# Data Structures and Algorithms

Khazhak Galstyan

# Session 4:
## Binary Search

# Searching Problem

# The Searching Problem (in a sorted array)

**INPUT**: a **sorted** list of n elements and **a single value**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 3 | 3 | 4 | 6 | 6 | 8 |

| |
|---|
| 6 |

# The Searching Problem (in a sorted array)

**INPUT**: a **sorted** list of n elements and **a single value**

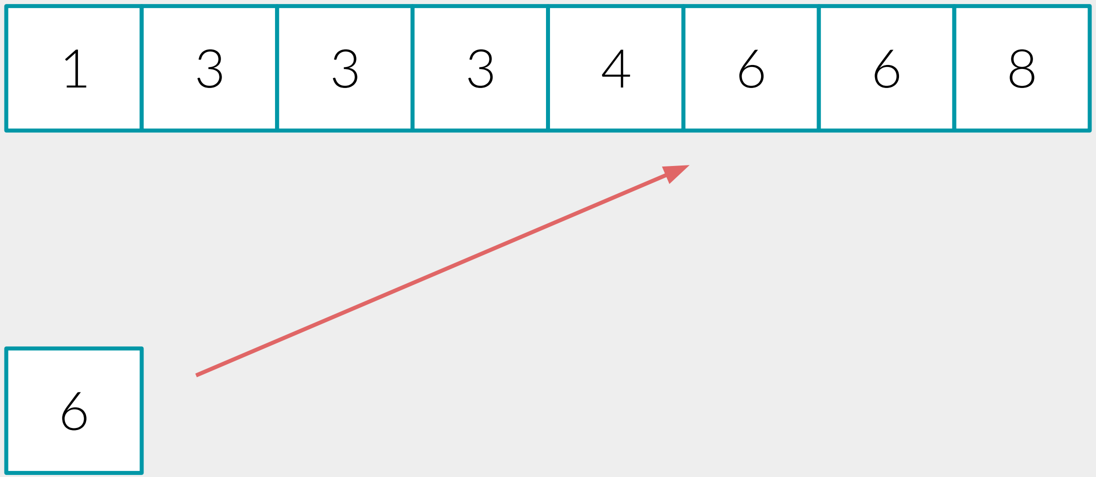
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 3 | 3 | 4 | 6 | 6 | 8 |

| |
|---|
| 6 |

**OUTPUT**: where is (or where would be) given value in a given list

# The Searching Problem (in a sorted array)

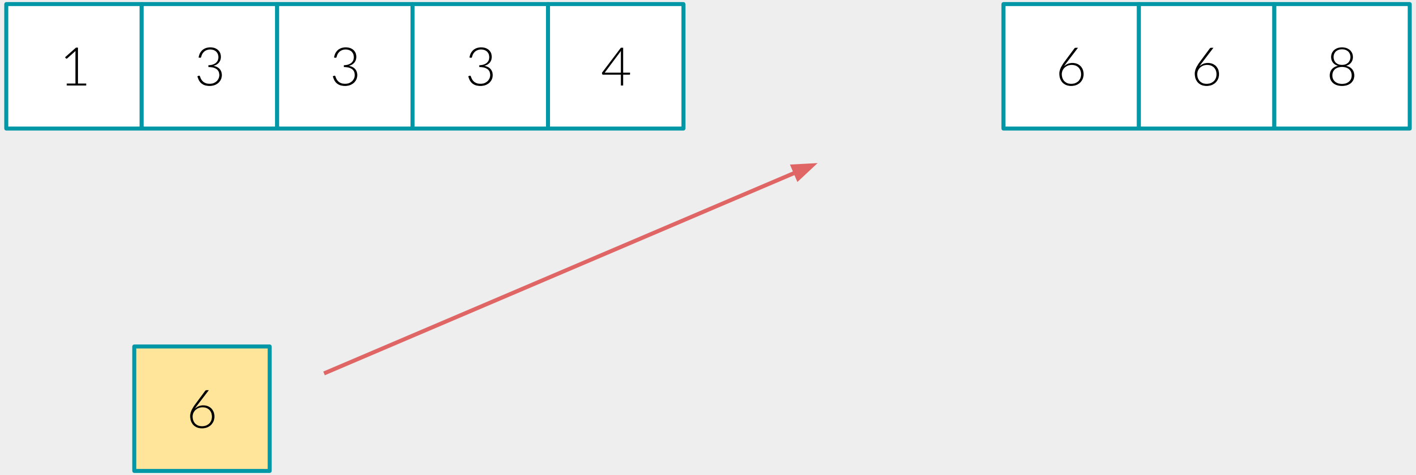
**INPUT**: a **sorted** list of n elements and **a single value**




