Algorithm analysis & design

Analysis Of searching Algorithms

Presented By:

T.A. Asmaa Hamad El-saied

E-mail: eng.asmaa134@gmail.com

Agenda

- Linear Search
- Binary Search

Searching

- Given a collection and an element (key) to find…
- **Output**
  - Return a value (position of key or -1)

Linear Search

Linear Search: A SimpleSearch

- Asearch traverses the collection until
  - The desired element is found
  - Or the collection is exhausted
- **There are two solutions**
  - Iterative solution
  - Recursive solution

6

Iterative solution

Linear Search pseudo-code: Iterative solution

Algorithm Linear_search (A, n, target)

    For i=1 to n **do**

        If A[i]=target **then**

            return i

        return -1

    End for

Linear Search: Example

- `my_array`

7	12	5	22	13	32
1	2	3	4	5	6

- **target = 13**

Linear Search: Example

`my_array`

7	12	5	22	13	32
1	2	3	4	5	6

target = 13

`my_array`

7	12	5	22	13	32
1	2	3	4	5	6

target = 13

Linear Search: Example

my_array

7	12	5	22	13	32
1	2	3	4	5	6

target = 13

my_array

7	12	5	22	13	32
1	2	3	4	5	6

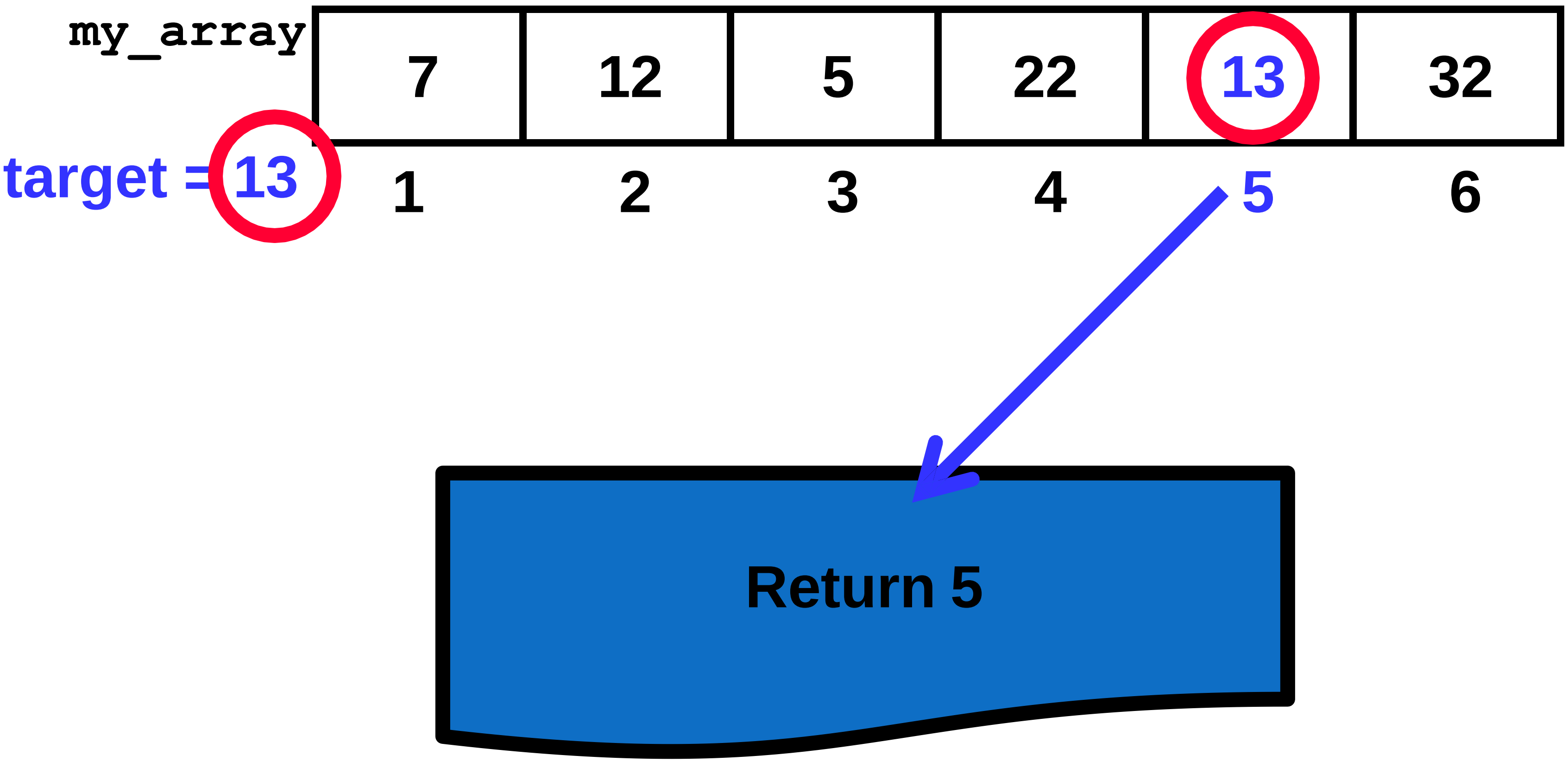
target = 13

my_array

7	12	5	22	13	32
1	2	3	4	5	6

target = 13

Linear Search: Example



Linear Search Analysis: Best Case


```
Algorithm Linear_search ( A, n, target )
    For i=1 to n do
        If A[i]=target then
            return i
        return -1
    End for
```

Best Case:
1 comparison

Best Case: match with the first item $\theta(1)$

7	12	5	22	13	32

target = 7

Linear Search Analysis: Worst Case


```
Algorithm Linear_search (A, n, target )
        For i=1 to n do
            If A[i]=target then
                return i
            return -1
        End for
```

Worst Case:
N comparisons

Worst Case: match with the last item (or no match)= $\boldsymbol{\theta}$**(N)**

7	12	5	22	13	32

target = 32

Recursive solution

Linear Search pseudo-code: Recursive solution


```
Algorithm Linear_search (A, n, target )
