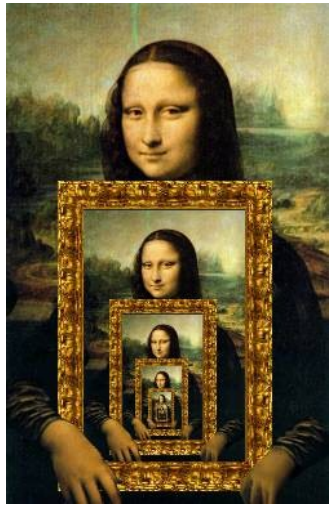
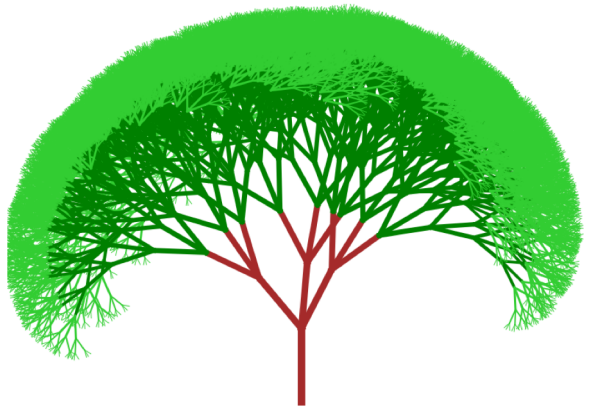
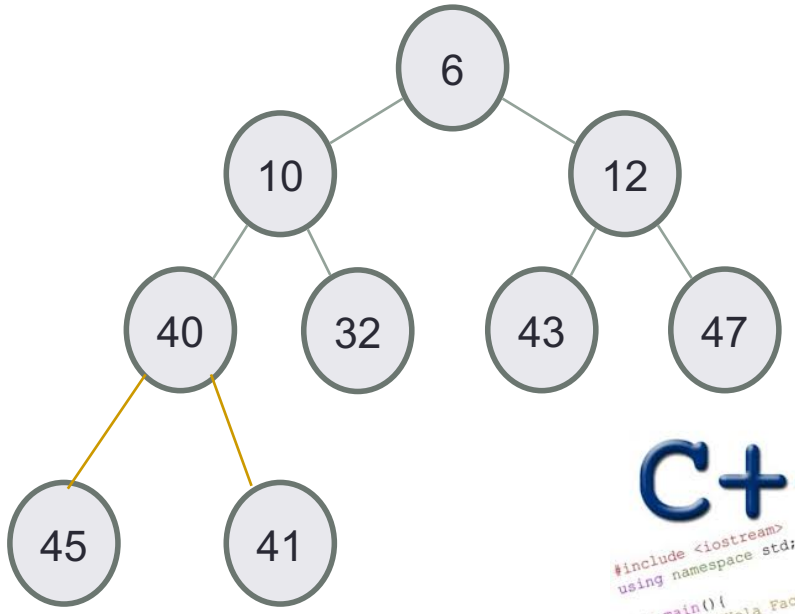
MORE ON RECURSION



Problem Solving with Computers-I





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

this is legal!

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

legal **!=** *recommended*

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

No ***base case*** -- the calls to **`fac`** will never stop (nicely)!

Make sure you have a **base case**, *then* worry about the recursion...


```
int fac(int N){
      if(N<=1)
         return 1;
      return fac(N);
}
```