**OUTPUT**: where is (or where would be) given value in a given list

# The Searching Problem (in a sorted array)

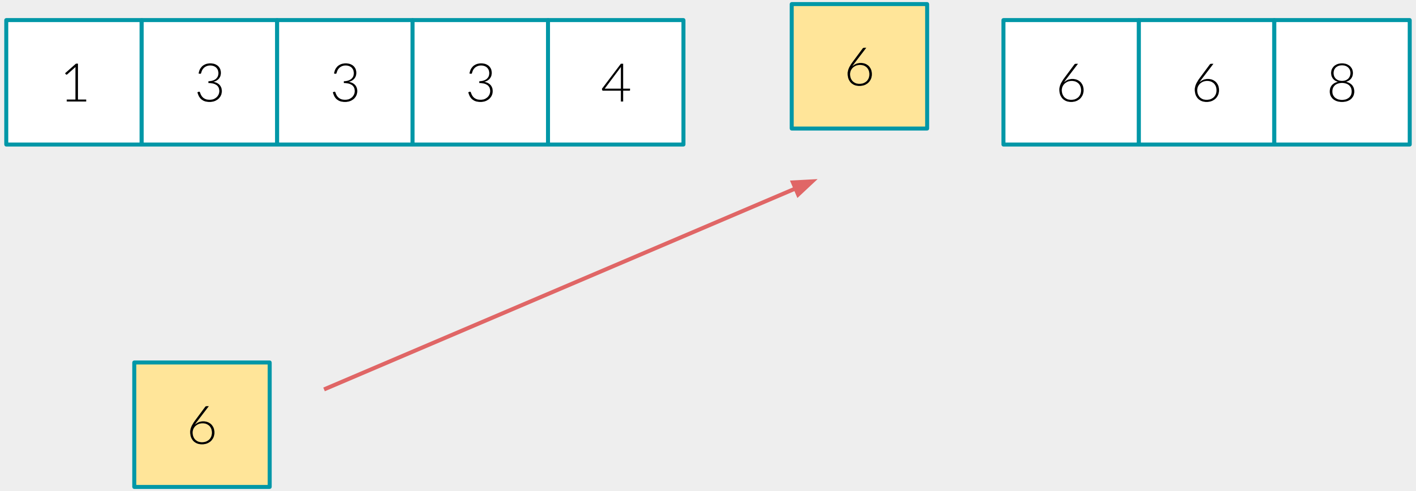
**INPUT**: a **sorted** list of n elements and **a single value**




**OUTPUT**: where is (or where would be) given value in a given list

# The Searching Problem (in a sorted array)

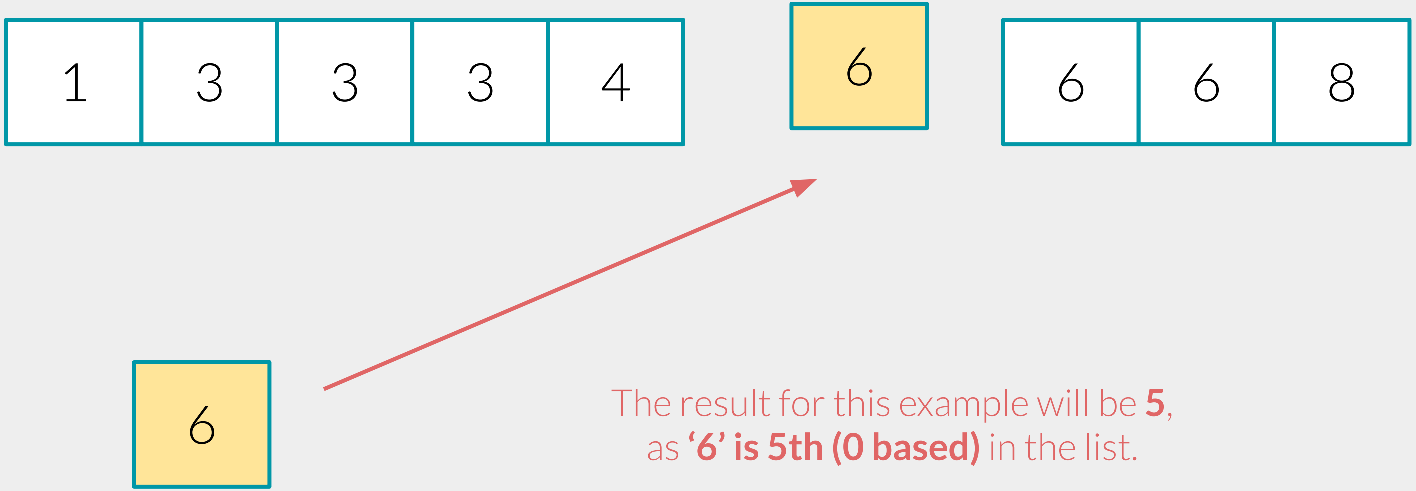
**INPUT**: a **sorted** list of n elements and **a single value**




**OUTPUT**: where is (or where would be) given value in a given list

# The Searching Problem (in a sorted array)

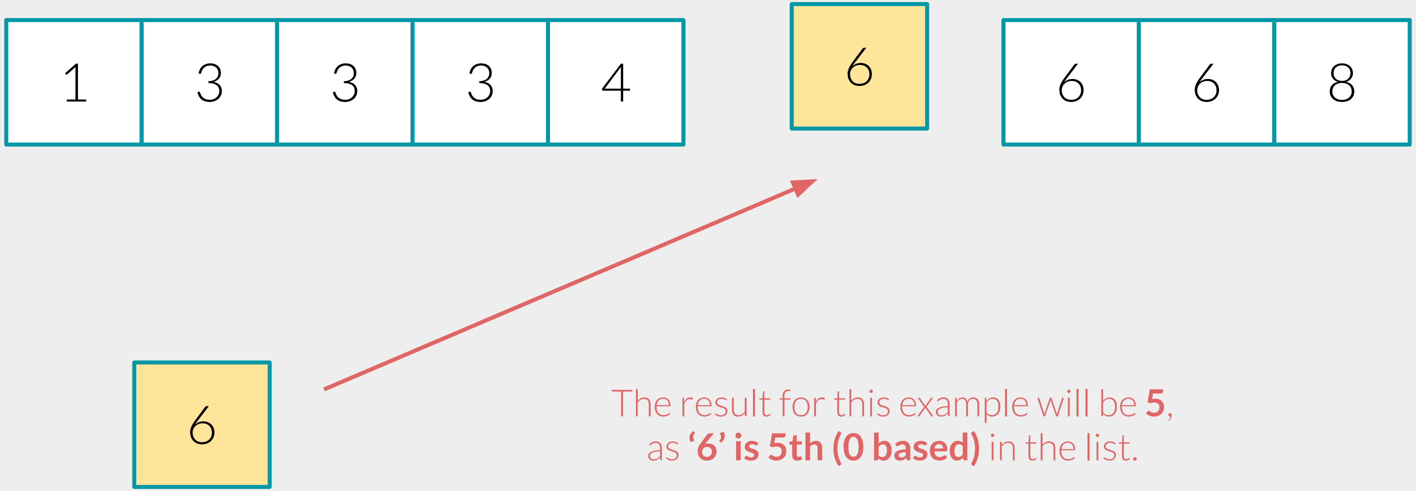
**INPUT**: a **sorted** list of n elements and **a single value**




