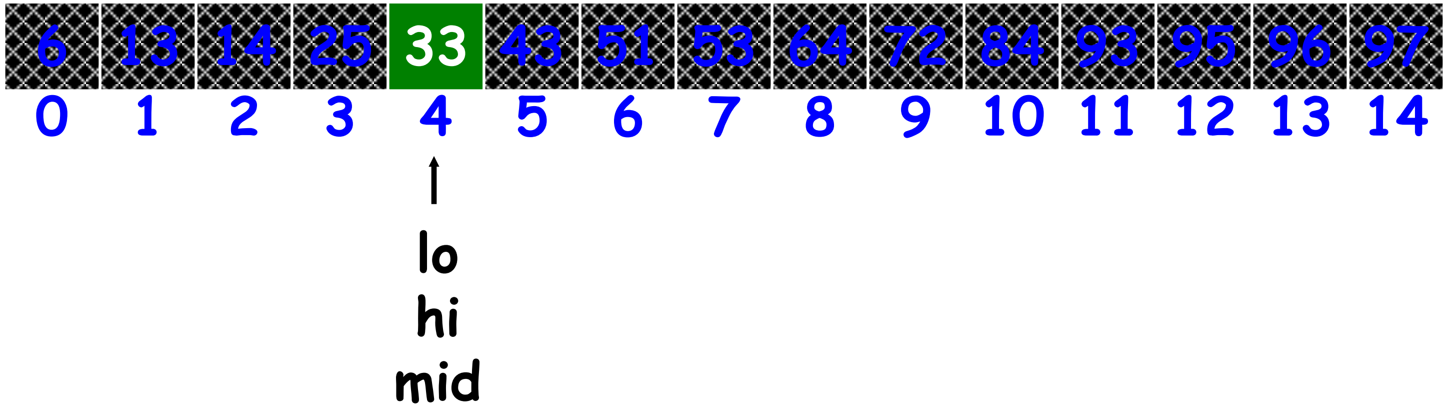
- Ex. Binary search for 33.



6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search

- Ex. Binary search for 33.



6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

lo
hi
mid

Write the recursive implementation of Binary search

```
int binarySearch(int v[], int value, int lo, int hi);
```

Which of the following is a valid base case?

```
int binarySearch(int v[], int value, int lo, int hi){
```

A:

```
if(hi<=lo){
    return -1;
}
```

B:

```
int mid = (lo + hi)/2;
if(v[mid] == value){
   return mid;
}
```

C: Both A and B

Fill in the blanks

```
int binarySearch(int v[], int value, int lo, int hi){
    if(hi<lo)
        return -1;
    int mid = (lo + hi)/2;
    if(v[mid] == value)
        return mid;
    if(v[mid] < value){
        binarySearch(v, value, _____, hi);
    }

}
```

```
A: lo
B: mid - 1
C: mid
D: mid + 1
E: hi
```

Is this a correct implementation?

A: Yes

B: No

```
int binarySearch(int v[], int value, int lo, int hi){
    if(hi<lo)
        return -1;
    int mid = (lo + hi)/2;
    if(v[mid] == value)
        return mid;
    if(v[mid] < value){
        binarySearch(v, value, mid + 1, hi);
    }else{
        binarySearch(v, value, lo, mid - 1);
     }
}
```

Searching a linked list

Given a linked list, implement a recursive search function

- Return true if a given value is present in the linked list

- Otherwise return false

Recursive function to free nodes in a linked list

Given a linked list, implement a recursive function to delete all the nodes in the linked list

Behind the curtain…

```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```

```
cout<<fac(1);
```

Result: **1**

The base case !

Behind the curtain…

```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```

```
fac(5)
```

Behind the curtain…

```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```

$$\overbrace{\texttt{5 * fac(4)}}^{\texttt{fac(5)}}$$

Behind the curtain…

```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```

$$\overbrace{5 * \overbrace{4 * \texttt{fac(3)}}^{\texttt{fac(4)}}}^{\texttt{fac(5)}}$$