    If n<=0 then
        return -1
    Else if A[n-1]=target then
        return n-1
    Else
        return Linear_search (A, n-1, target )
```

Linear Search analysis: Recursive solution


```
Algorithm Linear_search ( A, n, target )
      If n<=0 then
            return -1
      Else if A[n-1]=target then
            return n-1
      Else
            return Linear_search ( A, n-1, target )
```

- Recurrence relation for iterative Linear search
  - $T(n) = T(n-1) + c$
- The cost of searching n elements is the cost of looking at 1 element, plus the cost of searching n-1 elements

Solve the Recurrence

- **Solve this recurrence**

  T(N) = T(N-1) + c ;T(0) = -1

- **Use substitution (iteration) method**

$$T(N) = T(N-1) + c \quad \rightarrow (1)$$

$$T(N-1) = T(N-2) + c \quad \text{;substitute in (1)}$$

$$T(N) = T(N-2) + c + c = T(N-2) + 2c \quad \rightarrow (2)$$

$$T(N-2) = T(N-3) + c \quad \text{;substitute in (2)}$$

$$T(N) = T(N-3) + c + 2c = T(N-3) + 3c \quad \rightarrow (3)$$

$$T(N-3) = T(N-4) + c \quad \text{;substitute in (3)}$$

$$T(N) = T(N-4) + c + 3c = T(N-4) + 4c \quad \rightarrow (4)$$

Solve the Recurrence

Generally at K:

$$T(N) = T(N-K) + K \times c$$

<u>Termination:</u>

$$T(N-K) = T(0) \rightarrow N - K = 0$$

K=N

Substitute in general formula:

$$T(N) = T(0) + N \times c$$

T(N) = T(0) + N X C =NC+ C_0 = $\boldsymbol{\theta}$(N)

(T(0) is some constant, c_0)

19

Binary Search

Binary Search

- **Idea**
  - Requires a sorted array or a *binary search tree*.
  - Cuts the "search space" in half each time.
  - Keeps cutting the search space in half until the target is found or has exhausted the all possible locations.

7	12	42	59	71	86	104	212

Binary search

- **Binary search is a better search algorithm**
  - Of course we could use our simpler search and traverse the array. But we can use the fact that the array is sorted to our advantage. This will allow us to reduce the number of comparisons

