

# COMP 250

## INTRODUCTION TO COMPUTER SCIENCE

Lecture 20 – Recursion 2 (Binary Search)

Giulia Alberini, Fall 2018

## FROM LAST CLASS

---

- Some examples of recursive algorithms

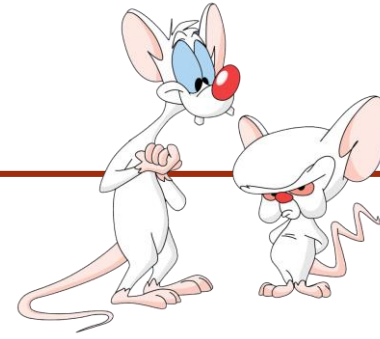
## REVERSING A LIST (RECURSIVE)

```
public static void reverse(List list) {  
    if(list.size()==1) {  
        return;  
    }  
    firstElement = list.remove(0); // remove first element  
    reverse(list); // now the list has n-1 elements  
    list.add(firstElement); // appends at the end of the list  
}
```

## EXAMPLE 5 – SORTING A LIST (RECURSIVE)

```
public static void sort(List list) {  
    if(list.size()==1) {  
        return;  
    }  
    minElement = removeMinElement(list);  
    sort(list); // now the list has n-1 elements  
    list.add(0, minElement); // insert at the beginning of list  
}
```

# WHAT ARE WE GOING TO DO TODAY?



- More recursive algorithms
- Binary Search

## RECALL: DECIMAL TO BINARY (ITERATIVE)

### ALGORITHM

#### Constructing Base 2 Expansions

**procedure** *BinaryExpansion*( $n$ )

$k := 0$

**While**  $m > 0$

$a_k := n \% 2$

$n := n / 2$

$k := k + 1$

**return**  $(a_{k-1}, \dots, a_1, a_0)$

Recall that a decimal number  $n$  requires approximately  $\log_2 n$  bits for its binary representation.

## DECIMAL TO BINARY (RECURSIVE)

### ALGORITHM

#### Constructing Base 2 Expansions

**procedure** *BinaryExpansion*( $n$ )

**If**  $n > 0$ :

*BinaryExpansion*( $n/2$ )

    print( $n\%2$ )

Also in this case, there are  $\log_2 n$  recursive calls

## POWER ( $x^n$ ) – ITERATIVE

**Let  $x$  a positive integer and let  $n$  be a positive number.**  
 $x$  has some number of bits e.g. 32.

```
power(x, n) {  
    result =1;  
    for(int i=1; i<=n; i++) {  
        result = result *x;  
    }  
    return result;  
}
```



## POWER ( $x^n$ ) – RECURSIVE

```
power(x, n) {  
    if(n==0) {  
        return 1;  
    } else {  
        return x*power(x,n-1);  
    }  
}
```

## POWER() – CAN WE DO BETTER?

More interesting approach using recursion:

$$x^{18} = x^9 * x^9$$

$$x^9 = x^4 * x^4 * x$$

$$x^4 = x^2 * x^2$$

## POWER() – CAN WE DO BETTER?

```
power( x, n) {  
    if (n == 0)  
        return 1;  
    else if (n == 1)  
        return x;  
    else{  
        tmp = power(x, n/2);  
        if (n%2==0)  
            return tmp*tmp;        // one multiplication  
        else  
            return tmp*tmp*x      // two multiplications  
    }  
}
```

## A SIMILAR IDEA CAN BE IMPLEMENTED ITERATIVELY

IDEA: Let's use the binary expansion of  $n$ , say  $n = (a_{k-1}, \dots, a_1, a_0)_2$ .

Note that:

$$x^n = x^{a_{k-1}2^{k-1} + \dots + a_12 + a_0} = x^{a_{k-1}2^{k-1}} \dots x^{a_12} \cdot x^{a_0}$$

This shows how to compute  $x^n$ : we only need to compute the values of  $x, x^2, (x^2)^2 = x^4, \dots, x^{2^k}$ . Once we have these terms we multiply the terms  $x^{2^j}$ , where  $a_j = 1$ .

