Search: Linear and Binary

Instructor: Krishna Venkatasubramanian

CSC 212

Announcements

- No class next week, however the lab session will continue
- Assignment 2 release postponed to Nov 5, 2019
- Assignment 1 deadline extended to Oct 31, 6pm (NOTE THE TIME)
- MY office hours cancelled today

Searching

- **Given a collection and an element (key) to find…**
- **Output**
  - **Print a message (“Found”, “Not Found)**
  - **Return a value (position of key)**
- **Don’t modify the collection in the search!**

Searching Example
(Linear Search)

Linear Search: A Simple Search

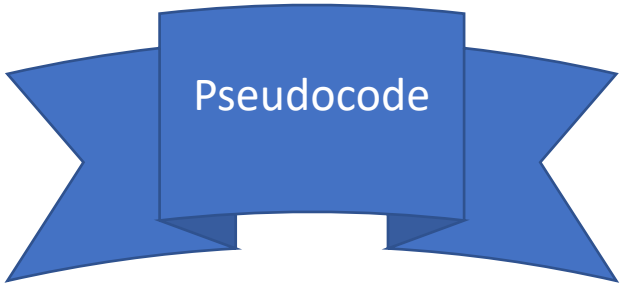
- **A search traverses the collection until**
  - **The desired element is found**
  - **Or the collection is exhausted**

- **If the collection is ordered, I might not have to look at all elements**
  - **I can stop looking when I know the element cannot be in the collection.**

Un-Ordered Iterative Array Search

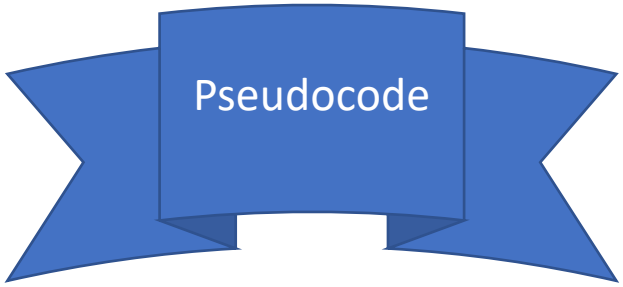
```
Search(A,target)

        for i = 1 … N:
                if A[i] = target:
                        return "Found"

        return("Not found")
```

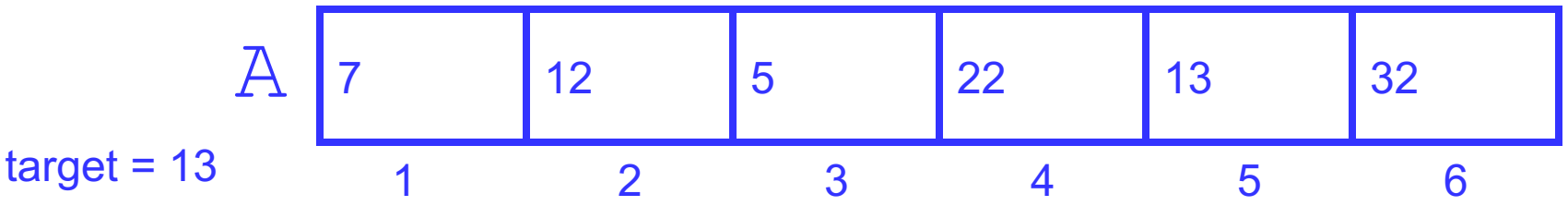
Scan the array



Un-Ordered Iterative Array Search

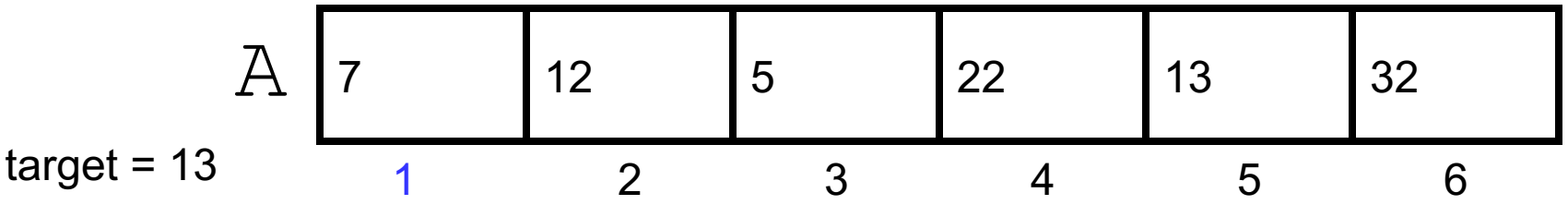
```
Search(A,target)

        for i = 1 … N:
                if A[i] = target:
                        return "Found"

        return("Not found")
```





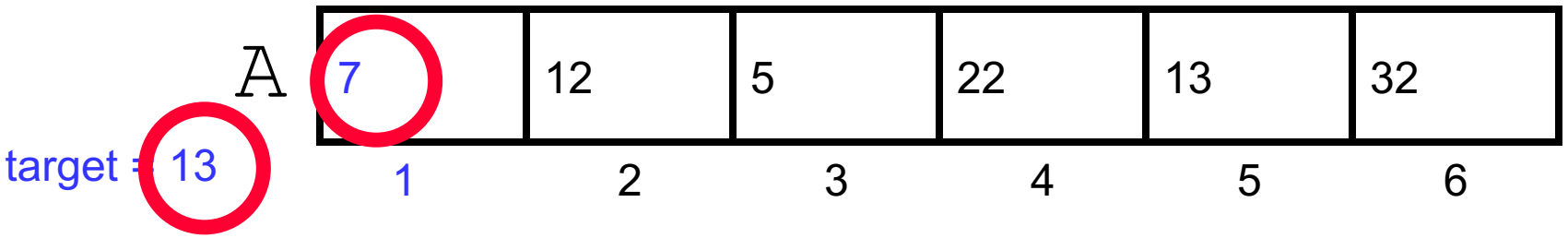
Un-Ordered Iterative Array Search

```
Search(A,target)

        for i = 1 … N:
                if A[i] = target:
                        return "Found"

        return("Not found")
```