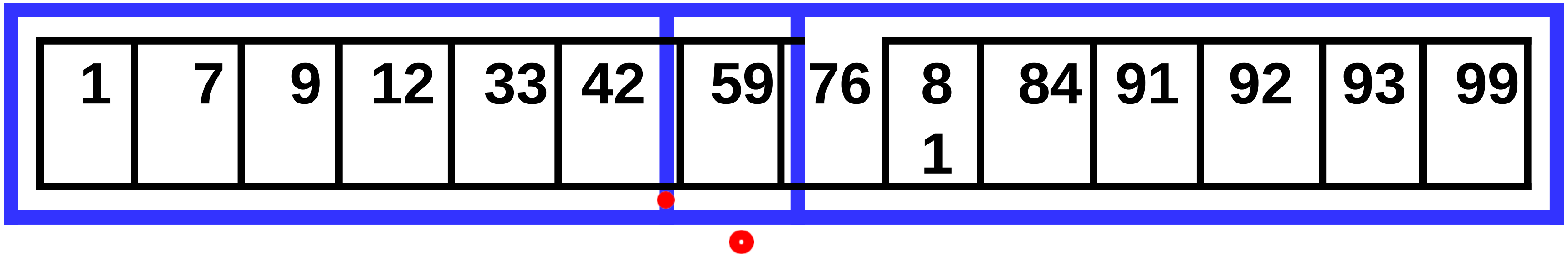
Binary Search Algorithm

- Look at “middle” element

- If no match then

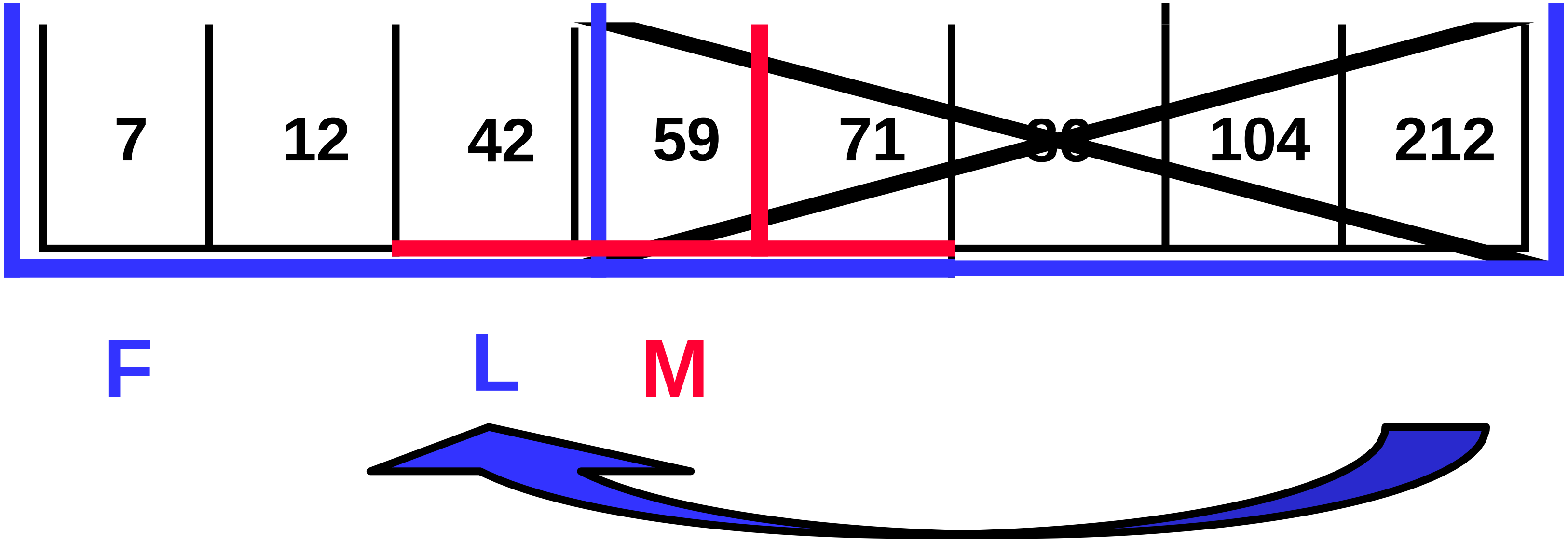
  look *left* (if need smaller) or

  right (if need larger)



