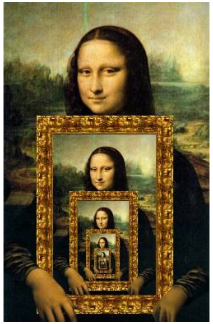
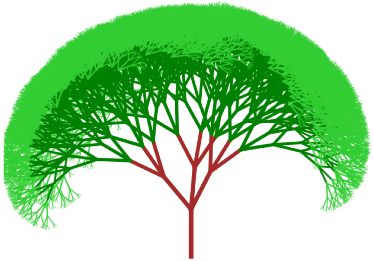
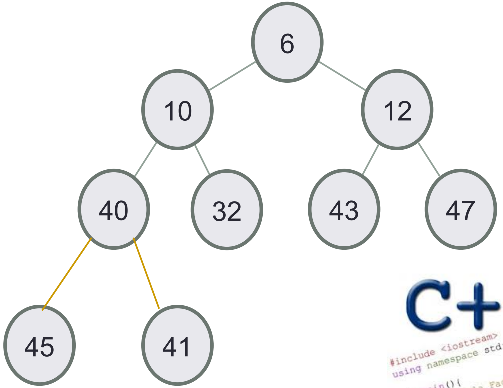
MORE ON RECURSION

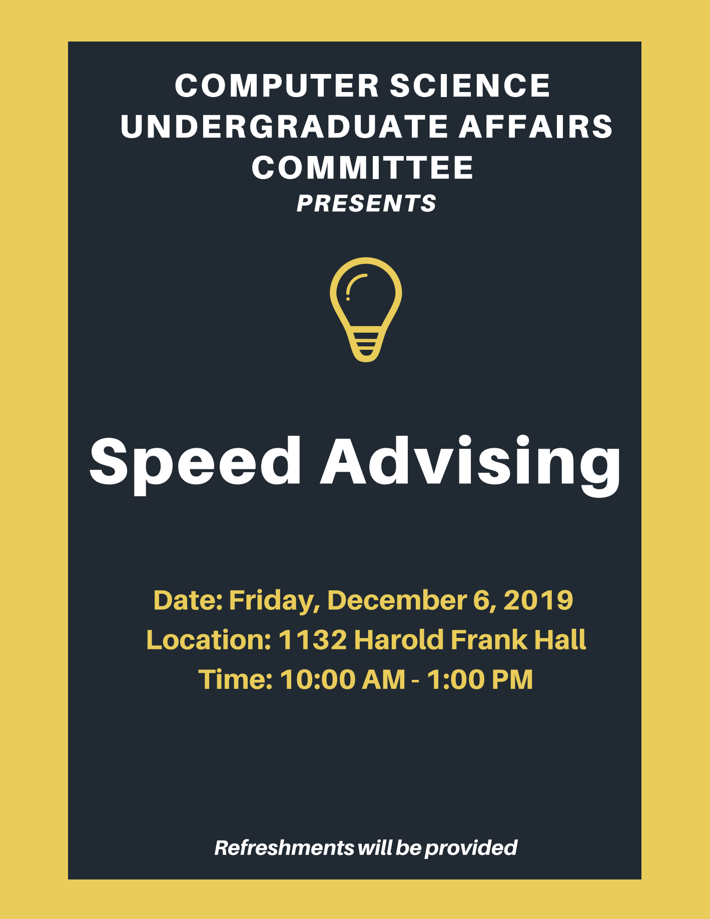


Problem Solving with Computers-I





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

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

}
```



COMPUTER SCIENCE UNDERGRADUATE AFFAIRS COMMITTEE

PRESENTS

Speed Advising

Date: Friday, December 6, 2019
Location: 1132 Harold Frank Hall
Time: 10:00 AM - 1:00 PM

Refreshments will be provided

Final Exam: Monday 12/09, noon-3:00p, Embarcadero Hall

Final Exam Review Session:

Day: Friday (12/06)
Time: 5:00p - 7:00p
Location: Phelps 35~~8~~26

Diba's OH: Wed. 10-11a. Fri- 2p-4pm

Thinking recursively !

return type

parameter

```
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 !