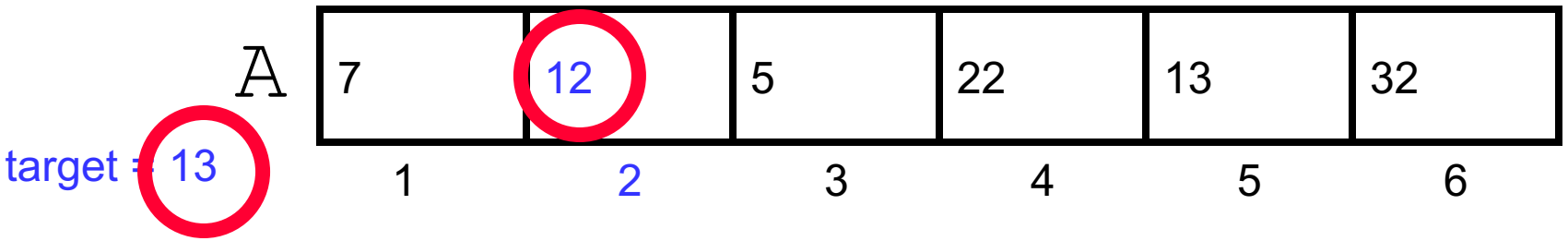
Un-Ordered Iterative Array Search

```
Search(A,target)

        for i = 1 … N:
                if A[i] = target:
                        return "Found"

        return("Not found")
```



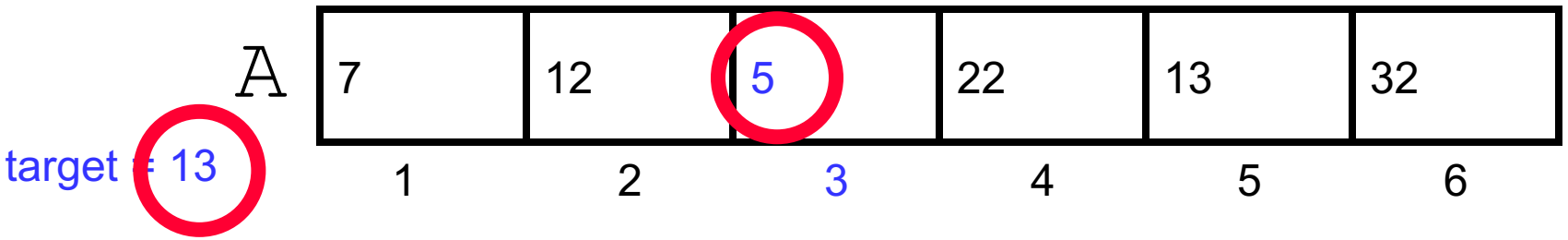
Un-Ordered Iterative Array Search

```
Search(A,target)

        for i = 1 … N:
                if A[i] = target:
                        return "Found"

        return("Not found")
```



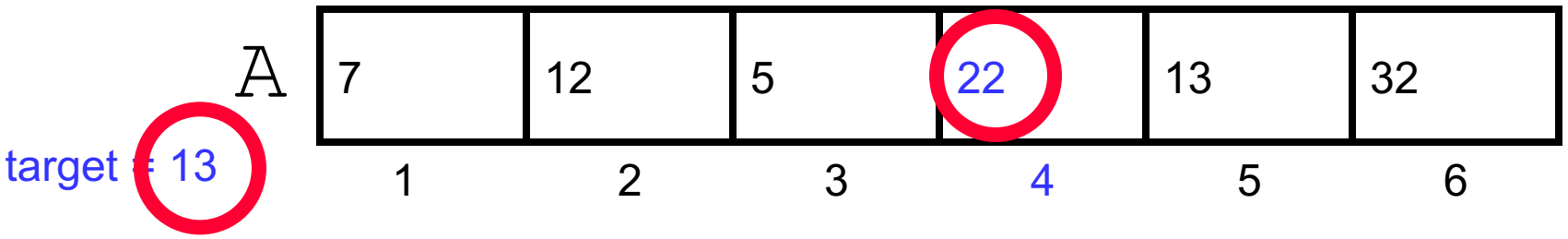
Un-Ordered Iterative Array Search

```
Search(A,target)

        for i = 1 … N:
                if A[i] = target:
                        return "Found"

        return("Not found")
```



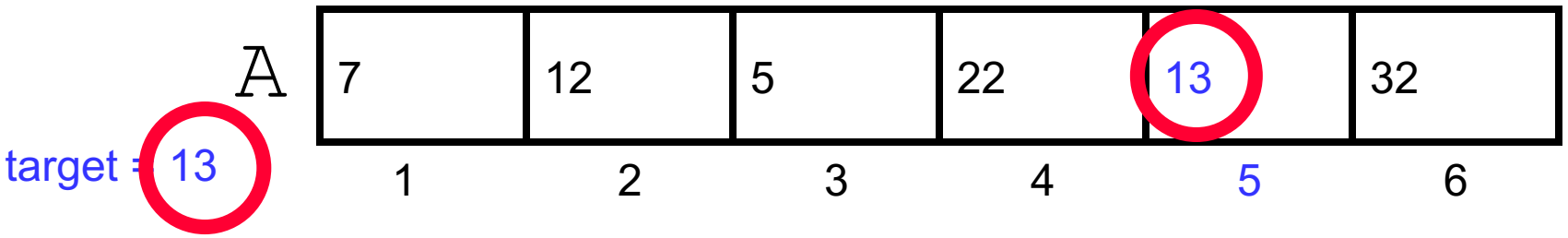
Un-Ordered Iterative Array Search

```
Search(A,target)

        for i = 1 … N:
                if A[i] = target:
                        return "Found"

        return("Not found")
```



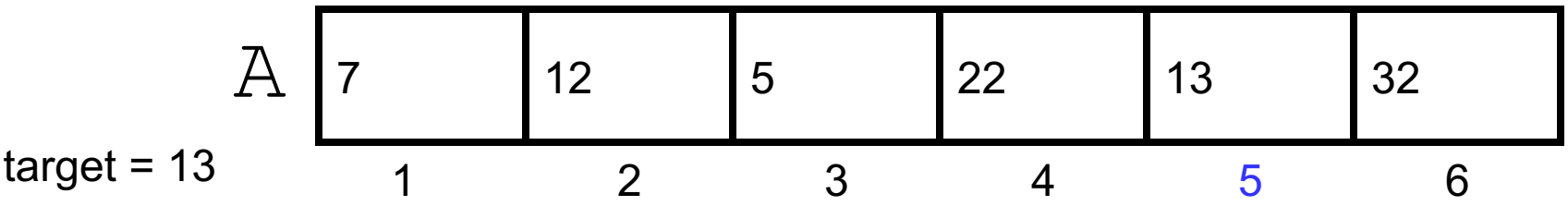
Un-Ordered Iterative Array Search

```
Search(A,target)

        for i = 1 … N:
                if A[i] = target:
                        return "Found"

        return("Not found")
```



Un-Ordered Iterative Array Search

```
Search(A,target)

        for i = 1 … N:
                if A[i] = target:
                        return "Found"
```