EXAMPLE:  $x^{243}$

---

$$n = (243)_{10} = (11110011)_2$$

Q: How many multiplications do we need?

EXAMPLE:  $x^{243}$

$$n = (243)_{10} = (11110011)_2$$

Q: How many multiplications do we need?

A: Recursive method:  $5*2 + 2*1 = 12$ .

Iterative method:  $7 + 7 = 14$

The highest order bit in the recursive method is the base case, and doesn't require a multiplication.

The lowest order bit in the iterative method does not require multiplication.

EXAMPLE:  $x^{243}$

---

$$n = (243)_{10} = (11110011)_2$$

**Q:** How many multiplications do we need?

**A:**  $O(\log_2 n)$

## OBSERVATIONS

The second approach we looked at uses fewer multiplications than the first one, and thus the second approach *seems faster*.

Q: Is this indeed the case?

A: No. Why not ?



## OBSERVATIONS

Hint: Let  $x$  be a positive integer with  $M$  digits.

- $x^2$  has about ? digits.
- $x^3$  has about ? digits.
- :
- $x^n$  has about ? digits.

## OBSERVATIONS

Hint: Let  $x$  be a positive integer with  $M$  digits.

- $x^2$  has about  $2M$  digits.
- $x^3$  has about  $3M$  digits.
- $\vdots$
- $x^n$  has about  $n * M$  digits.

*We cannot assume that multiplication takes 'constant' time.*

Taking large powers gives very large numbers and multiplications becomes more expensive.

The background features a series of concentric circles in a light gray color, some of which are dashed. A solid dark red rectangle is positioned in the center of the image. The text "BINARY SEARCH" is written in white, uppercase letters within this red rectangle.

# BINARY SEARCH

## SEARCHING A LIST

- Goal: find a given element in a list.
- Solution: go through all the elements in the list and check whether the element is there (*linear search*).
- Could we do this any faster if the list was sorted to begin with?

Think of how you search for a term in an index. Do you start at the beginning and then scan through to the end? (No.)

Exponential inequality 185  
Exponential running time 186,  
661, 911  
Extended Church-Turing the-  
sis 910  
Extensible library 101  
External path length 418, 832

## F

Factor an integer 919  
Factorial function 185  
Fail-fast design 107  
Fail-fast iterator 160, 171  
Farthest pair 210  
Fibonacci heap 628, 682  
Fibonacci numbers 57  
FIFO. *See* First-in first-out policy  
FIFO queue.  
    *See* Queue data type  
File system 493  
Filter 60  
    blacklist 491  
    dedup 490  
    whitelist 8, 491  
Final access modifier 105–106  
Fingerprint search 774–778  
Finite state automaton.

*See also* Maxflow problem  
flow network 888  
inflow and outflow 888  
residual network 895  
st-flow 888  
st-flow network 888  
value 888  
Floyd, R. W. 326  
Floyd's method 327  
for loop 16  
Ford-Fulkerson 891–893  
    analysis of 900  
    maximum-capacity path 901  
    shortest augmenting path 897  
Ford, L. 683  
Foreach loop 138  
    arrays 160  
    strings 160  
Forest  
    graph 520  
    spanning 520  
Forest-of-trees 225  
Formatted output 37  
Fortran language 217  
Fragile base class problem 112  
Frazer, W. 306  
Fredman, M. L. 628  
Function call stack 346, 415

symbol tables 363  
type parameter 122, 134  
Genomics 492, 498  
Geometric data types 76–77  
Geometric sum 185  
getClass() method 101, 103  
Girth of a graph 559  
Global variable 113  
Gosper, R. W. 759  
Graph data type 522–527  
Graph isomorphism 561, 919  
Graph processing 514–693.  
    *See also* Directed graph;  
    *See also* Edge-weighted  
    digraph; *See also* Edge-  
    weighted graph; *See*  
    *also* Undirected graph;  
    *See also* Directed acyclic  
    graph  
    Bellman-Ford 668–681  
    breadth-first search 538–541  
    components 543–546  
    critical-path method 664–666  
    depth-first search 530–537  
    Dijkstra's algorithm 652  
    Kosaraju's algorithm 586–590  
    Kruskal's algorithm 624–627  
    longest paths 611, 612

# BINARY SEARCH

- Inputs:
  - A sorted list.
  - The element we are looking for (the *key*)
- IDEA: First compare the key with the element in the middle of the list
  - If the key is less than the middle element, we only need to search the first half of the list, so we continue searching on this smaller list.
  - If the key is greater than the middle element, we only need to search the second half of the list, so we continue searching on this smaller list.
  - If the key equals the middle element, we have a match – return its index.