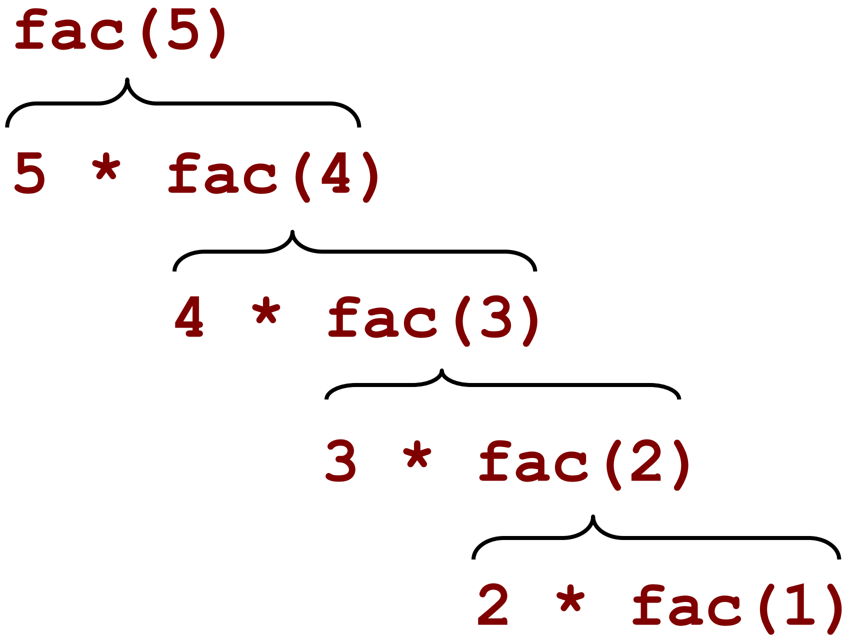
Behind the curtain…

```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```

```
fac(5)

5 * fac(4)

    4 * fac(3)

        3 * fac(2)
```

Behind the curtain…

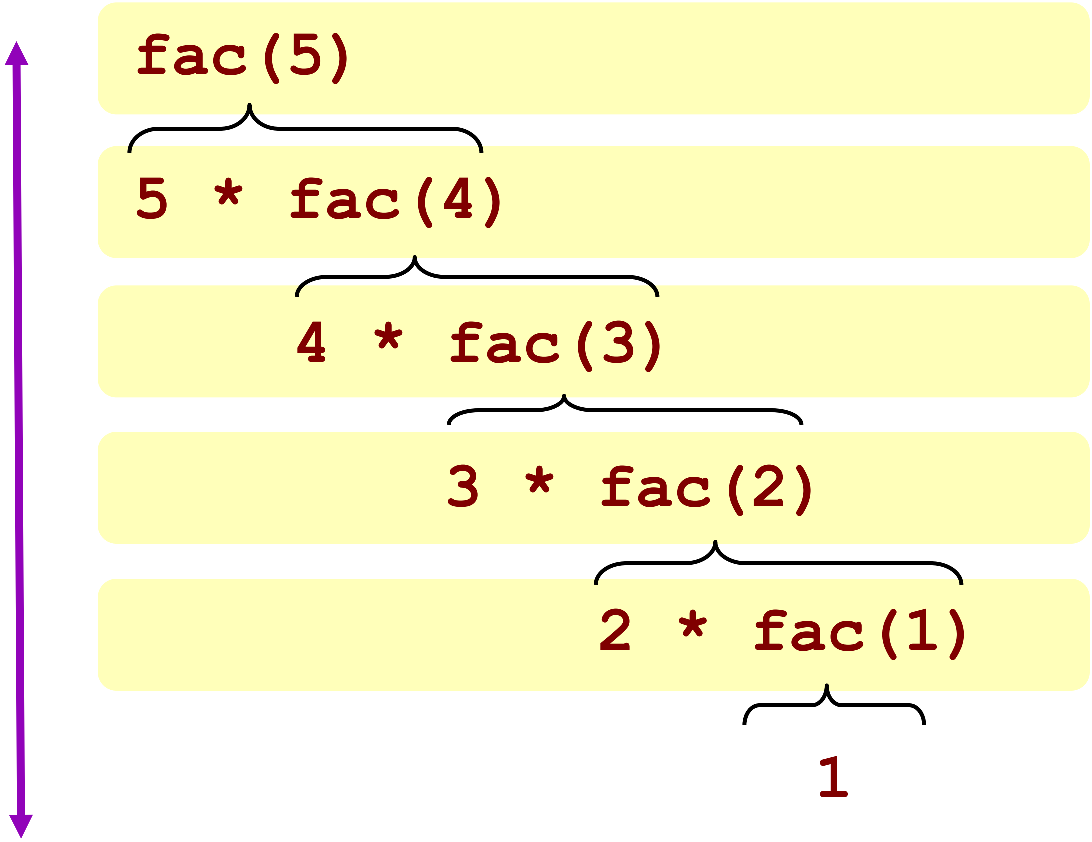
```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```



```
fac(5)
5 * fac(4)
    4 * fac(3)
        3 * fac(2)
            2 * fac(1)
```