1 2

```
int fac(int N){

          1
    if (N <= 1)
        return 1;

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

fac(2-1) * 2

1 * 2

N	Return
fac(1)	1
fac(2)	2
fac(3)	6

Base case

Recursive case

(shorter version)

Human: Base case and 1 step

Computer: Everything else

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)

5 * fac(4)

Behind the curtain…

```
int fac(int N){

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

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

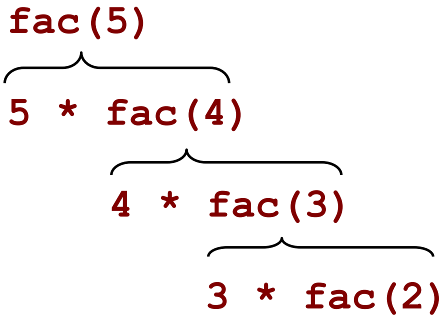
Behind the curtain…

```
int fac(int N){

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

```
fac(5)

5 * fac(4)

    4 * fac(3)
```

Behind the curtain…

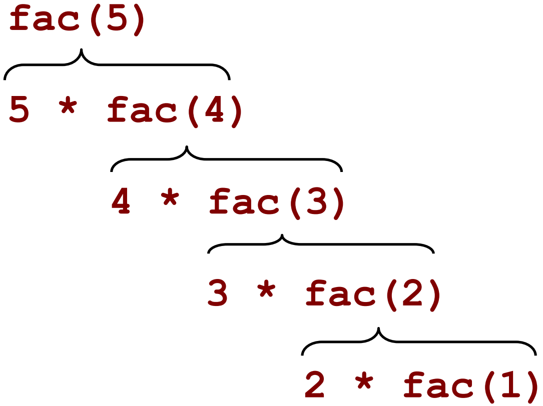
```
int fac(int N){

    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)

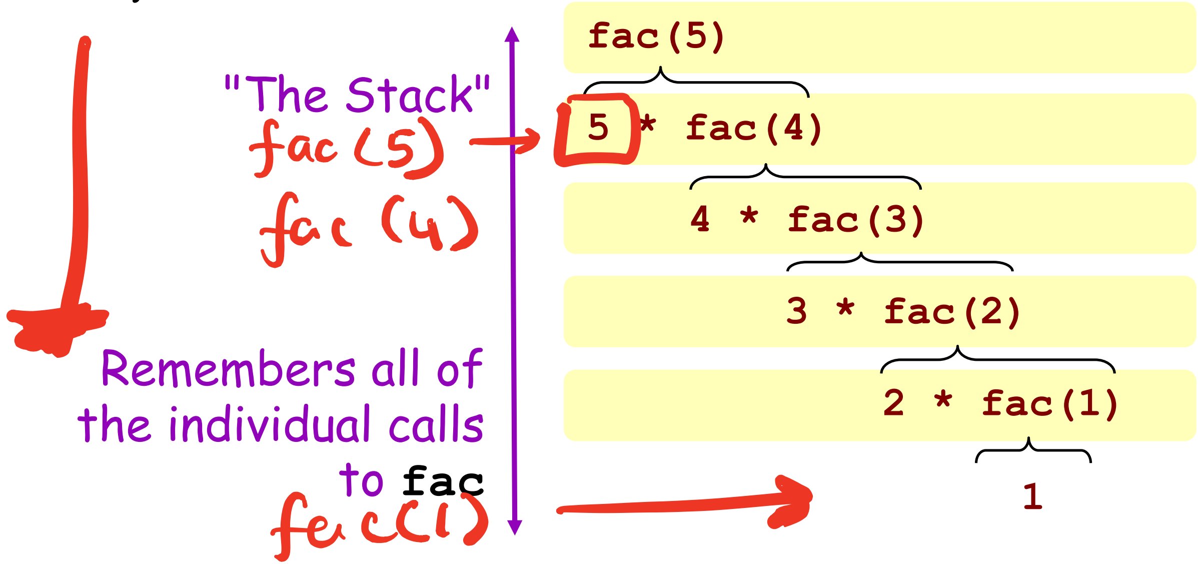
4 * fac(3)

3 * fac(2)

2 * fac(1)

Behind the curtain…

```
int fac(int N){

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



"The Stack"

fac(5)

5 * fac(4)

4 * fac(3)

3 * fac(2)

2 * fac(1)

1

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);
}
```

$3 * \underset{\uparrow 2}{\boxed{\text{fac(2)}}}$

$$\overbrace{\texttt{5 * }\overbrace{\texttt{4 * }\overbrace{\texttt{3 * }\overbrace{\boxed{\texttt{2 * 1}}}^{\texttt{fac(2)}}}^{\texttt{fac(3)}}}^{\texttt{fac(4)}}}^{\texttt{fac(5)}}$$