Roadsigns and recursion

examples of self-fulfilling danger

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…

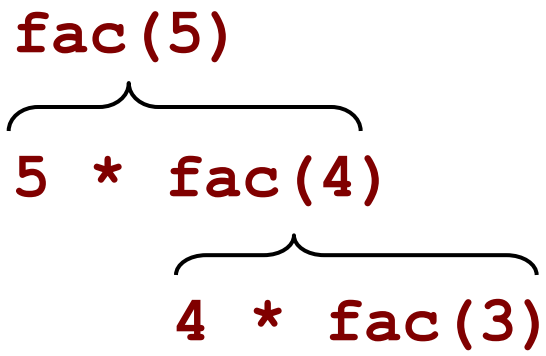
```
int fac(int N){

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

fac(5)

5 * fac(4)

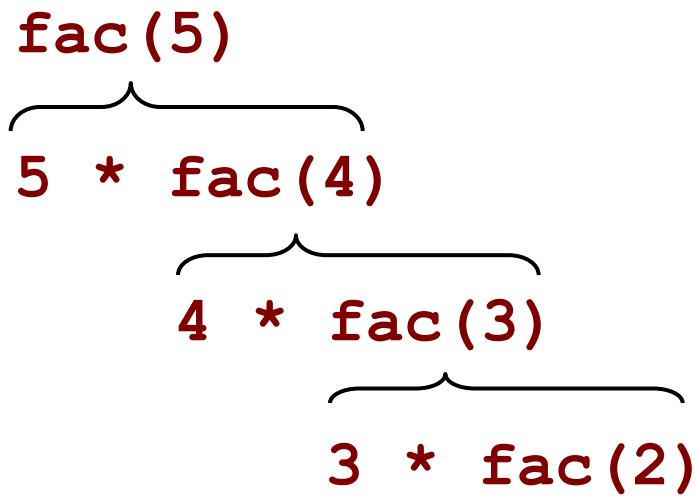
Behind the curtain…

```
int fac(int N){

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



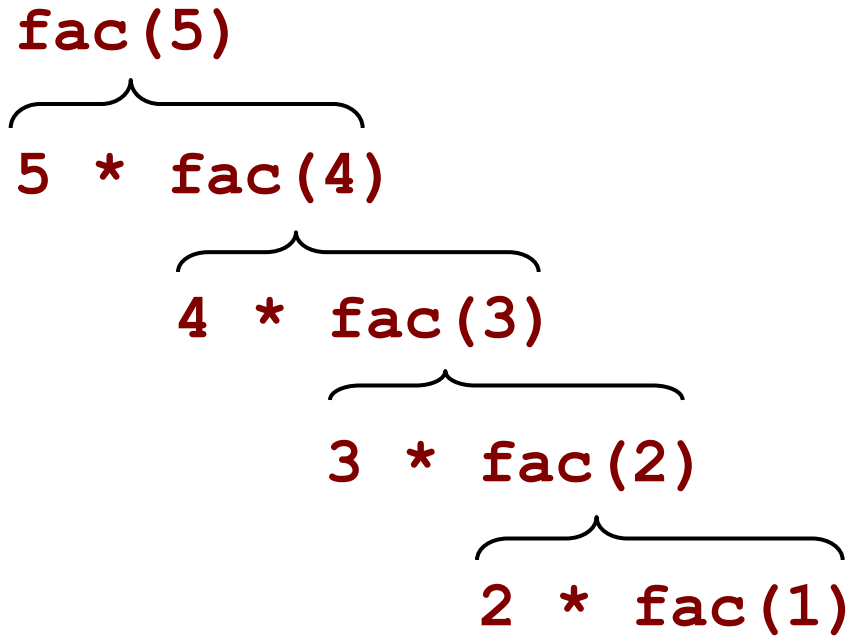
Behind the curtain…

```
int fac(int N){

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



Behind the curtain…