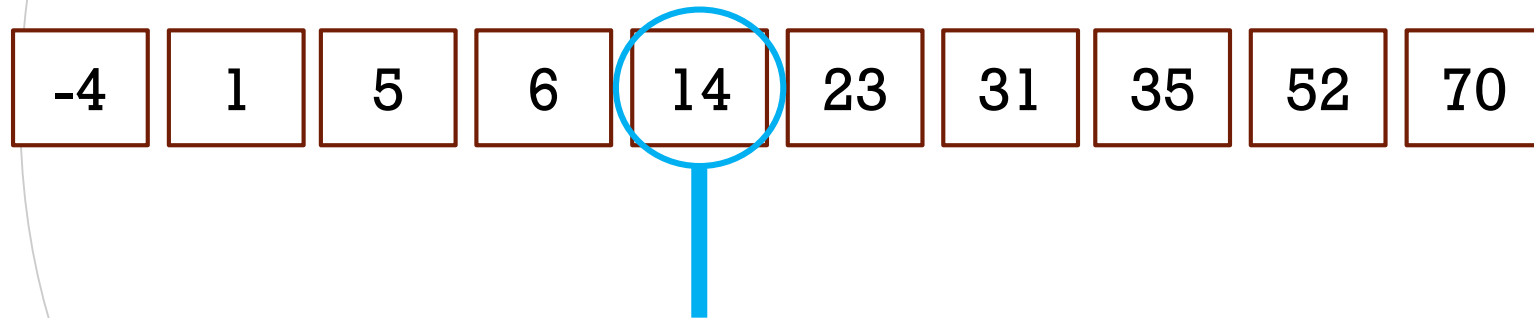
## EXAMPLE

- Search for 25

|    |   |   |   |    |    |    |    |    |    |
|----|---|---|---|----|----|----|----|----|----|
| -4 | 1 | 5 | 6 | 14 | 23 | 31 | 35 | 52 | 70 |
|----|---|---|---|----|----|----|----|----|----|

## EXAMPLE

- Search for 25
- Look at the middle element and compare





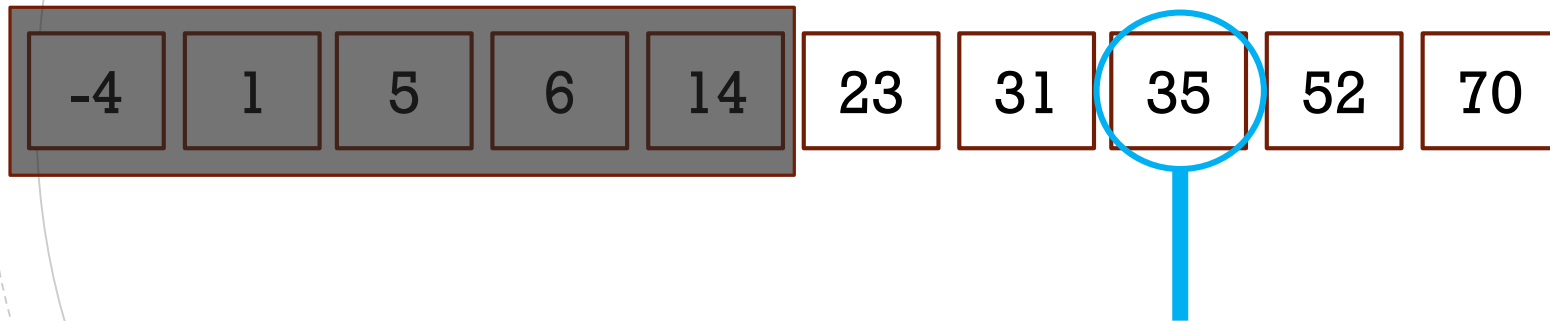
## EXAMPLE

- Search for 25
- Look at the middle element and compare
- If not equal: discard half of the list and keep searching on the other half