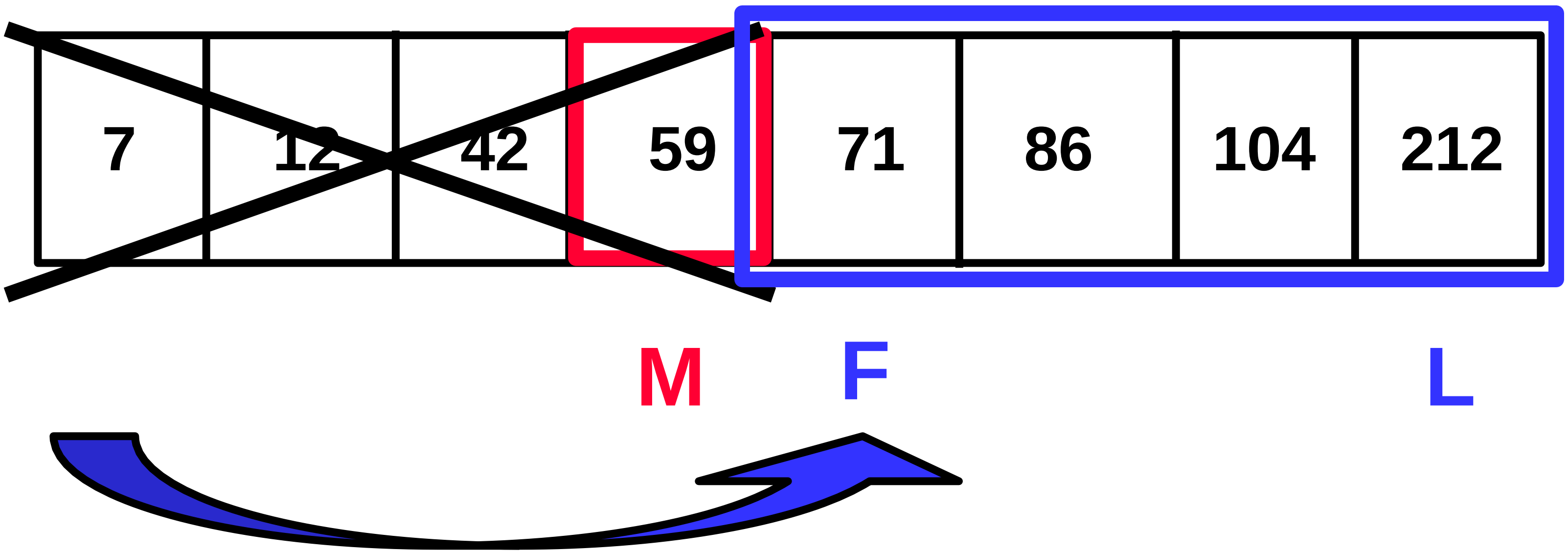
Binary Search Algorithm

- **<u>Looking left:</u>**
  - Use indices "first" and "last" to keep track of where we are looking
  - Move left by setting first= middle + 1

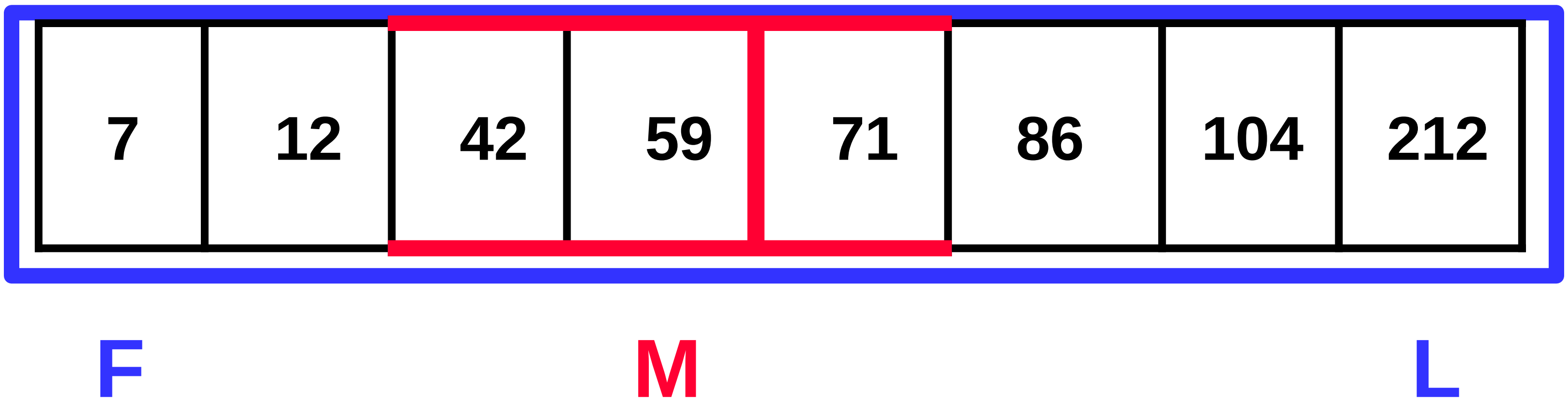


Binary Search algorithm

- **<u>Looking right:</u>**
  - Use indices "first" and "last" to keep track of where we are looking
  - Move right by setting last = middle – 1