**OUTPUT**: where is (or where would be) given value in a given list

# The Searching Problem (in a sorted array)

**INPUT**: a **sorted** list of n elements and **a single value**



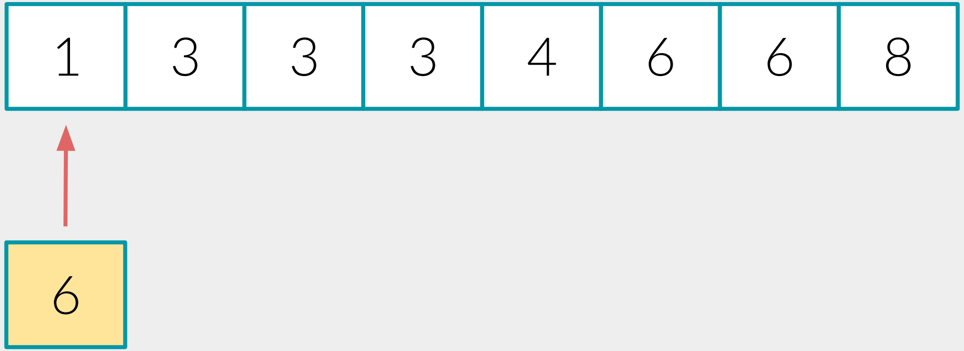

The result for this example will be **5**,
as **‘6’ is 5th (0 based)** in the list.

**OUTPUT**: where is (or where would be) given value in a given list*

* For equal elements take the smallest possible index (leftmost available position)

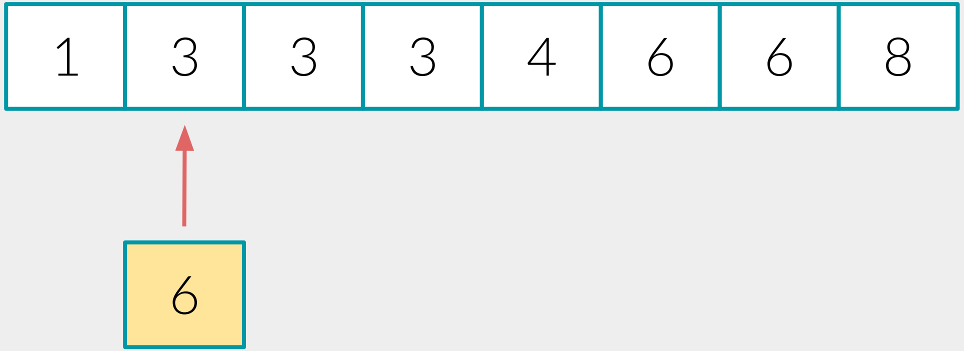
# The Searching Problem: Iterative Solution

```
search(A, x):
  n = length of A
  i = 0
  while i < n and A[i] < x:
    i += 1
  return i
```



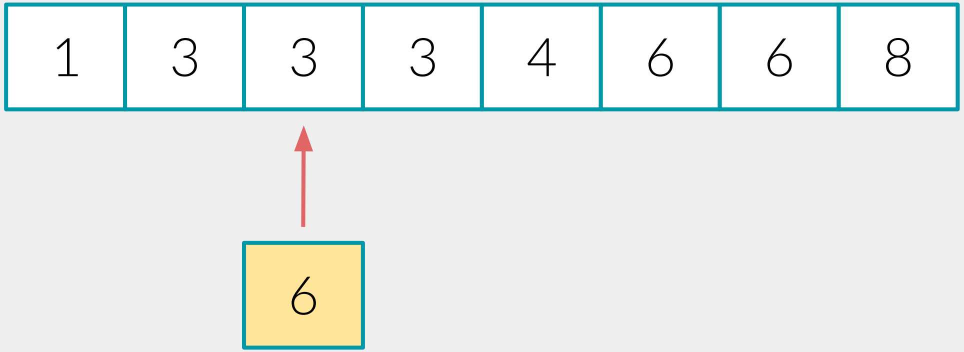
# The Searching Problem: Iterative Solution

```
search(A, x):
  n = length of A
  i = 0
  while i < n and A[i] < x:
    i += 1
  return i
```