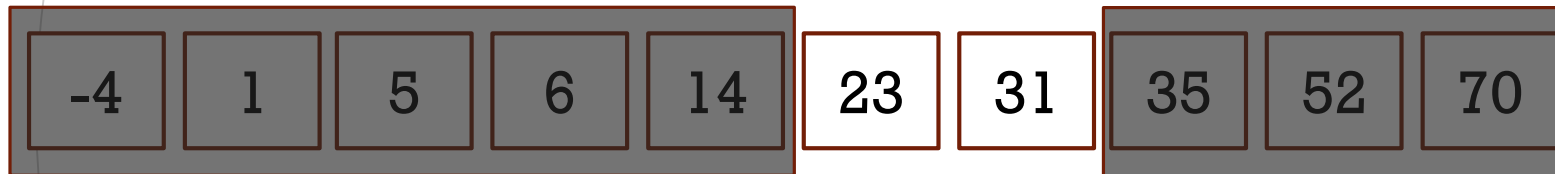
## EXAMPLE

- Search for 25
- Look at the middle element and compare



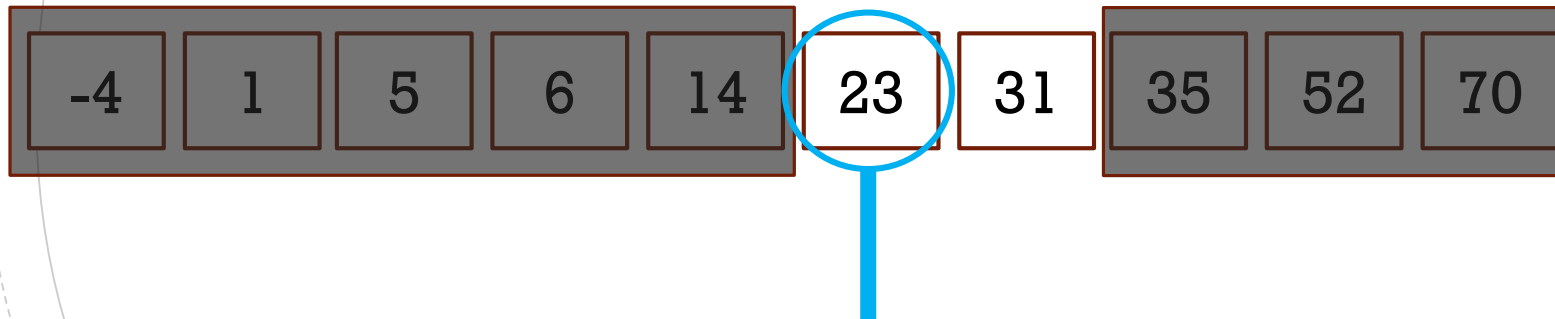
## EXAMPLE

- Search for 25
- Look at the middle element and compare
- If not equal: discard half of the list and keep searching on the other half



## EXAMPLE

- Search for 25
- Look at the middle element and compare



## EXAMPLE

- Search for 25
- Look at the middle element and compare
- If not equal: discard half of the list and keep searching on the other half



## EXAMPLE

- Search for 25
- Look at the middle element and compare



## EXAMPLE

- Search for 25
- Look at the middle element and compare
- If not equal: discard half of the list and keep searching on the other half

|    |   |   |   |    |    |    |    |    |    |
|----|---|---|---|----|----|----|----|----|----|
| -4 | 1 | 5 | 6 | 14 | 23 | 31 | 35 | 52 | 70 |
|----|---|---|---|----|----|----|----|----|----|

## EXAMPLE

- Search for 25
- There are no more elements in the list → the element is not there! Return -1.

|    |   |   |   |    |    |    |    |    |    |
|----|---|---|---|----|----|----|----|----|----|
| -4 | 1 | 5 | 6 | 14 | 23 | 31 | 35 | 52 | 70 |
|----|---|---|---|----|----|----|----|----|----|



## IMPLEMENT BINARY SEARCH

- Idea: keep track of the left and right indices denoting the section of the list that needs to be searched.
- What is the index of the element that we compare to the key as a function of the left and right indices?