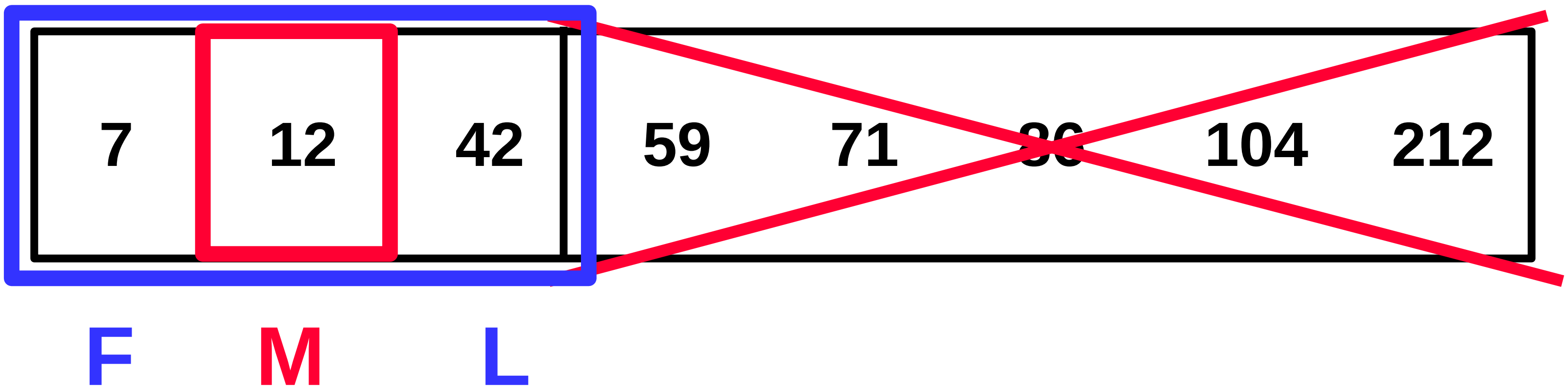


Binary Search Example - Found



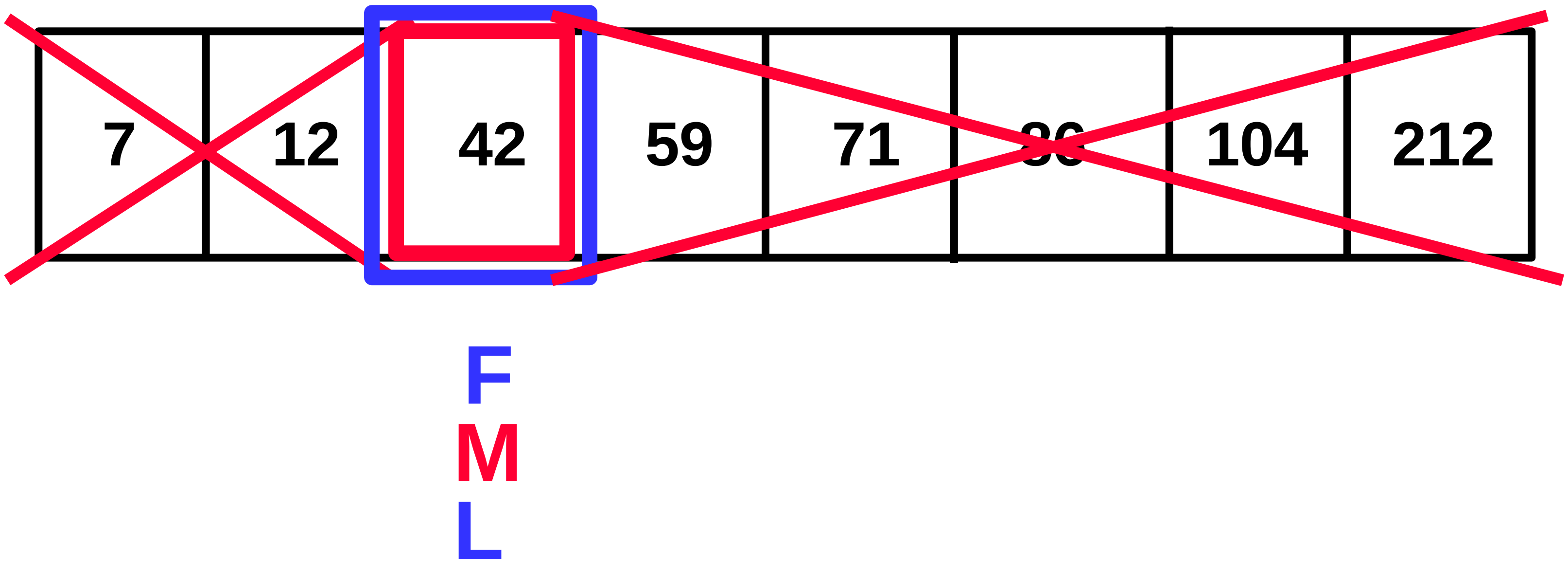
Looking for 42

Binary Search Example - Found



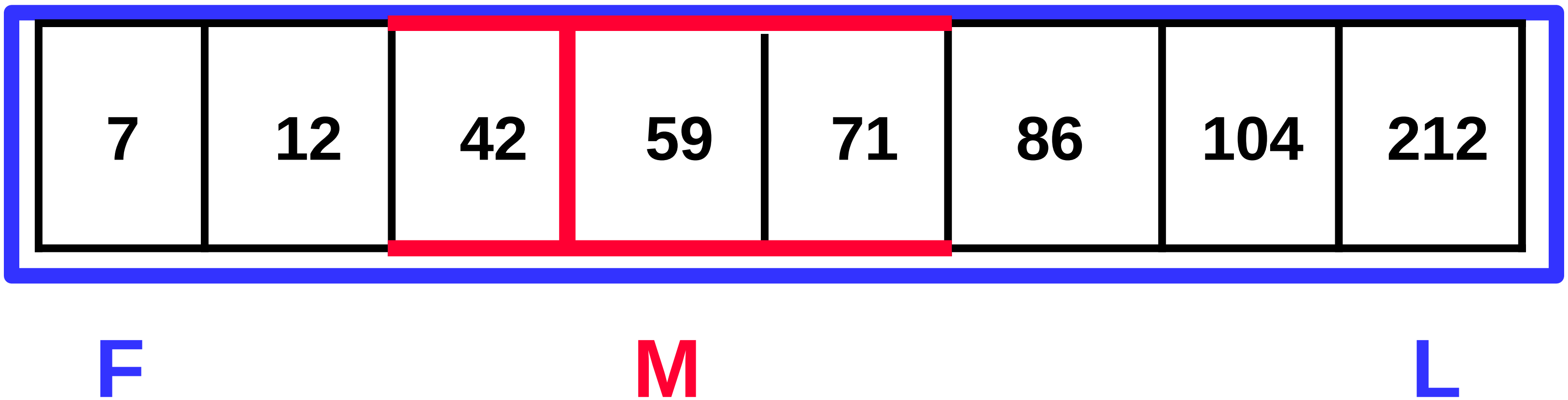
Looking for 42

Binary Search Example - Found



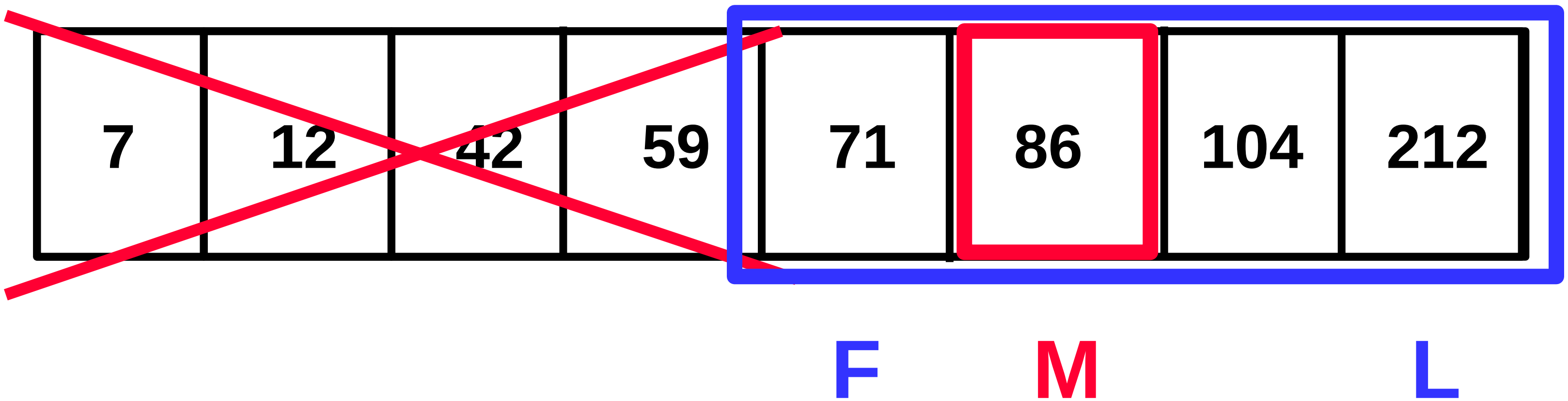
42 found – in 3 comparisons

Binary Search Example – Not Found