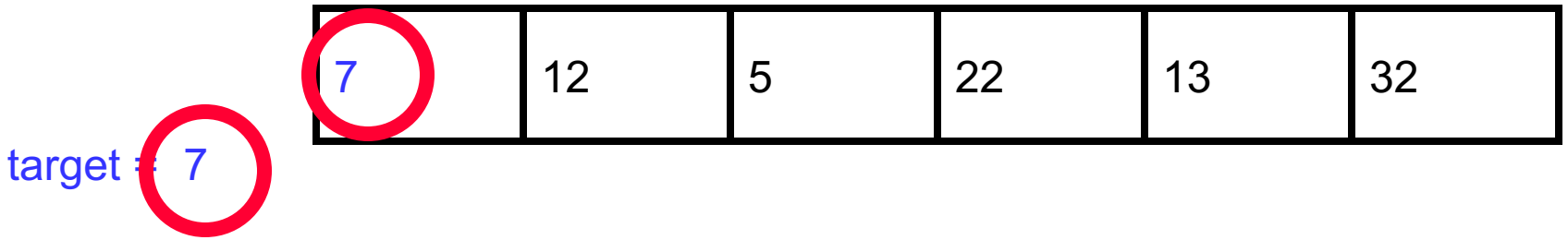


Linear Search Analysis: Best Case

```
Search(A,target)

        for i = 1 … N:
                if A[i] = target:
                        return "Found"

        return("Not found")
```

Best Case:
1 comparison

Best Case: match with the first item

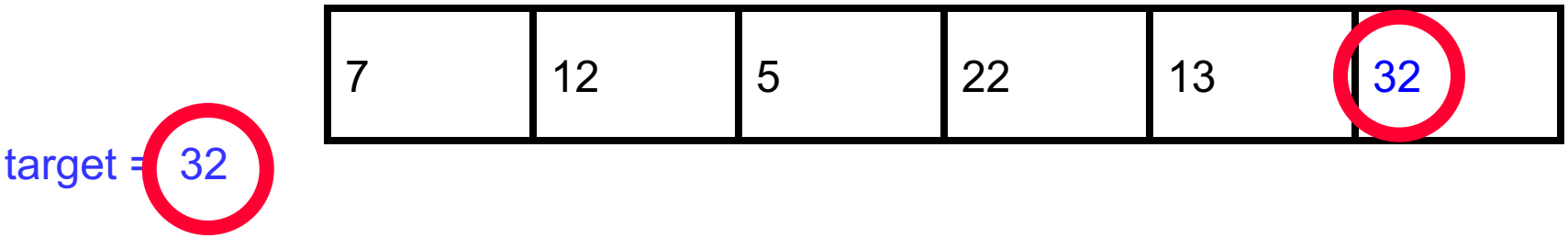


Linear Search Analysis: Worst Case

```
Search(A,target)

        for i = 1 … N:
                if A[i] = target:
                        return "Found"

        return("Not found")
```

Worst Case:
N comparisons

Worst Case: match with the last item (or no match)

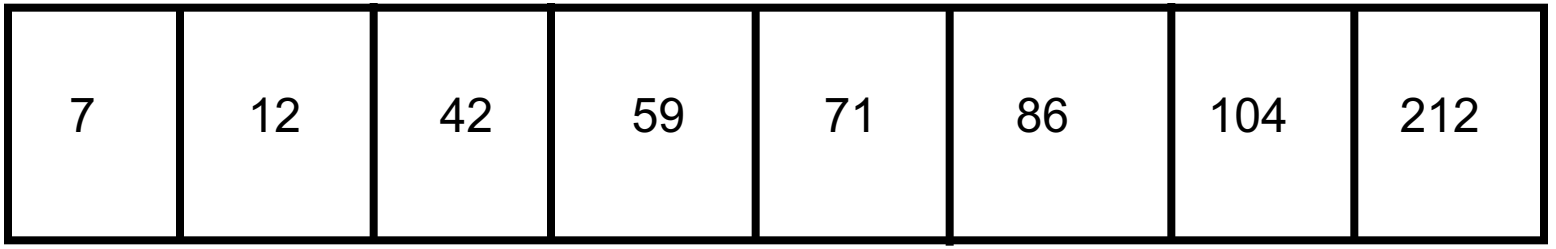


Searching Example
(Binary Search on Sorted List)

The Scenario

- **We have a sorted array**

- **We want to determine if a particular element is in the array**
  - **Once found, print or return (index, boolean, etc.)**
  - **If not found, indicate the element is not in the collection**



A Better Search Algorithm

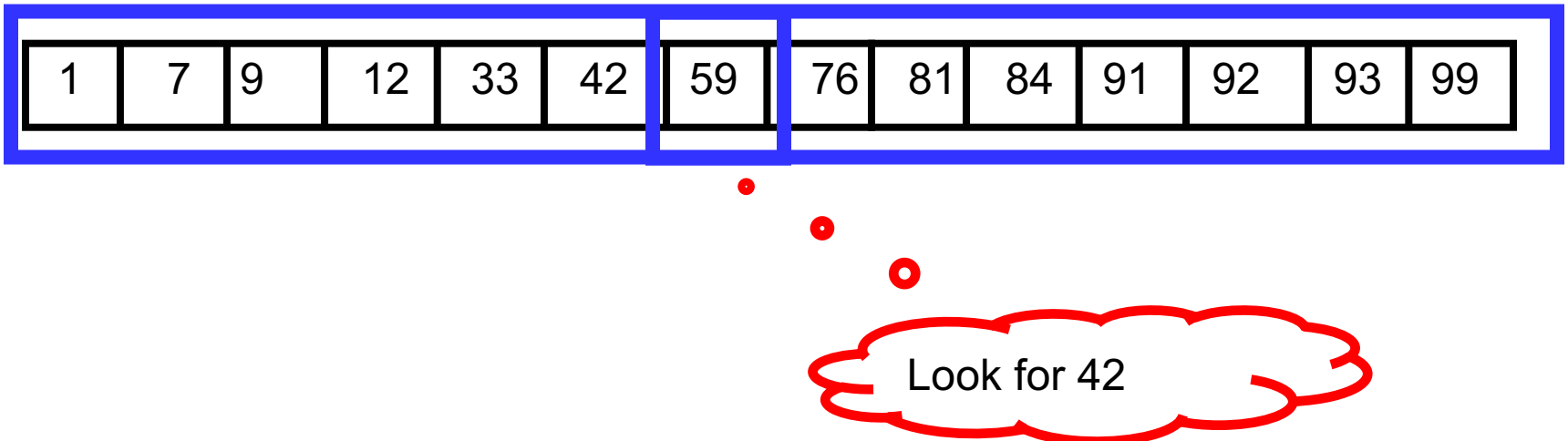
- **Of course we could use our simpler search and traverse the array**
- **But we can use the fact that the array is sorted to our advantage**
- **This will allow us to reduce the number of comparisons**

Binary Search

- **Requires a sorted array or a *binary search tree.***
  (We will see this later in the course)
- **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.**