## BACK TO EXAMPLE

- Search for 25 (initialize left and right)

|    |   |   |   |    |    |    |    |    |    |
|----|---|---|---|----|----|----|----|----|----|
| -4 | 1 | 5 | 6 | 14 | 23 | 31 | 35 | 52 | 70 |
|----|---|---|---|----|----|----|----|----|----|

left = 0

right = 9

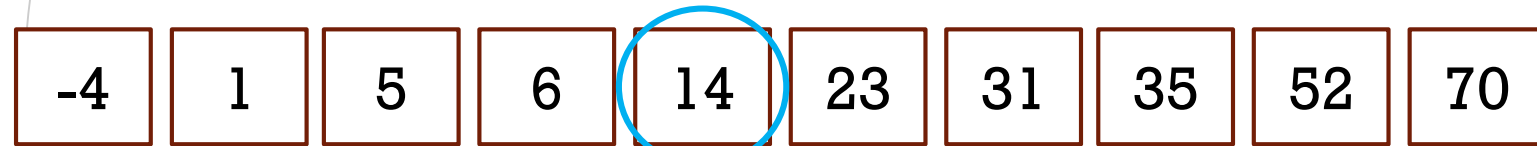
right = size - 1

## BACK TO EXAMPLE

- Search for 25
- Look at the middle element and compare (compute mid)

$$\text{mid} = (\text{left} + \text{right}) / 2$$

$$\text{mid} = 4$$



left = 0

right = 9

## BACK TO EXAMPLE

- Search for 25
- Look at the middle element and compare
- If not equal: discard half of the list and keep searching on the other half (update left)