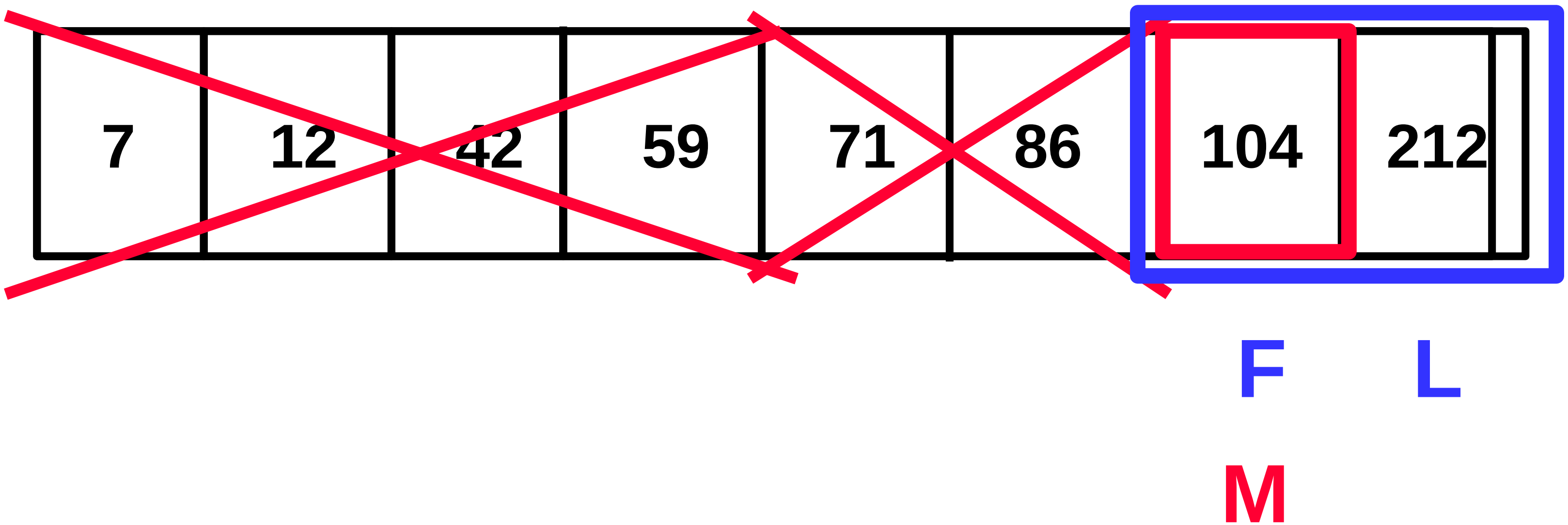
Looking for 89

Binary Search Example – Not Found



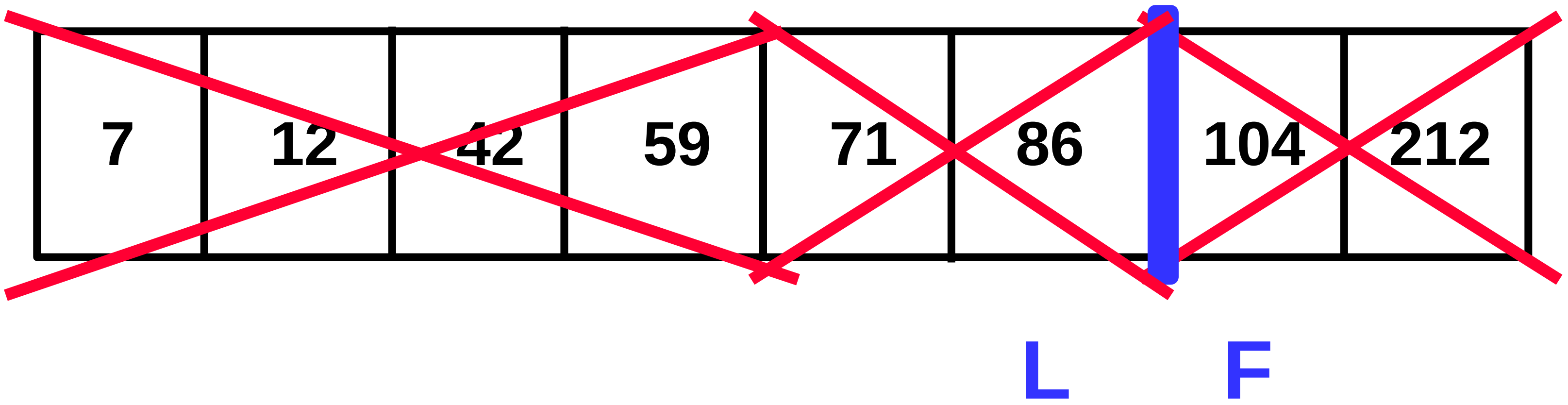
Looking for 89

Binary Search Example – Not Found



Looking for 89

Binary Search Example – Not Found



89 not found – 3 comparisons

Binary Search

- **There are two solutions**
  - Iterative solution
  - Recursive solution

Iterative solution

Binary Search pseudo-code: Iterative solution


```
Algorithm Binary_search ( A, n, target )
        first=1
        last=n
        While first<=last do
            mid=(first+last)/2
            If A[mid]=target then
                return mid
            Else if A[mid]<target then
                first=mid+1
            Else
              last=mid-1
         End while
           return -1
```