```
int fac(int N){

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

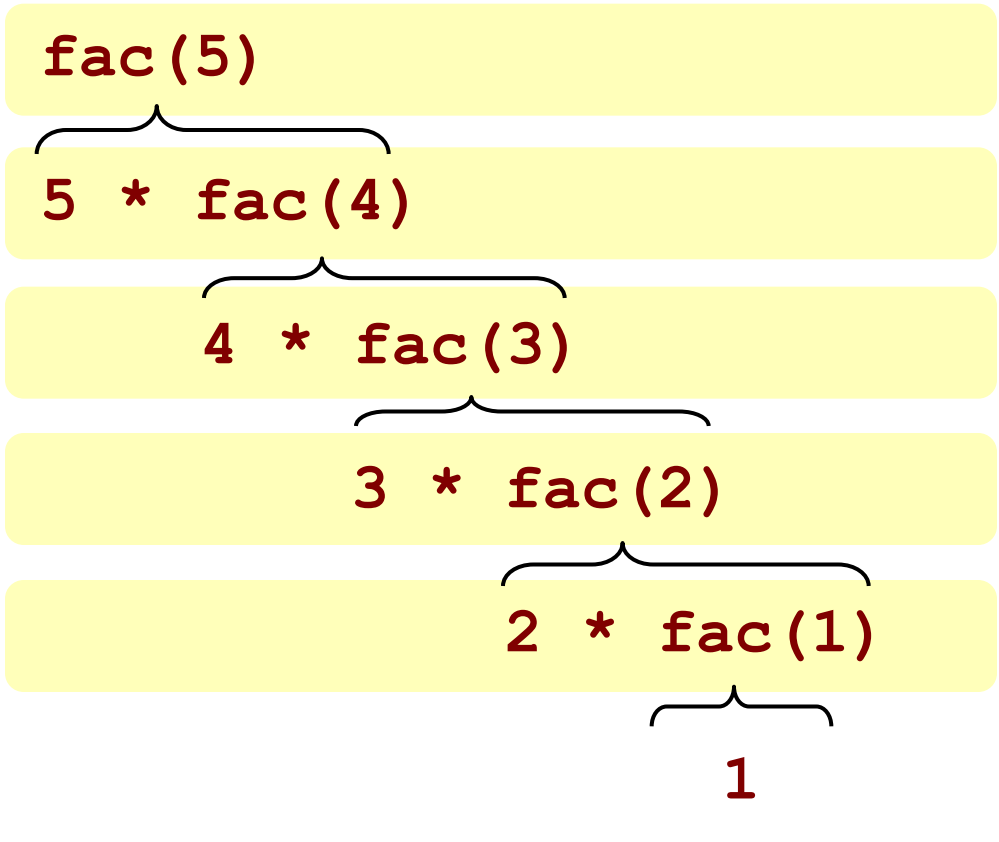


Behind the curtain…

```
int fac(int N){

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