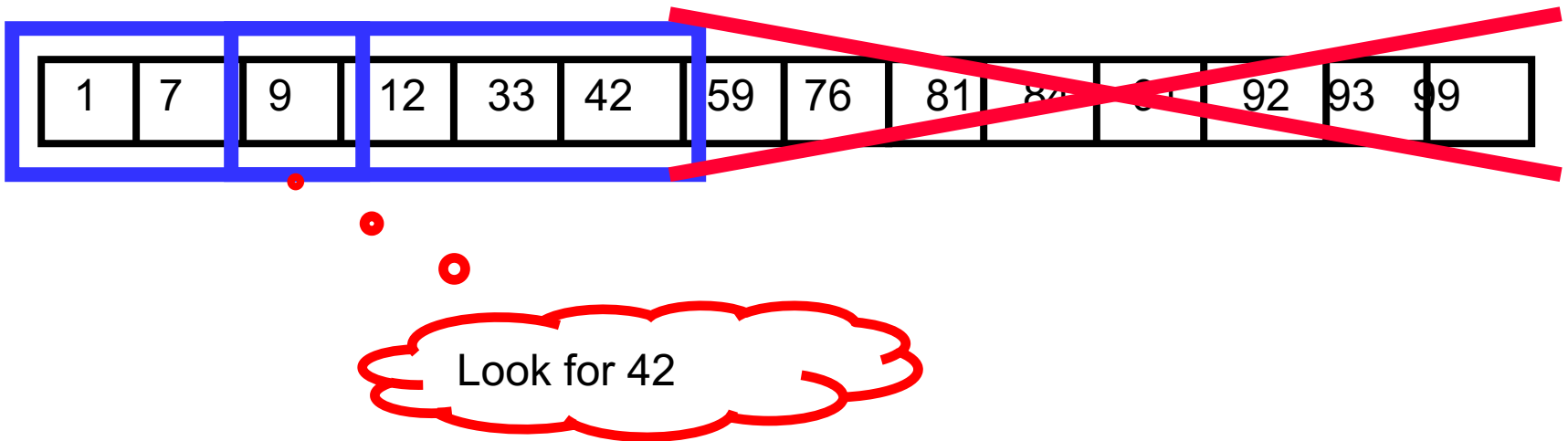
Binary Search Algorithm

```
Searching for X
  look at "middle" element
  if no match then
      look @ smaller half if X < middle
   or @ larger half if X > middle
```



The Algorithm

```
Searching for X
  look at "middle" element
  if no match then
    look @ smaller half if X < middle
    or @ larger half if X > middle
```



The Algorithm

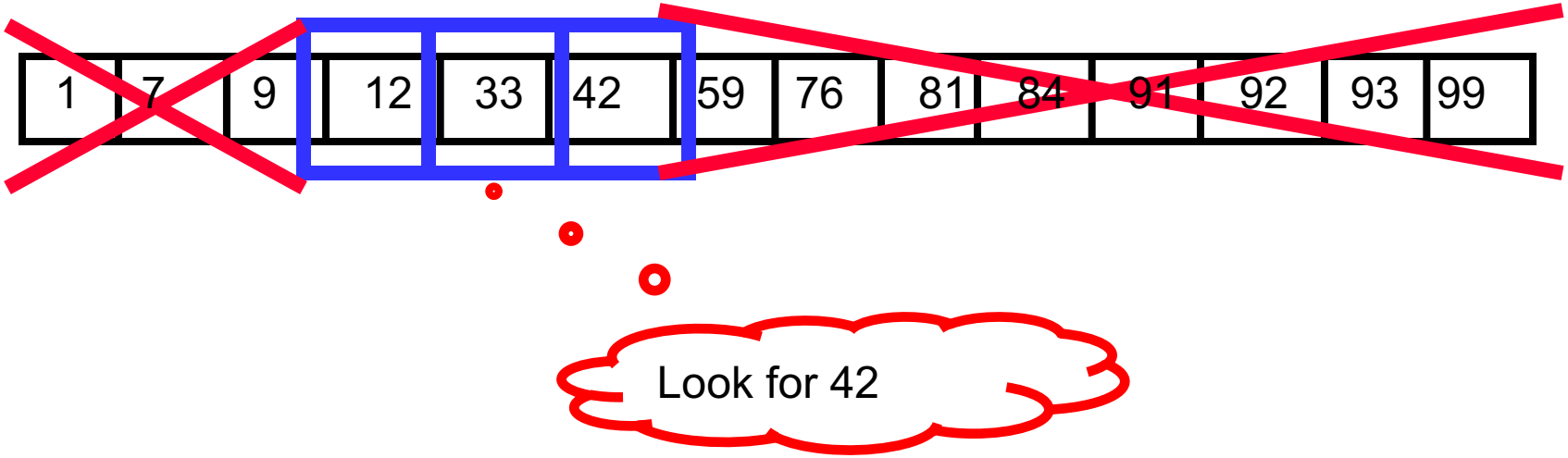
Searching for X

look at "middle" element

if no match then

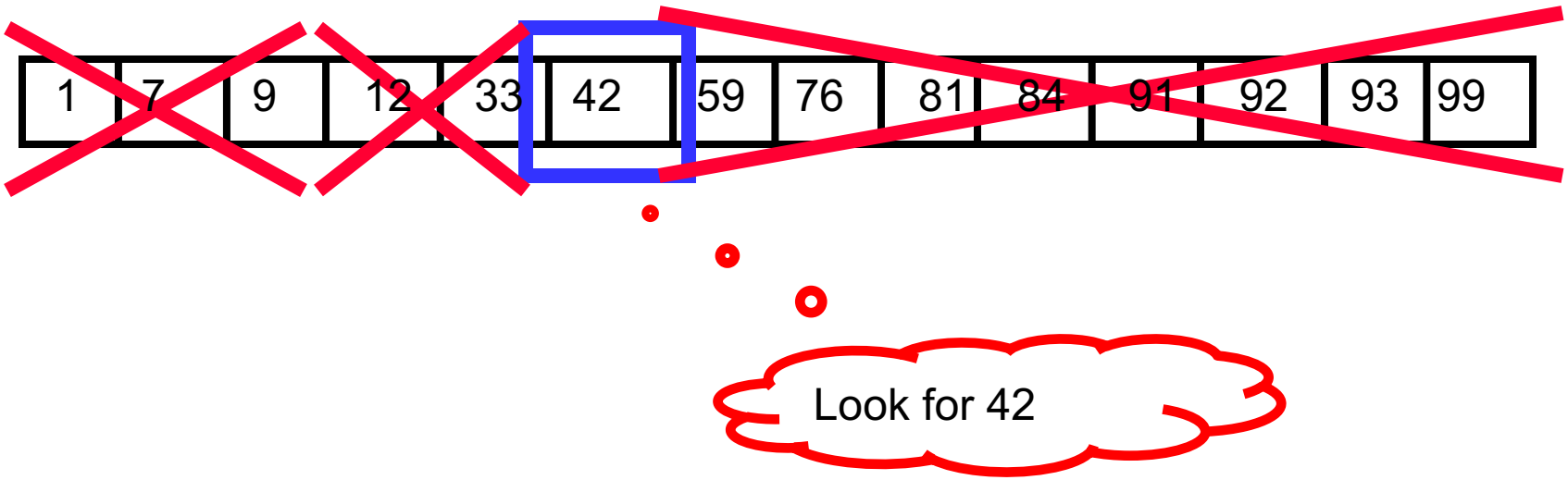
look @ smaller half if X < middle

or @ larger half if X > middle



The Algorithm

```
Searching for X
  look at "middle" element
  if no match then
    look @ smaller half if X < middle
    or @ larger half if X > middle
```