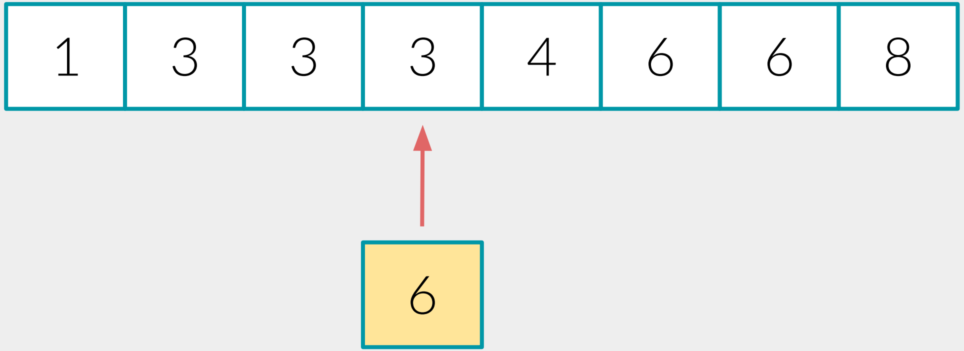
# The Searching Problem: Iterative Solution

```
search(A, x):
  n = length of A
  i = 0
  while i < n and A[i] < x:
    i += 1
  return i
```



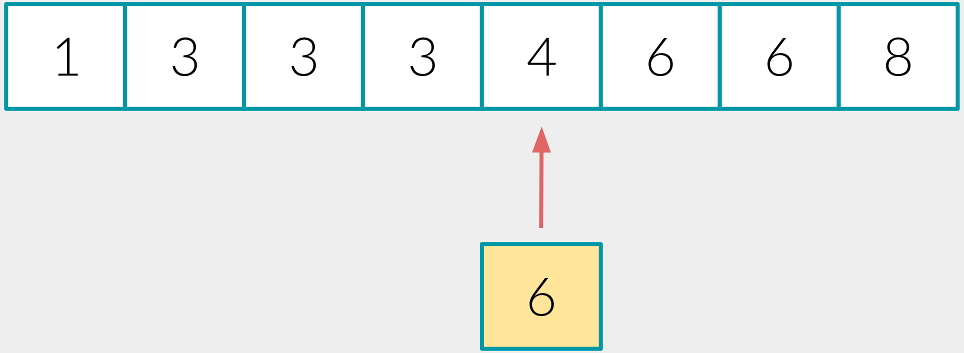
# The Searching Problem: Iterative Solution

```
search(A, x):
  n = length of A
  i = 0
  while i < n and A[i] < x:
    i += 1
  return i
```



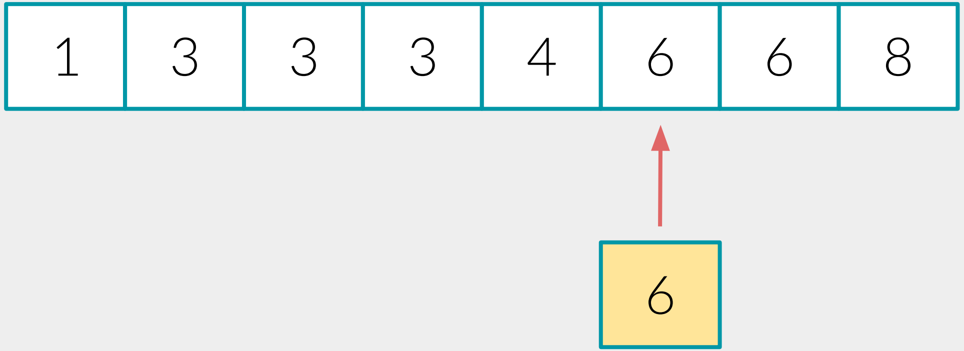
# The Searching Problem: Iterative Solution

```
search(A, x):
  n = length of A
  i = 0
  while i < n and A[i] < x:
    i += 1
  return i
```



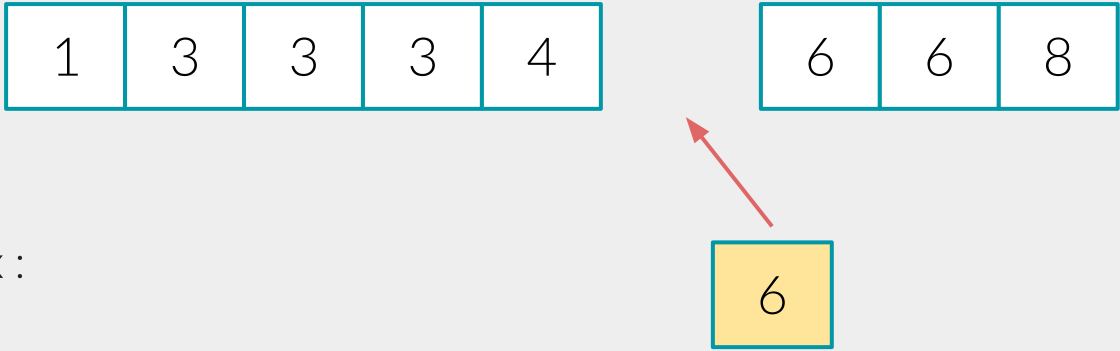
# The Searching Problem: Iterative Solution

```
search(A, x):
  n = length of A
  i = 0
  while i < n and A[i] < x:
    i += 1
  return i
```