Behind the curtain…

```
int fac(int N){

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

fac(5)

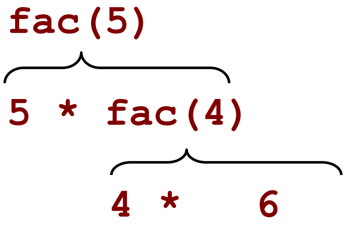
$$\overbrace{\texttt{5 * fac(4)}}$$

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

$$\overbrace{\texttt{3 *   2}}$$

Behind the curtain…

```
int fac(int N){

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



fac(5)

5 * fac(4)

4 * 6

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);
}
```

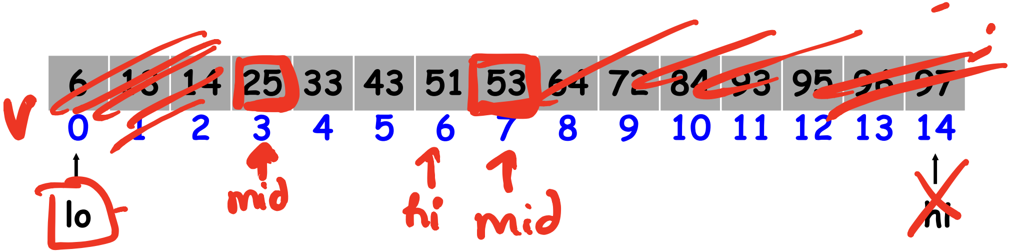
`fac(5)`

Result: `120`

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]`.

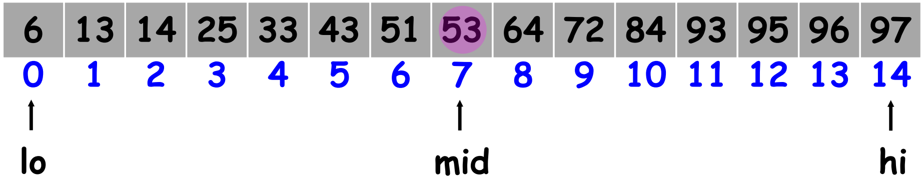
- Ex. Binary search for 33.



Search through the sub-array starting at index(lo) ending at index(hi)

Binary Search

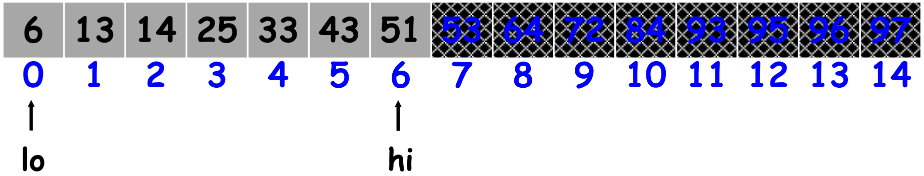
- Ex. Binary search for 33.



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

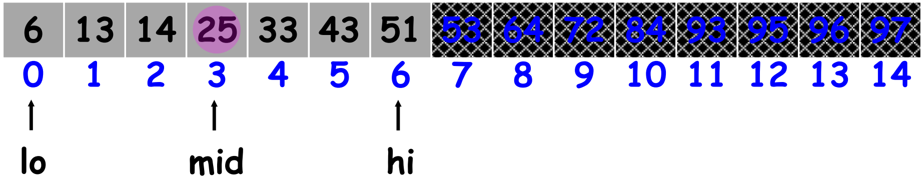
- Ex. Binary search for 33.



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
lo						hi								

Binary Search

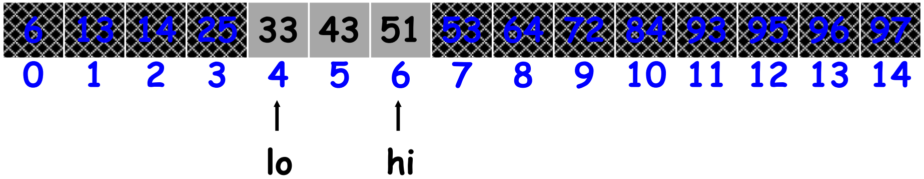
- Ex. Binary search for 33.



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

Binary Search

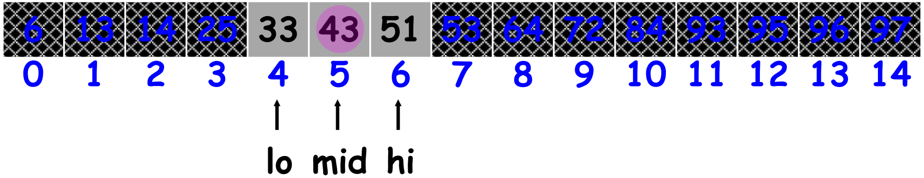
- Ex. Binary search for 33.



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

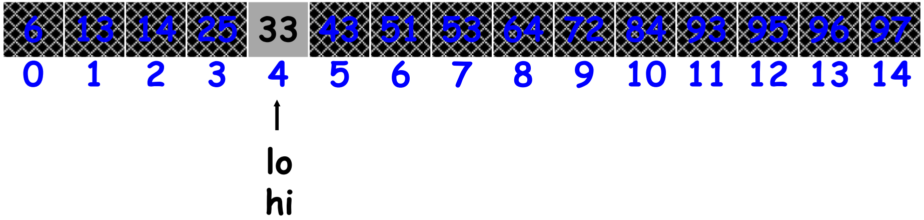
- Ex. Binary search for 33.



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

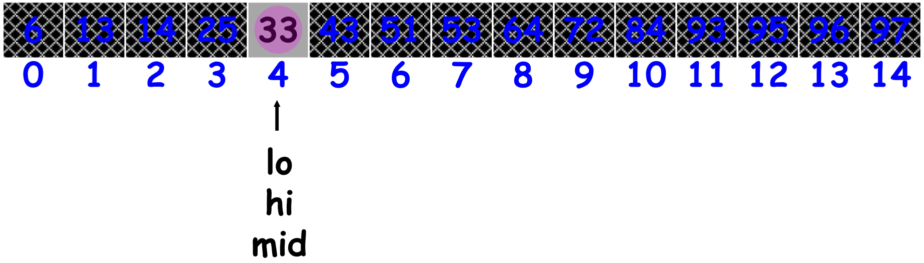
- Ex. Binary search for 33.



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

Binary Search

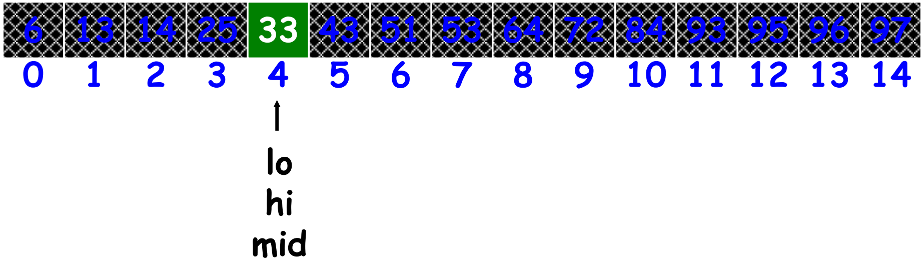
- Ex. Binary search for 33.



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

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){
```