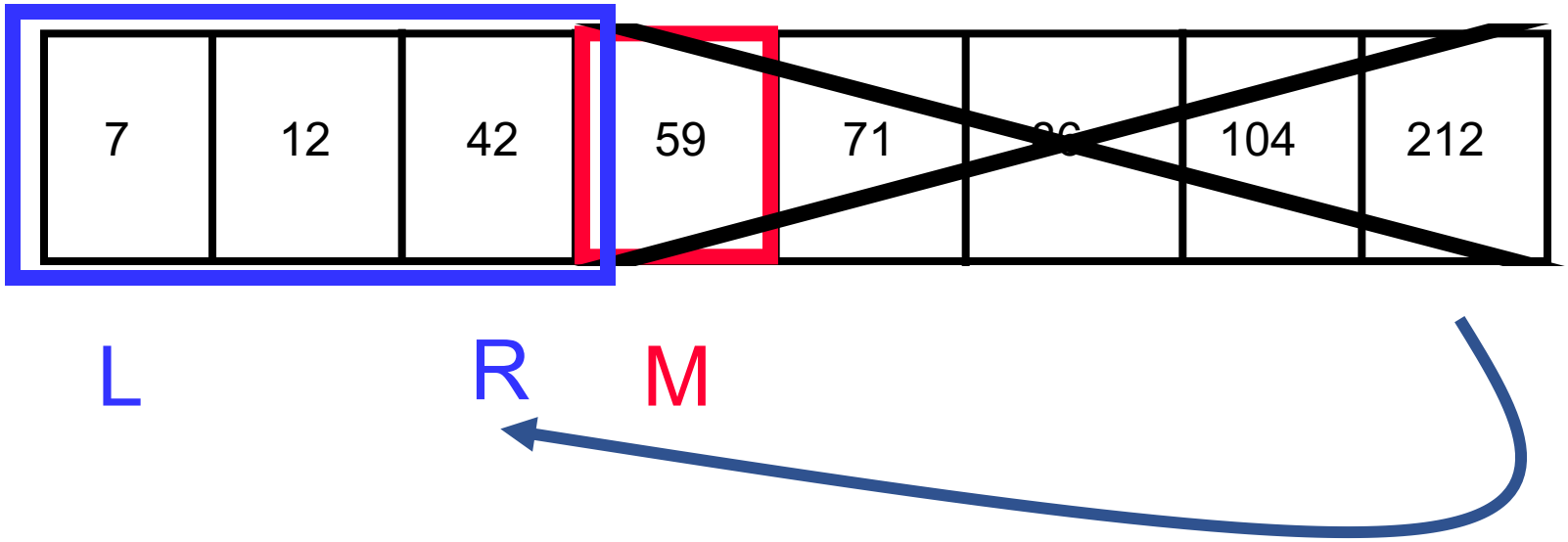


The Binary Search Algorithm (Pseudocode)

```
Searching for X
  calculate middle position
  if (Left-index and Right-index have "crossed")
  then
       "Item not found"
  elseif (element at middle = to_find) then
       "Item Found"
  elseif to_find < element at middle then
       Move Right-index to middle -1
  else
       Move the Left-index to middle +1
```

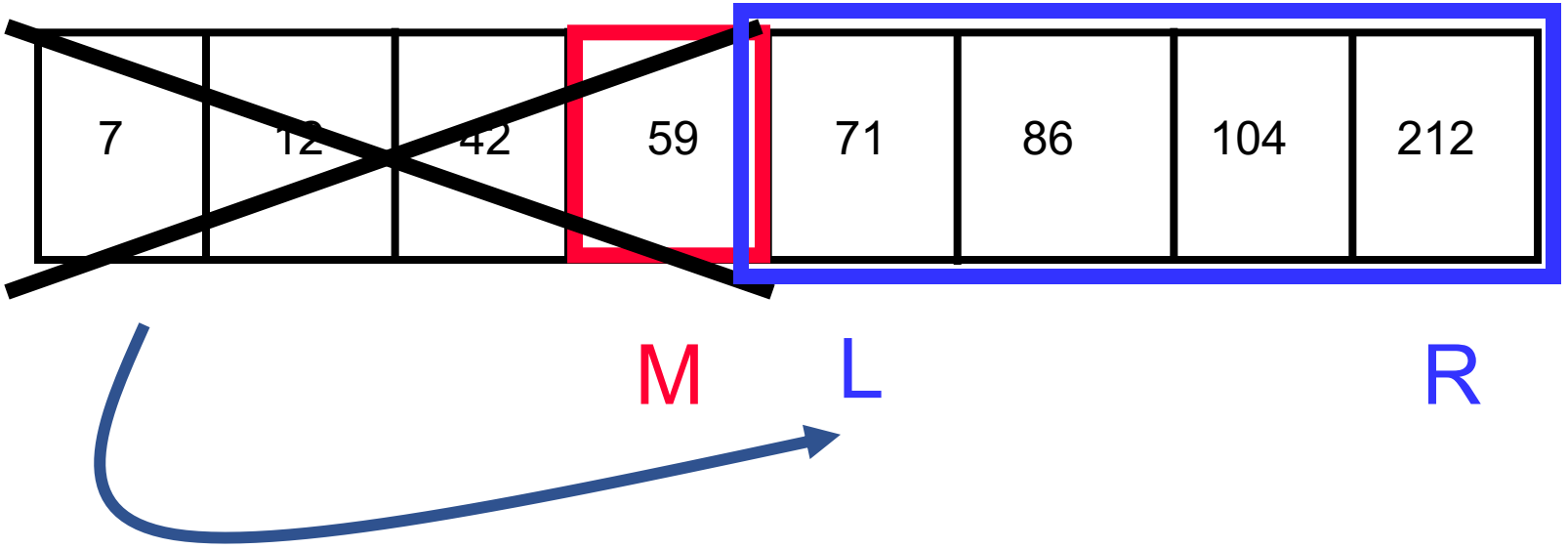
Visually: Moving Right Index

- **Use indices "Left (L)" and "Right(R)" to keep track of where we are looking**
- **Set Right = middle – 1. Leave Left alone.**