# The Searching Problem: Iterative Solution

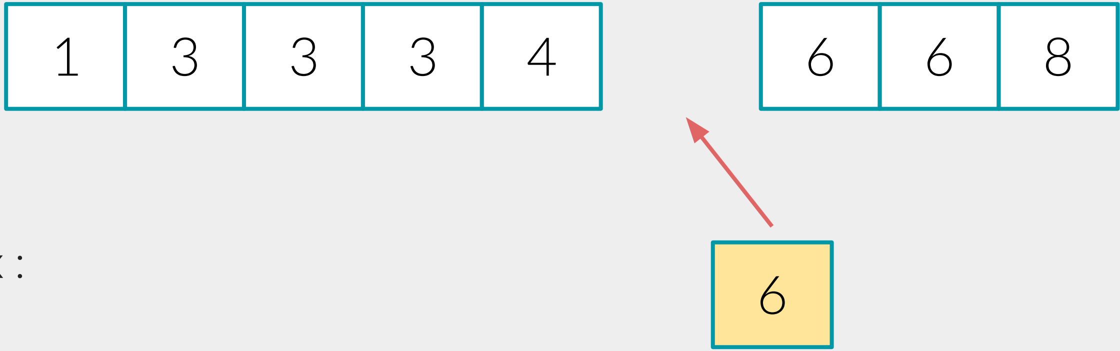
```
search(A, x):
  n = length of A
  i = 0
  while i < n and A[i] < x:
    i += 1
  return i
```




**Complexity**

# The Searching Problem: Iterative Solution

```
search(A, x):
  n = length of A
  i = 0
  while i < n and A[i] < x:
    i += 1
  return i
```




**Complexity**
**O(n)**

# Binary Search

# Binary Search

# Binary Search

```
BinarySearch(A, x):
  if A is empty:
    return 0
  n = length of A
  m = index of the middle element
  if middle element is >= than x:
    return BinarySearch(A[:m], x)
  else:
    return BinarySearch(A[m:], x) + m + 1
```

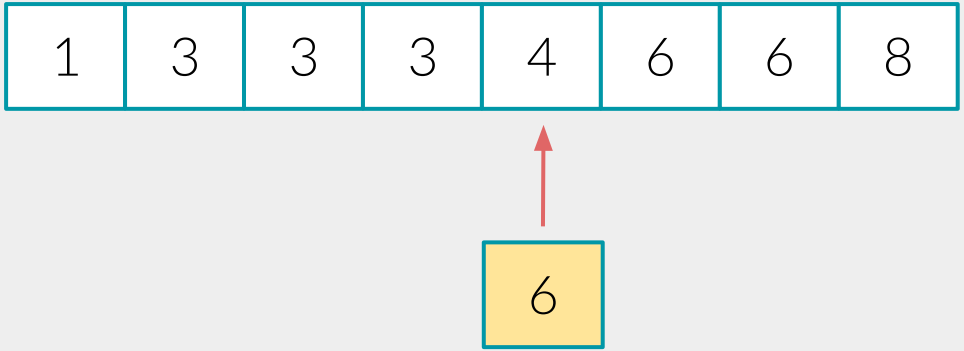
# Binary Search

```
BinarySearch(A, x):
  if A is empty:
    return 0
  n = length of A
  m = index of the middle element
  if middle element is >= than x:
    return BinarySearch(A[:m], x)
  else:
    return BinarySearch(A[m:], x) + m + 1
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 3 | 3 | 4 | 6 | 6 | 8 |

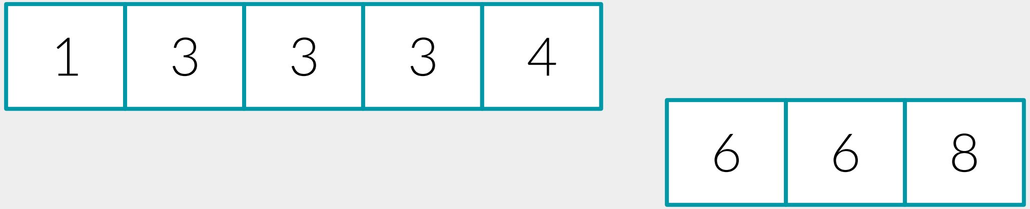
# Binary Search

```
BinarySearch(A, x):
  if A is empty:
    return 0
  n = length of A
  m = index of the middle element
  if middle element is >= than x:
    return BinarySearch(A[:m], x)
  else:
    return BinarySearch(A[m:], x) + m + 1
```



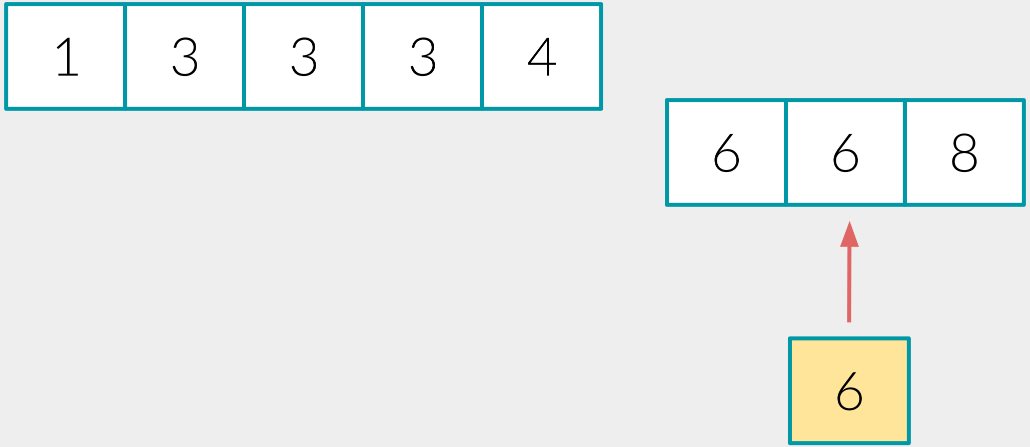
# Binary Search