Incorrect return value for the case hi == lo

A:

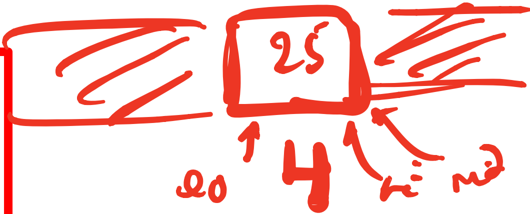
```
if(hi<=lo){
    return -1;
}
```

D: Neither

33

B:

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



lo = 4
hi = 4
mid = 4

lo = 5
hi = 4

hi < low
(Base case A)

C: Both A and B

Fill in the blanks

⑤

2	10	30
0	1	2

l

```
int binarySearch(int v[], int value, int lo, int hi){
      if(hi < lo)
            return -1;                  ] Base case 1
      int mid = (lo + hi)/2;
      if(v[mid] == value)
            return mid;                 ] Base case 2
      if(v[mid] < value){ // search right half
return binarySearch(v, value, mid+1 , hi);
      } else // search left half.
return binarySearch(v, value, lo, mid-1).
}
```

return statement is Important

lo	hi	mid
0	2	1
0	0	0
1	0	

A: lo

B: mid - 1

C: mid

[✓ D: mid + 1]

E: hi

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

See lecture code

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

See code written in next lecture

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);
    }
}
```

Missing return