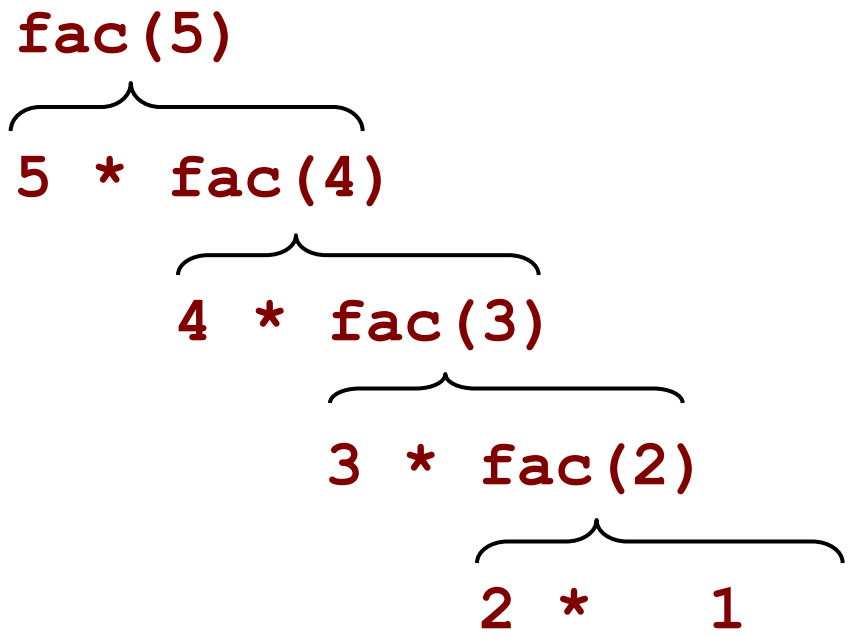
"The Stack"

Remembers all of the individual calls to **fac**



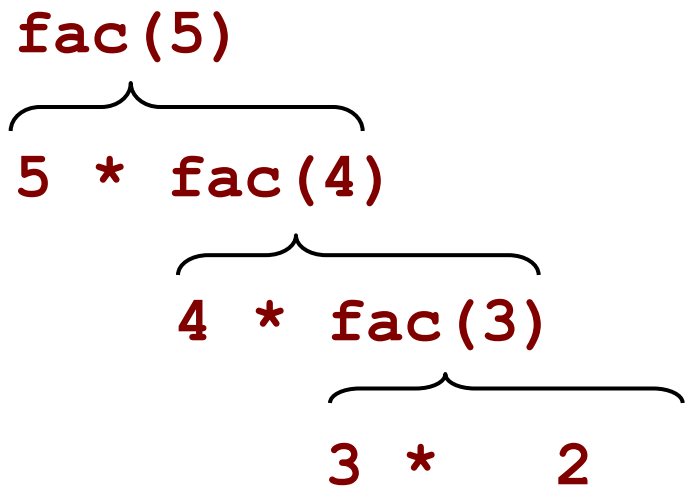
Behind the curtain…