```
BinarySearch(A, x):
  if A is empty:
    return 0
  n = length of A
  m = index of the middle element
  if middle element is >= than x:
    return BinarySearch(A[:m], x)
  else:
    return BinarySearch(A[m:], x) + m + 1
```



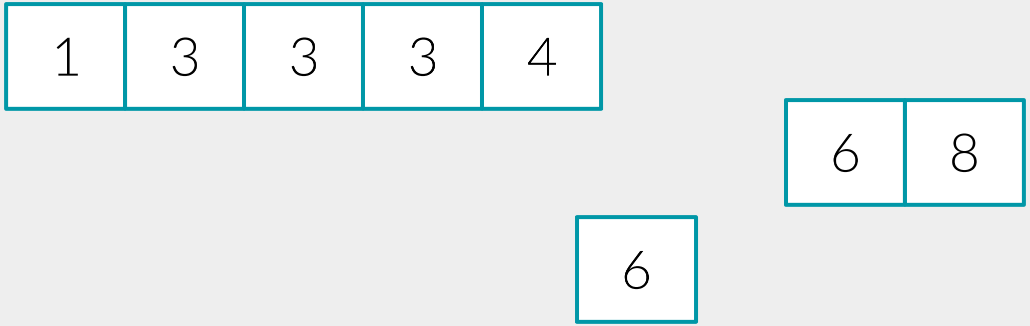
# Binary Search

```
BinarySearch(A, x):
   if A is empty:
      return 0
   n = length of A
   m = index of the middle element
   if middle element is >= than x:
      return BinarySearch(A[:m], x)
   else:
      return BinarySearch(A[m:], x) + m + 1
```



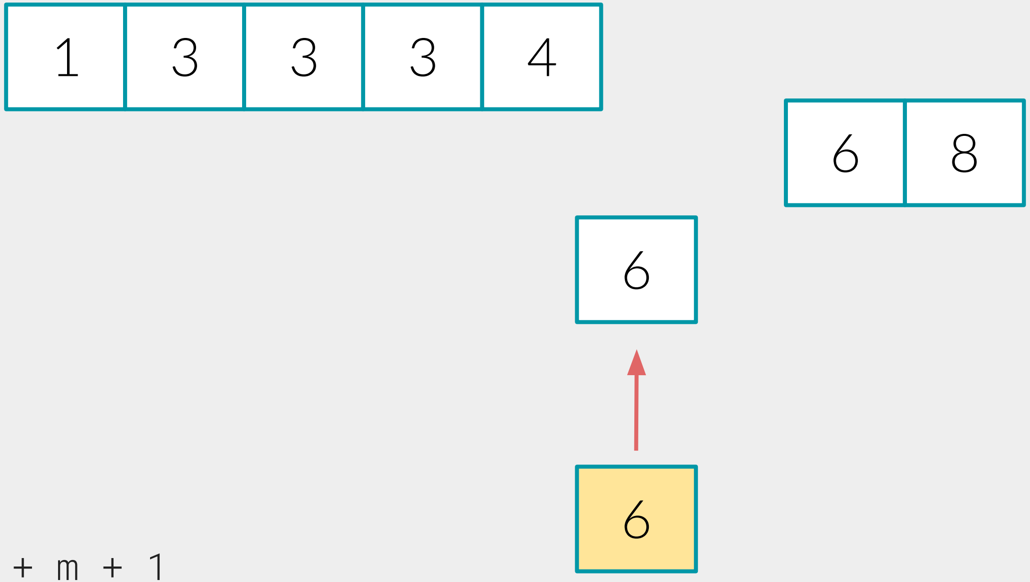
# Binary Search

```
BinarySearch(A, x):
  if A is empty:
    return 0
  n = length of A
  m = index of the middle element
  if middle element is >= than x:
    return BinarySearch(A[:m], x)
  else:
    return BinarySearch(A[m:], x) + m + 1
```



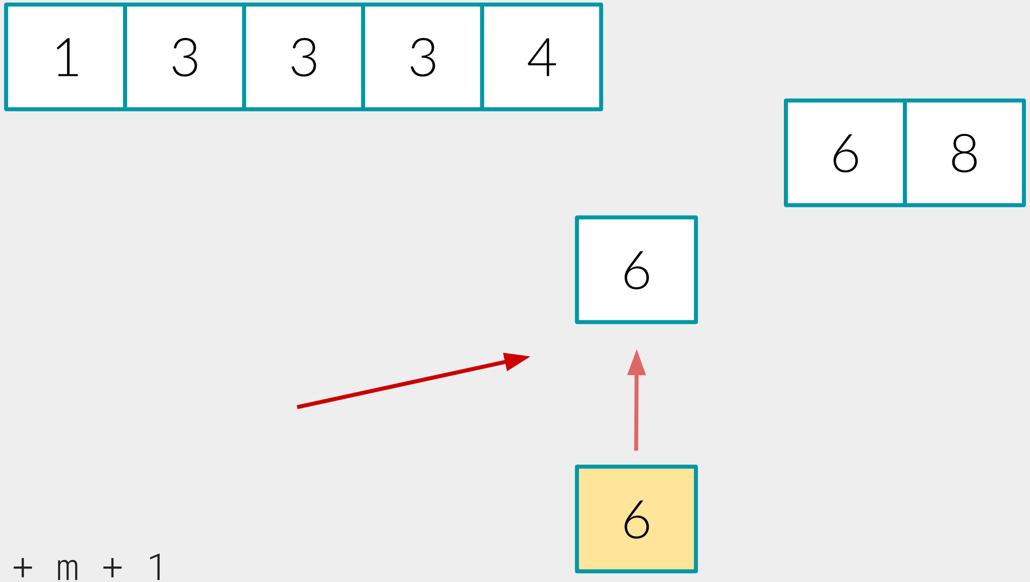
# Binary Search