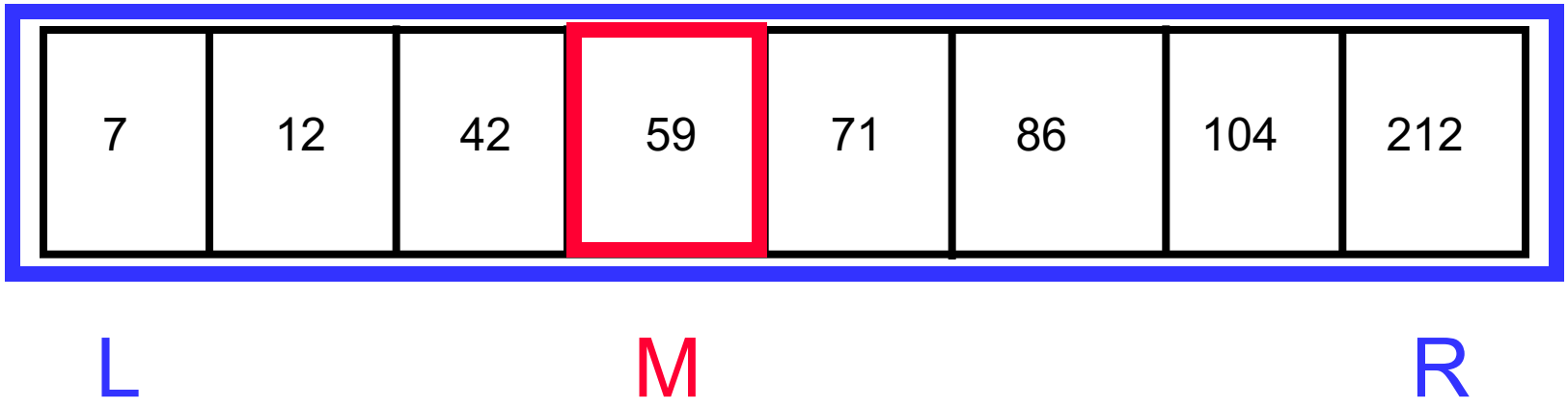


Visually: Moving Left Index

- **Use indices "Left" and "Right" to keep track of where we are looking**
- **Set Left = middle + 1. Leave Right alone.**