mid = 4



left = 5

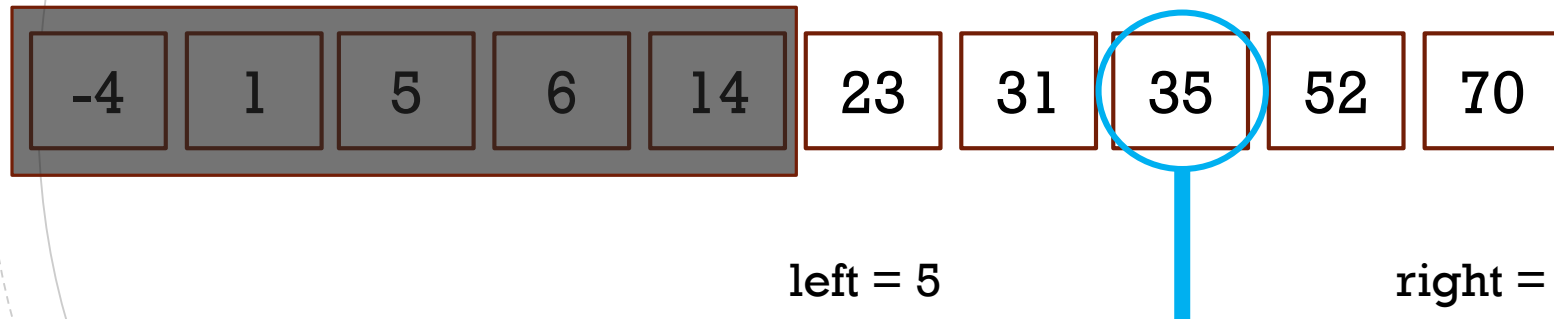
right = 9

left = mid + 1

## EXAMPLE

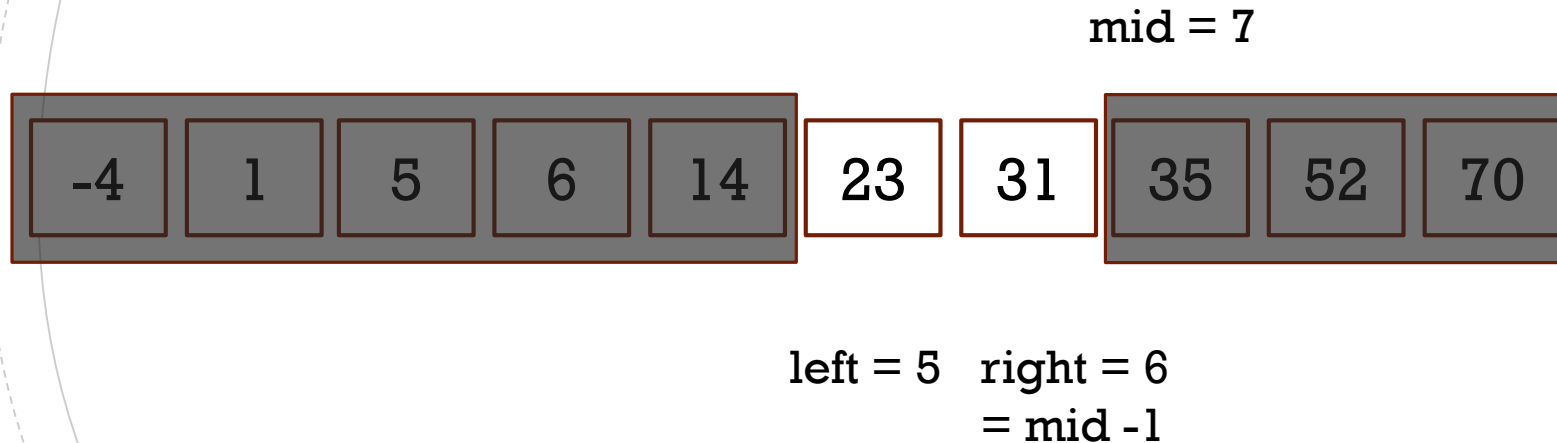
- Search for 25
- Look at the middle element and compare (compute mid)

$$\begin{aligned}\text{mid} &= (\text{left} + \text{right}) / 2 \\ &= 7\end{aligned}$$