```
int fac(int N){

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



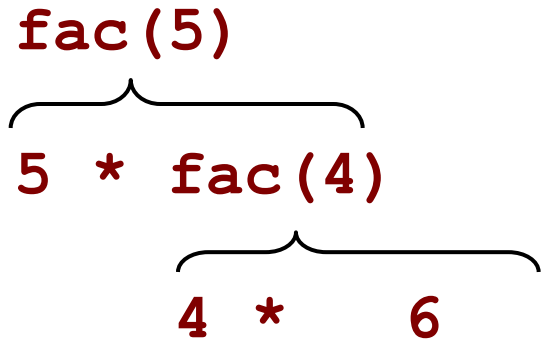
Behind the curtain…

```
int fac(int N){

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



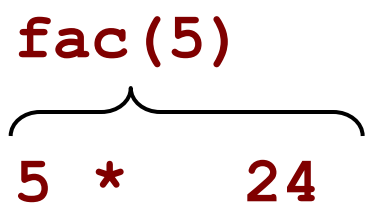
Behind the curtain…

```
int fac(int N){

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



Behind the curtain…

```
int fac(int N){

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



Behind the curtain…

```
int fac(int N){

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

fac(5)

Result: 120

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

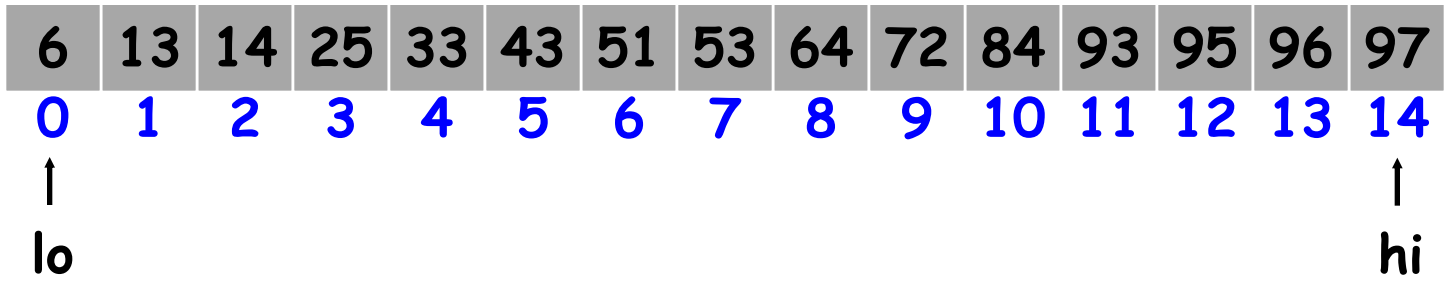
Delete all nodes with a given value

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

Binary Search: Efficient search in a sorted array

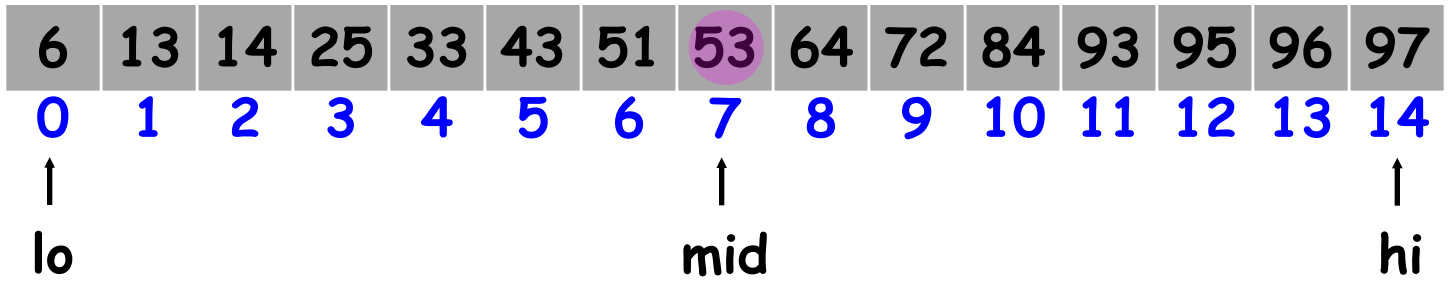
- Binary search. Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Invariant. Algorithm maintains `a[lo]` ≤ `value` ≤ `a[hi]`.

- Ex. Binary search for 33.



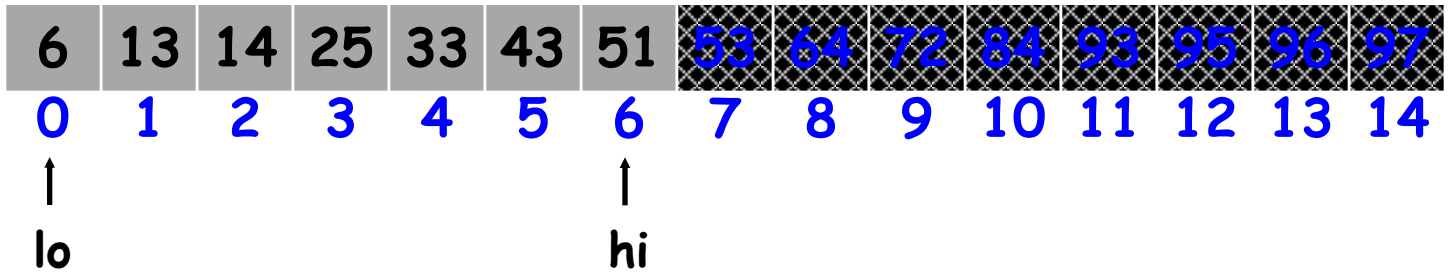
Binary Search

- Ex. Binary search for 33.