```
BinarySearch(A, x):
  if A is empty:
    return 0
  n = length of A
  m = index of the middle element
  if middle element is >= than x:
    return BinarySearch(A[:m], x)
  else:
    return BinarySearch(A[m:], x) + m + 1
```



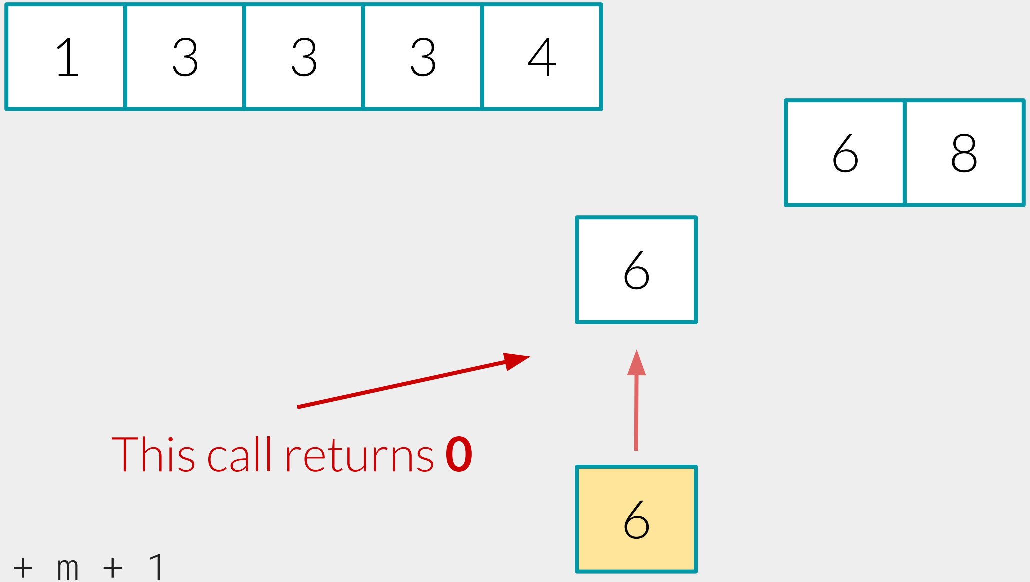
# Binary Search

```
BinarySearch(A, x):
  if A is empty:
    return 0
  n = length of A
  m = index of the middle element
  if middle element is >= than x:
    return BinarySearch(A[:m], x)
  else:
    return BinarySearch(A[m:], x) + m + 1
```



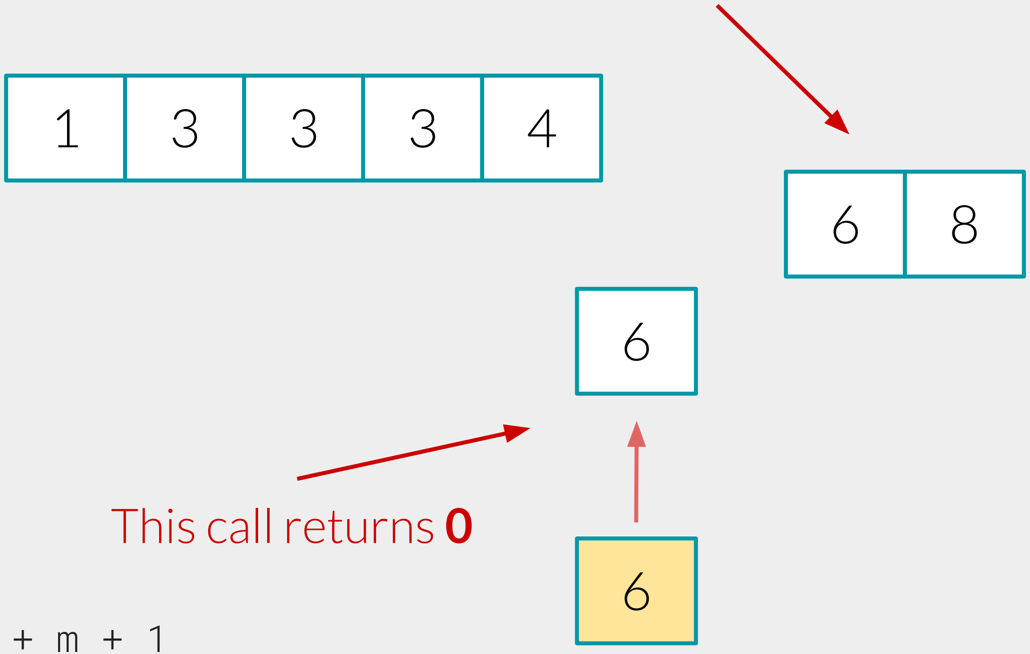
# Binary Search