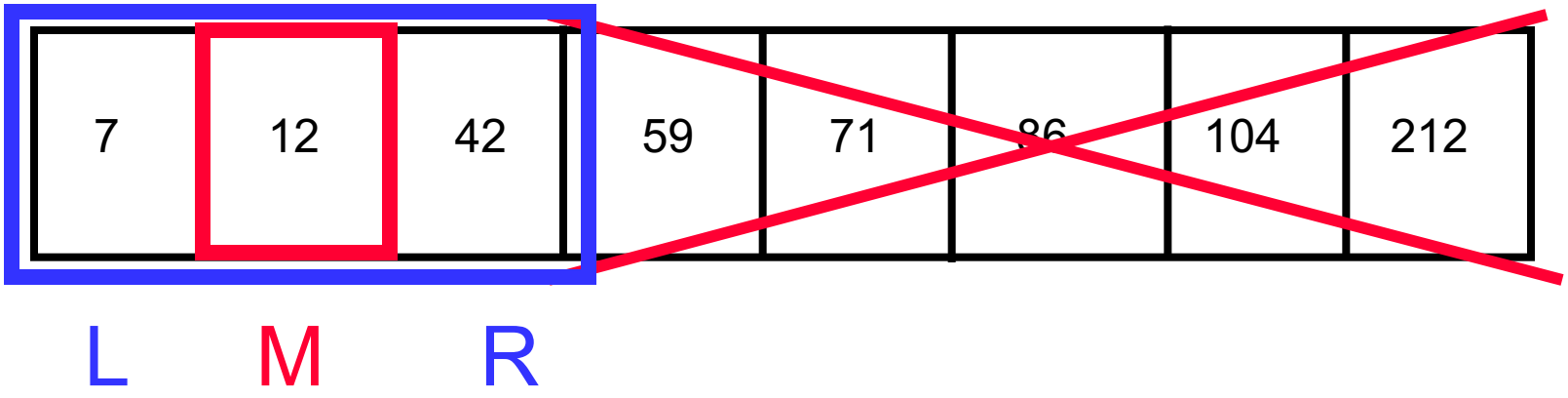


Binary Search Example – Found



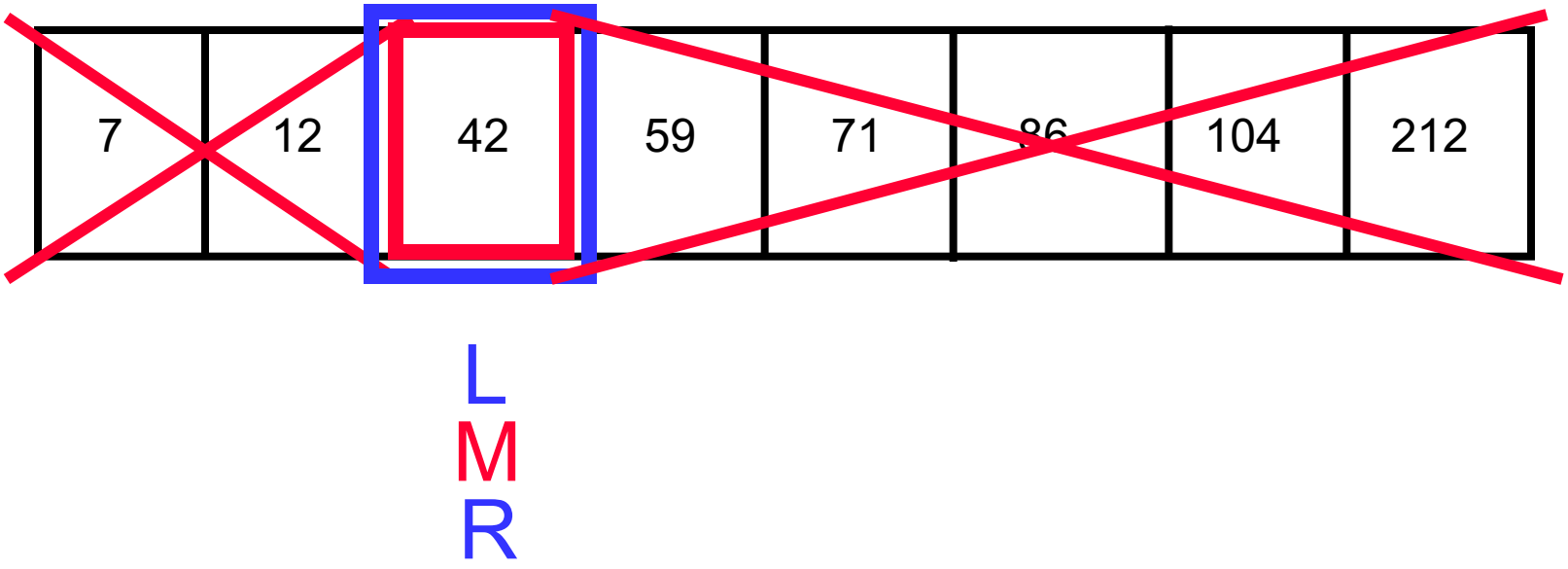
Looking for 42

Binary Search Example – Found



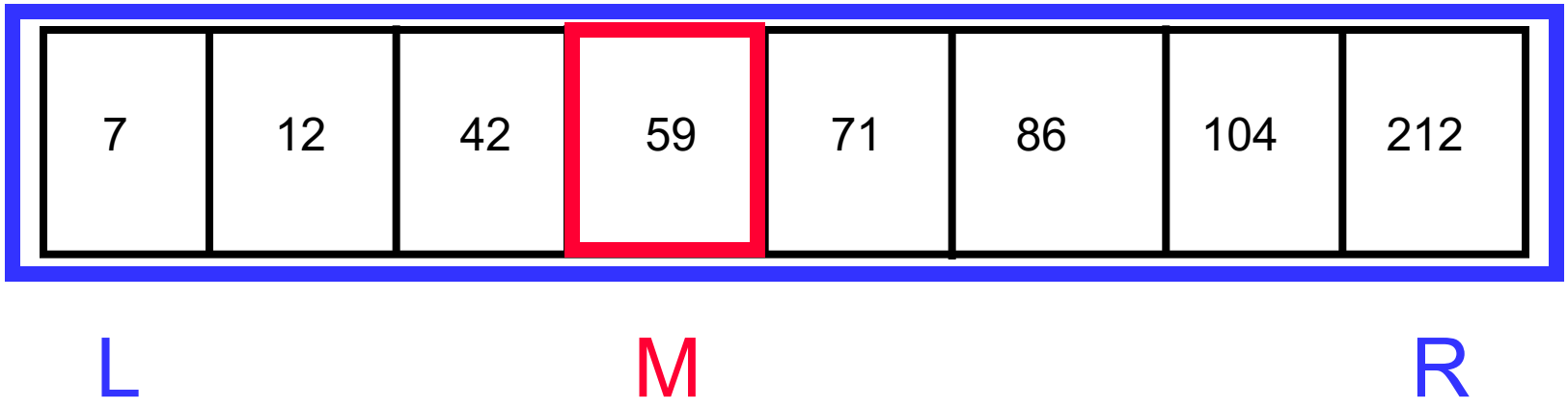
Looking for 42

Binary Search Example – Found



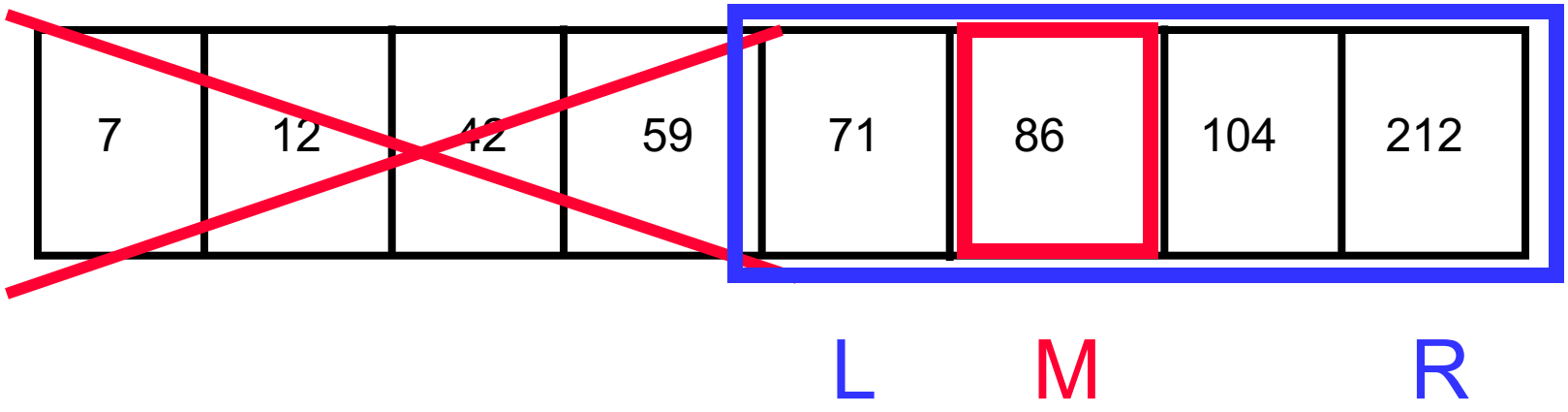
42 found – in 3 comparisons

Binary Search Example – Not Found



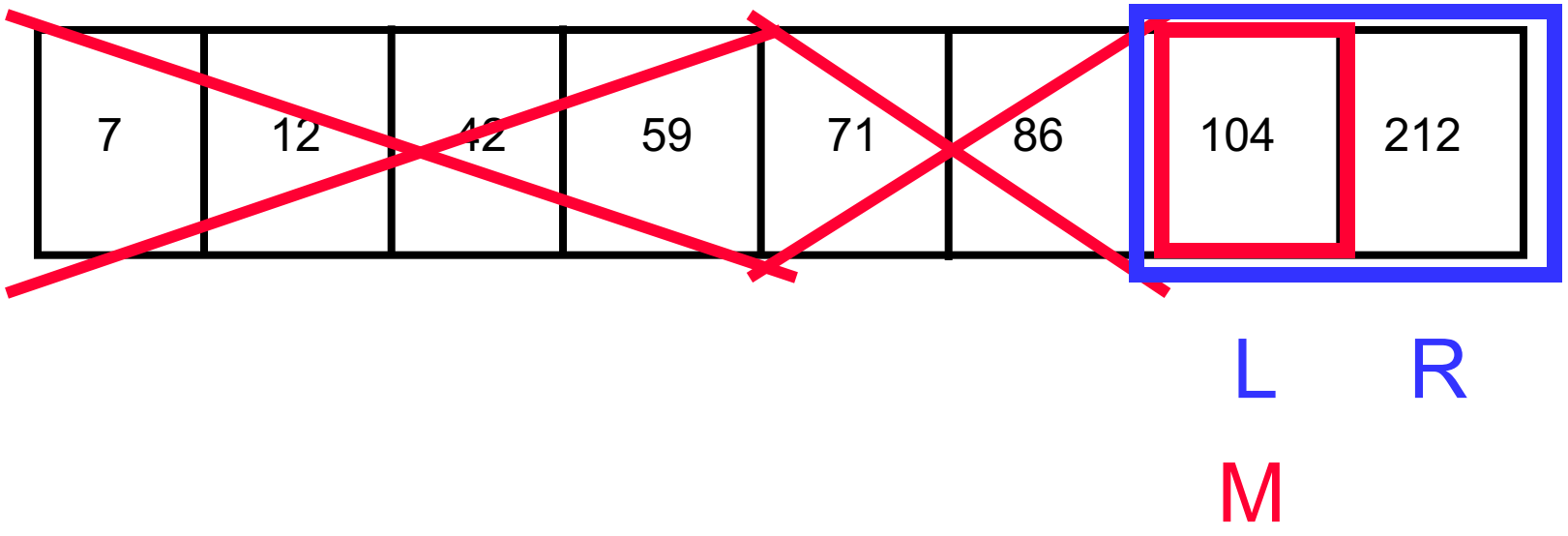
Looking for 89

Binary Search Example – Not Found



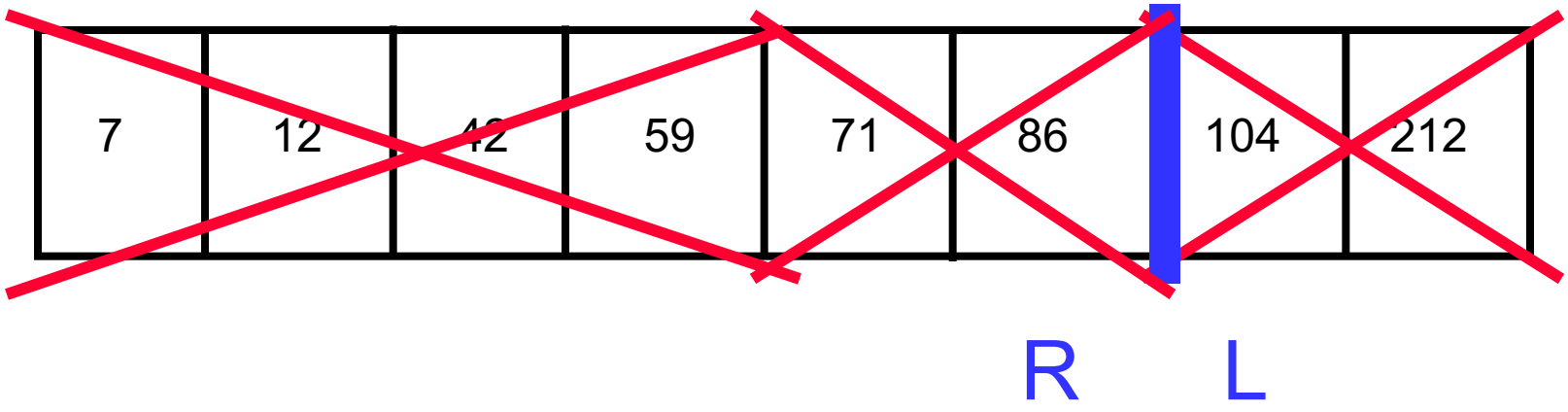
Looking for 89

Binary Search Example – Not Found



Looking for 89

Binary Search Example – Not Found



89 not found – 3 comparisons

Binary Search Function

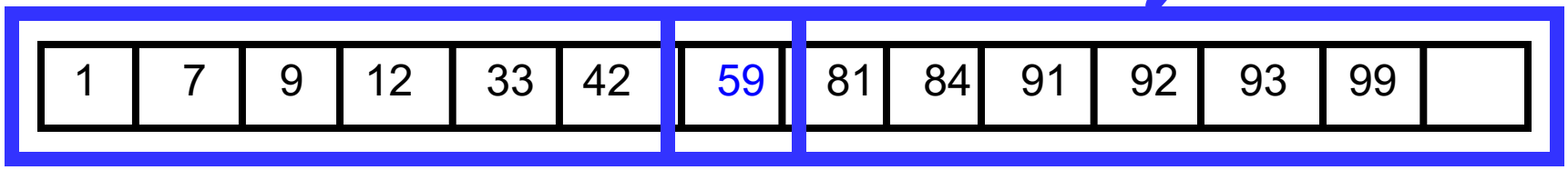
```
def BinarySearch(A, first, last, target):
  middle <- (first + last) / 2

  if first > last:
       return false
  if A[middle] = target:
       return true
  if target< A[middle]:
       return BinarySearch(A, first, middle-1, target)
  else:
       return BinarySearch(A, middle+1, last, target)
```

Binary Search Analysis: Best Case

```
def BinarySearch(A, first, last, target):