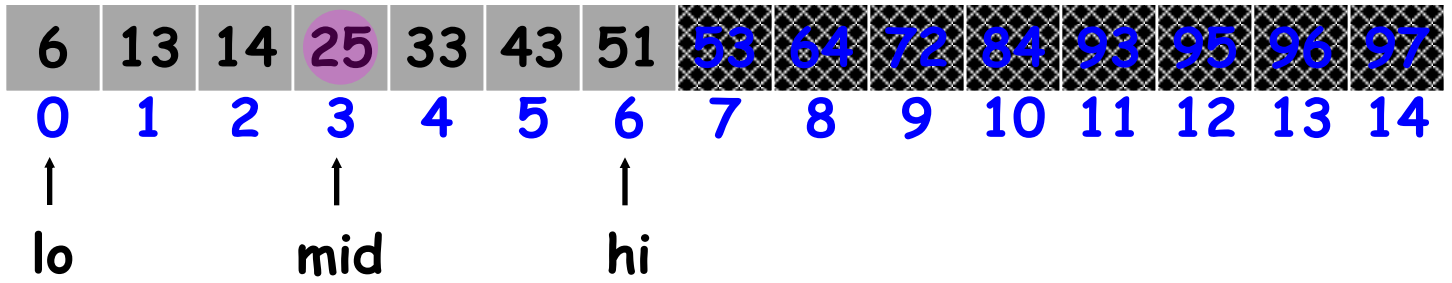
Binary Search

- Ex. Binary search for 33.



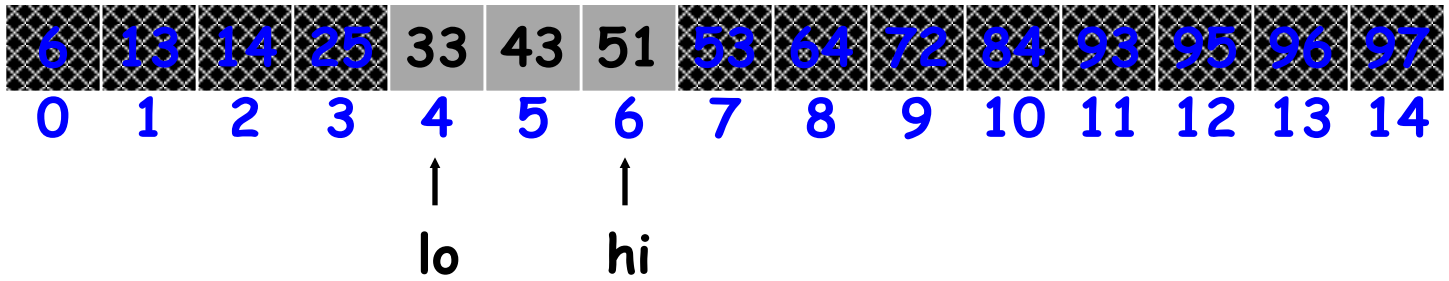
Binary Search

- Ex. Binary search for 33.



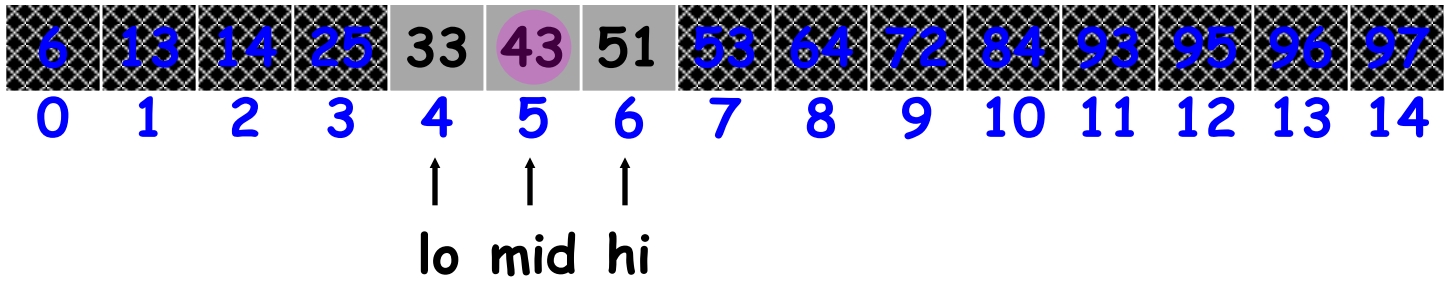
Binary Search

- Ex. Binary search for 33.