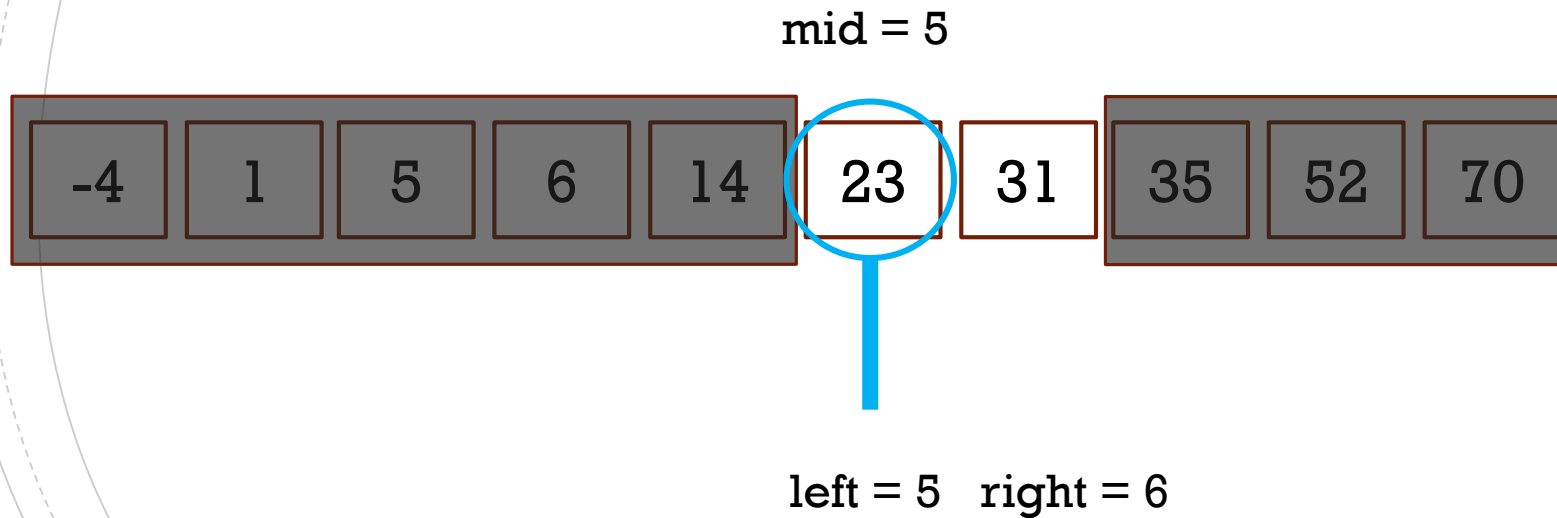
## EXAMPLE

- Search for 25
- Look at the middle element and compare
- If not equal: discard half of the list and keep searching on the other half (update right)



## EXAMPLE

- Search for 25
- Look at the middle element and compare (compute mid)



## EXAMPLE

- Search for 25
- Look at the middle element and compare
- If not equal: discard half of the list and keep searching on the other half (update left)

mid = 5

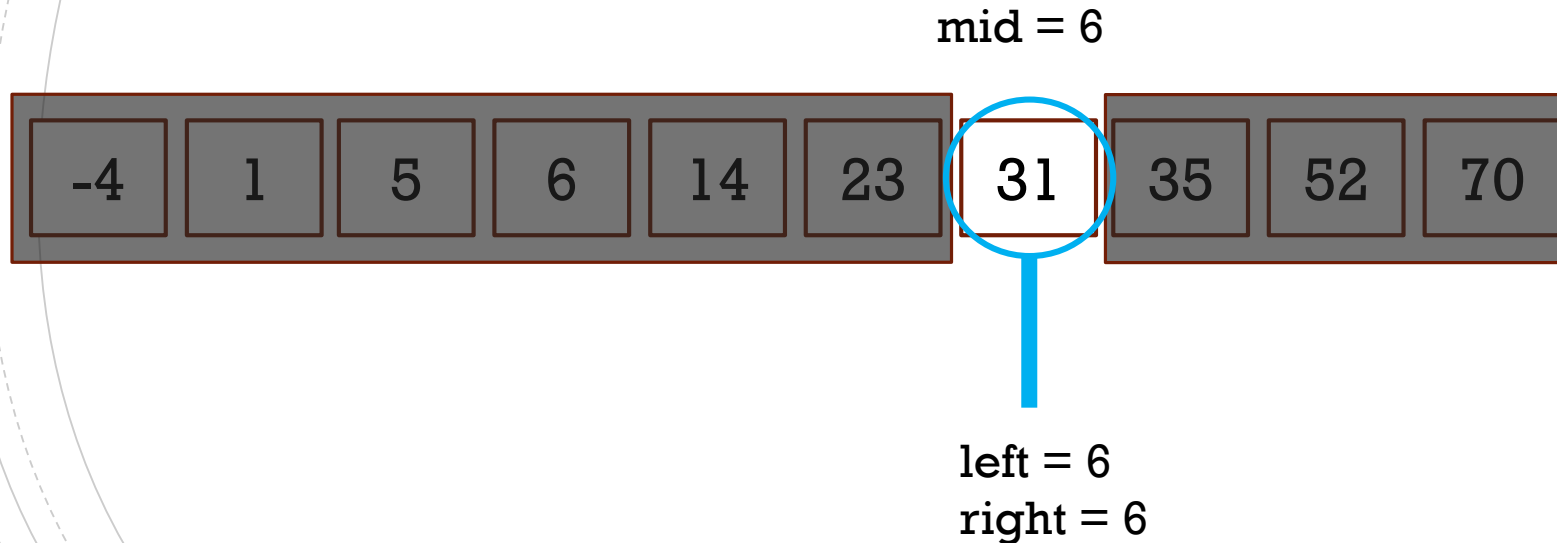


left = 6  
right = 6



## EXAMPLE

- Search for 25
- Look at the middle element and compare (compute mid)