   middle <- (first + last) / 2

   if first > last:
        return false
   if A[middle] = target:
        return true
   if target< A[middle]:
        return BinarySearch(A, first, middle-1, target)
   else:
        return BinarySearch(A, middle+1, last, target)
```

Best Case: 1 comparison

Best Case: match from the firs comparison



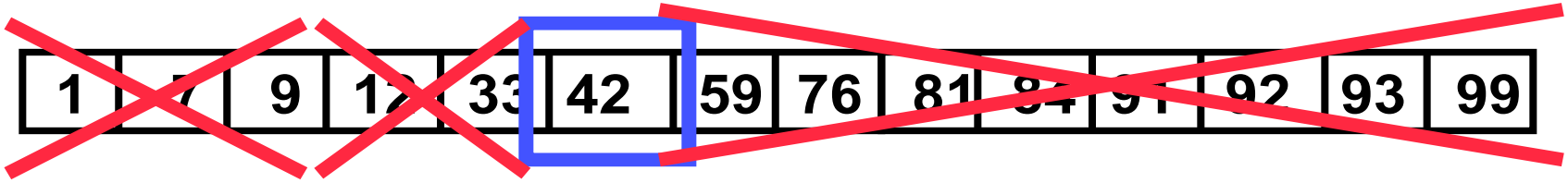
Target: 59

Binary Search Analysis: Worst Case

```
def BinarySearch(A, first, last, target):
    middle <- (first + last) / 2

    if first > last:
          return false
    if A[middle] = target:
          return true
    if target< A[middle]:
          return BinarySearch(A, first, middle-1, target)
    else:
          return BinarySearch(A, middle+1, last, target)
```

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

Summary

- **Binary search reduces the work by half at each comparison**

- **If array is not sorted ➔ Linear Search**
  - **Best Case O(1)**
  - **Worst Case O(N)**

- **If array is sorted ➔ Binary search**
  - **Best Case O(1)**
  - **Worst Case O(Log_2N)**


That's all Folks!
Any Question ?