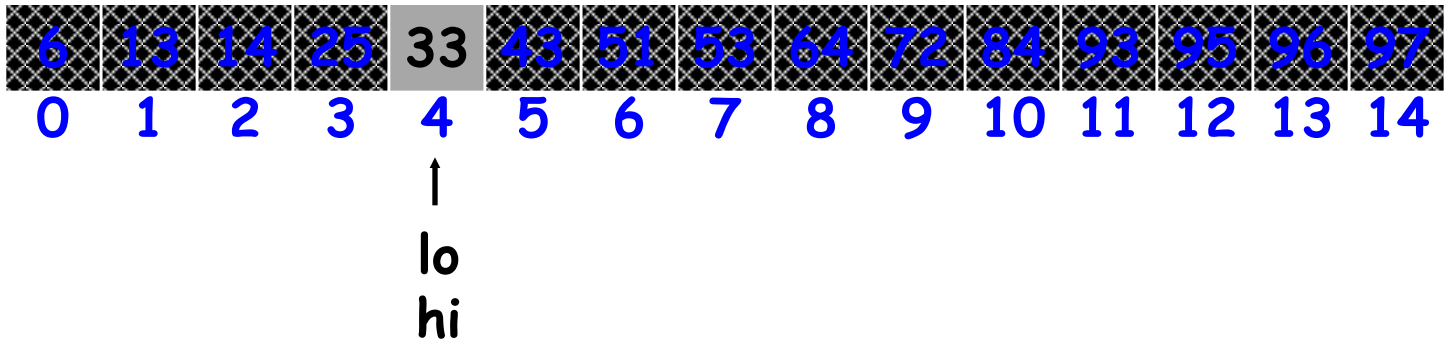
Binary Search

- Ex. Binary search for 33.



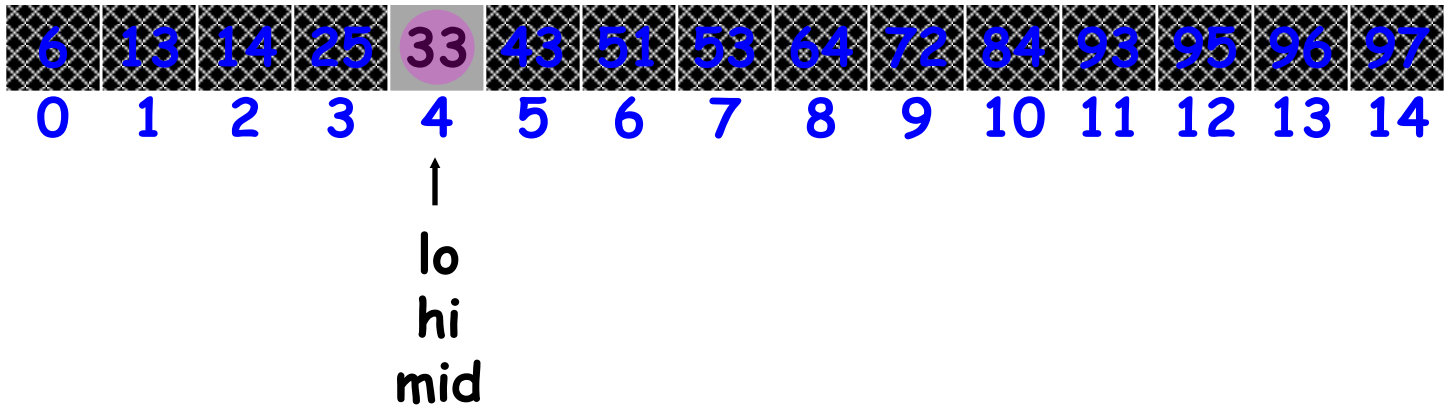
Binary Search

- Ex. Binary search for 33.



Binary Search

- Ex. Binary search for 33.



Binary Search

- Ex. Binary search for 33.