Behind the curtain…

```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```



fac(5)

5 * fac(4)

4 * fac(3)

3 * fac(2)

2 * fac(1)

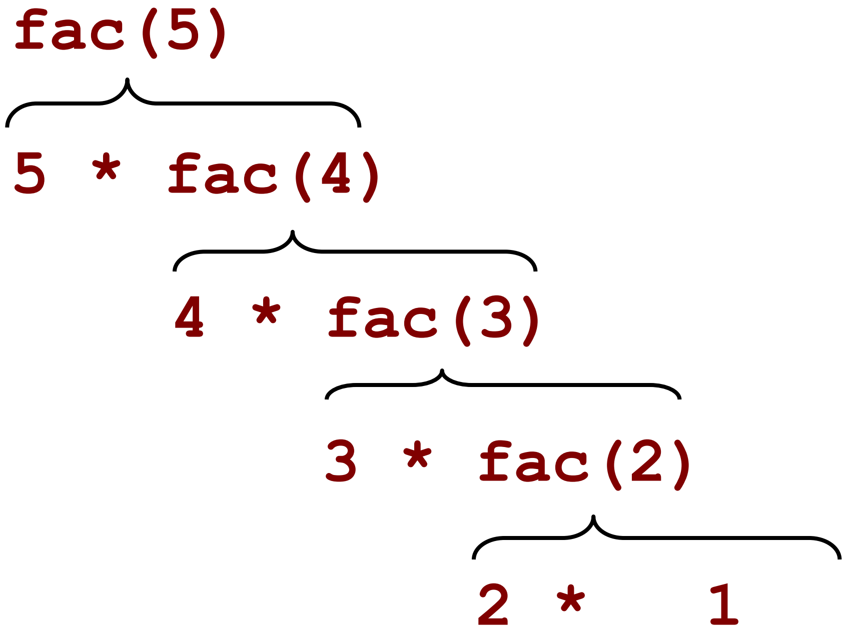
1

"The Stack"

Remembers all of the individual calls to `fac`

Behind the curtain…

```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```



fac(5)

5 * fac(4)

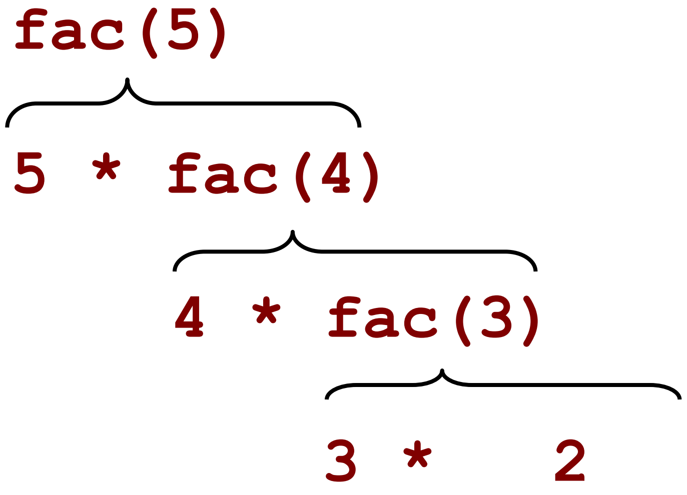
4 * fac(3)

3 * fac(2)

2 * 1

Behind the curtain…

```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```



fac(5)

5 * fac(4)

4 * fac(3)

3 *   2

Behind the curtain…

```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```

$$\overbrace{5 * \overbrace{4 * \quad 6}^{\texttt{fac(4)}}}^{\texttt{fac(5)}}$$

Behind the curtain…

```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```

$$\overbrace{\texttt{5 * 24}}^{\texttt{fac(5)}}$$

Behind the curtain…

```
int fac(int N){

    if (N <= 1)
        return 1;
    else
        return N * fac(N-1);
}
```

`fac(5)`

Result: 120