## EXAMPLE

- Search for 25
- Look at the middle element and compare
- If not equal: discard half of the list and keep searching on the other half (update right)

mid = 6

|    |   |   |   |    |    |    |    |    |    |
|----|---|---|---|----|----|----|----|----|----|
| -4 | 1 | 5 | 6 | 14 | 23 | 31 | 35 | 52 | 70 |
|----|---|---|---|----|----|----|----|----|----|

right= 5   left = 6

## EXAMPLE

- Search for 25
- There are no more elements in the list (**right < left**)  
→ the element is not there! Return -1.

|    |   |   |   |    |    |    |    |    |    |
|----|---|---|---|----|----|----|----|----|----|
| -4 | 1 | 5 | 6 | 14 | 23 | 31 | 35 | 52 | 70 |
|----|---|---|---|----|----|----|----|----|----|

right= 5   left = 6

## BINARY SEARCH (ITERATIVE)

```
binarySearch(list, key){  
    left = 0  
    right = list.size() - 1    } initialize left and right  
    while(low <= high){ // until there are elements to search  
  
        ...  
  
    }  
    return -1                // key not in list  
}
```

## BINARY SEARCH (ITERATIVE)

```
binarySearch(list, key){
    left = 0
    right = list.size() - 1 } initialize left and right
    while(low <= high){ // until there are elements to search
        mid = (left + right)/2 // compute mid
        if(list[mid]==key) // compare element with key
            return mid
        else {
            // update either left or right

        }
    }
    return -1 // key not in list
}
```

## BINARY SEARCH (ITERATIVE)

```
binarySearch(list, key){  
    left = 0  
    right = list.size() - 1  
    while(low <= high){ // until there are elements to search  
        mid = (left + right)/2 // compute mid  
        if(list[mid]==key) // compare element with key  
            return mid  
        else {  
            if(key<list[mid])  
                right = mid -1 // update right  
            else  
                left = mid + 1 // update left  
        }  
    }  
    return -1 // key not in list  
}
```

## BINARY SEARCH (RECURSIVE)

```
binarySearch(list, key){  
    left = 0  
    right = list.size() - 1  
    while(low <= high){  
        mid = (left + right)/2  
        if(list[mid]==key)  
            return mid  
        else {  
            if(key<list[mid])  
                right = mid -1  
            else  
                left = mid + 1  
        }  
    }  
    return -1  
}
```

What should change?

## BINARY SEARCH (RECURSIVE)

```
binarySearch(list, key, left, right) {  
    while(low <= high) {  
        mid = (left + right)/2  
        if(list[mid]==key)  
            return mid  
        else {  
            if(key<list[mid])  
                right = mid -1  
            else  
                left = mid + 1  
        }  
    }  
    return -1  
}
```

Pass left and right as  
parameters to the method



## BINARY SEARCH (RECURSIVE)

```
binarySearch(list, key, left, right) {  
    if(low <= high) {  
        mid = (left + right)/2  
        if(list[mid]==key)  
            return mid  
        else {  
            if(key<list[mid])  
                right = mid -1  
            else  
                left = mid + 1  
        }  
    }  
    return -1  
}
```

Replace the while with an if

## BINARY SEARCH (RECURSIVE)

```
binarySearch(list, key, left, right) {  
    if(low <= high) {  
        mid = (left + right)/2  
        if(list[mid]==key)  
            return mid  
        else {  
            if(key<list[mid])  
                binarySearch(list, key, left, mid-1) Add recursive calls  
            else  
                binarySearch(list, key, mid+1, right)  
        }  
    }  
    return -1  
}
```

## OBSERVATIONS

---

**Q:**    How many times through the while loop ?    (iterative)  
          How many recursive calls?    (recursive)

**A:**

## OBSERVATIONS

Q: How many times through the while loop ? (iterative)  
How many recursive calls? (recursive)

A: Worst case: the element cannot be found. Then, worst time is  $O(\log_2 n)$  where  $n$  is size of the list. Why? Because each time we are approximately halving the size of the list.

An orange rectangular banner with a rough, paint-splattered edge. To its right is a paint roller with a red handle and a silver frame, positioned as if it has just finished painting the banner. The background features faint, concentric circular lines on the left and diagonal lines on the right.

# Coming Soon

- Mergesort
- Quicksort