Binary Search Analysis: Best Case


```
Algorithm Binary_search ( A, n, target )
        first=1
        last=n
        While first<=last do
            mid=(first+last)/2
              If A[mid]=target then
                     return mid
              Else if A[mid]<target then
                  first=mid+1
              Else
                last=mid-1
        End while
           return -1
```

Best Case:
  1 comparison

Best Case: **match from the firs comparison $\theta(1)$**

1	7	9	12	33	42	59	76	81	84	91	92	93	99

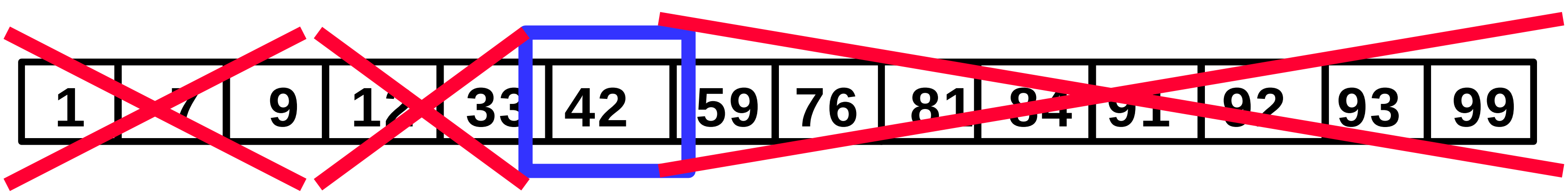
Target: 59

Binary Search Analysis: WorstCase


```
Algorithm Binary_search ( A, n, target )
        first=1
        last=n
        While first<=last do
            mid=(first+last)/2
            If A[mid]=target then
                    return mid
             Else if A[mid]<target then
                  first=mid+1
             Else
                 last=mid-1
         End while
           return -1
```

How many comparisons??

Worst Case: divide until reach one item, or no match.



Binary Search Analysis: Worst Case

- With each comparison we throw away ½ of the list

N .... **1 comparison**

N/2 .... **1 comparison**

N/4 .... **1 comparison**

N/8 .... **1 comparison**

⋮

1 .... **1 comparison**

Termination:

$$\frac{N}{2^{i-1}} = 1 \rightarrow N = 2^{i-1}$$

$i = Log_2 (N+1)$ i:iteration

Number of steps is at most ➔ *Log_2N*

Worst case $\rightarrow \Theta(Log_2N)$

Recursive solution

Binary Search pseudo-code: Recursive solution

Algorithm Binary_search (A, first, last, target)
    mid=(first+last)/2
    If first > last **then**
        return -1
    Else
        If A[mid]=target **then**
            return mid
        Else if target< A[mid] **then**
            return Binary_search (A, first, mid-1, target)
        Else
            return Binary_search (A, mid+1, last, target)

Binary Search analysis: Recursive solution

- Recurrence relation for iterative Binary search

$$T(N) = 1+T(N/2)$$

- The cost of searching n elements is the cost of looking at mid element, plus the cost of searching N/2 elements (ONE OF THE haves of array)

Solve the Recurrence

$T(N) = 1 + T(N/2)$

- **Use Master`s theorem**

  a = 1, b = 2, f(N) = 1

  Compare f(N) vs. $N^{\log_b a}$

  $N^{\log_b a} = N^{\log_2 1} = N^0 = 1$ vs. $f(N) = 1$

  $N^{\log_b a}$ = f(N) ➔ It's case (2),

Case 2

$\Rightarrow$ f(n) = Θ(n) $\Rightarrow$ T(n) = Θ(logn)

Summary

- Binary search reduces the work by half at each comparison
- If array is not sorted ➔ Linear Search
  - Best Case $\Theta(1)$
  - Worst Case $\Theta(N)$
- If array is sorted ➔ Binary search
  - Best Case $\Theta(1)$
  - Worst Case $\Theta(Log_2N)$

Thanks