```
BinarySearch(A, x):
  if A is empty:
    return 0
  n = length of A
  m = index of the middle element
  if middle element is >= than x:
    return BinarySearch(A[:m], x)
  else:
    return BinarySearch(A[m:], x) + m + 1
```



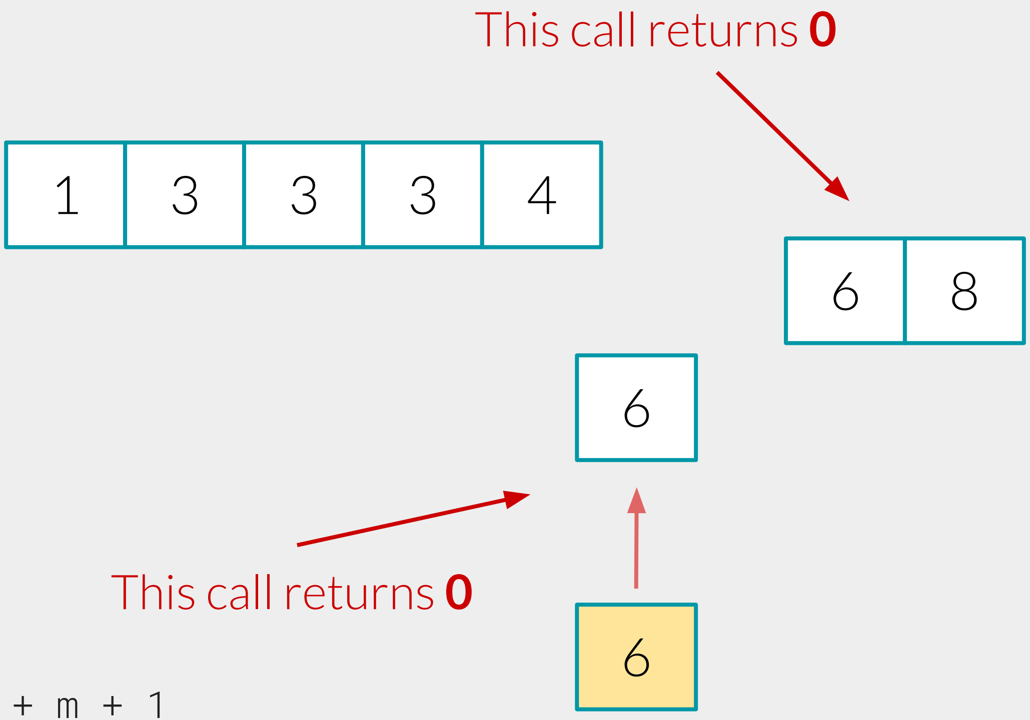
# Binary Search

```
BinarySearch(A, x):
   if A is empty:
      return 0
   n = length of A
   m = index of the middle element
   if middle element is >= than x:
      return BinarySearch(A[:m], x)
   else:
      return BinarySearch(A[m:], x) + m + 1
```



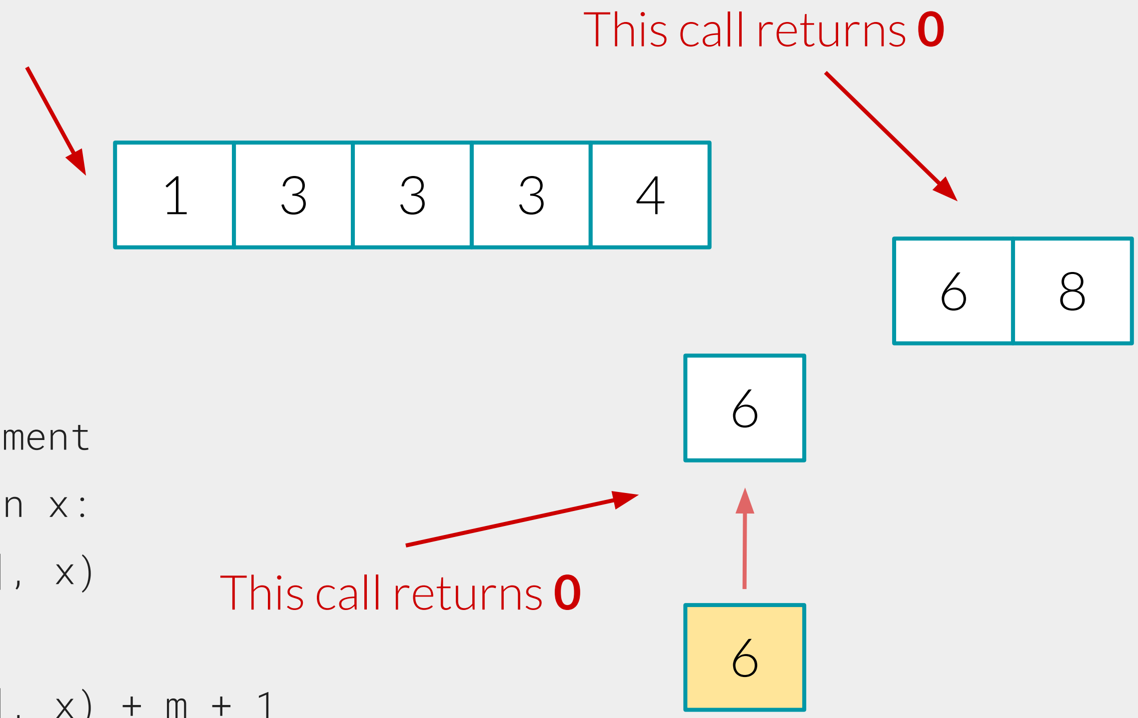
# Binary Search

```
BinarySearch(A, x):
  if A is empty:
    return 0
  n = length of A
  m = index of the middle element
  if middle element is >= than x:
    return BinarySearch(A[:m], x)
  else:
    return BinarySearch(A[m:], x) + m + 1
```



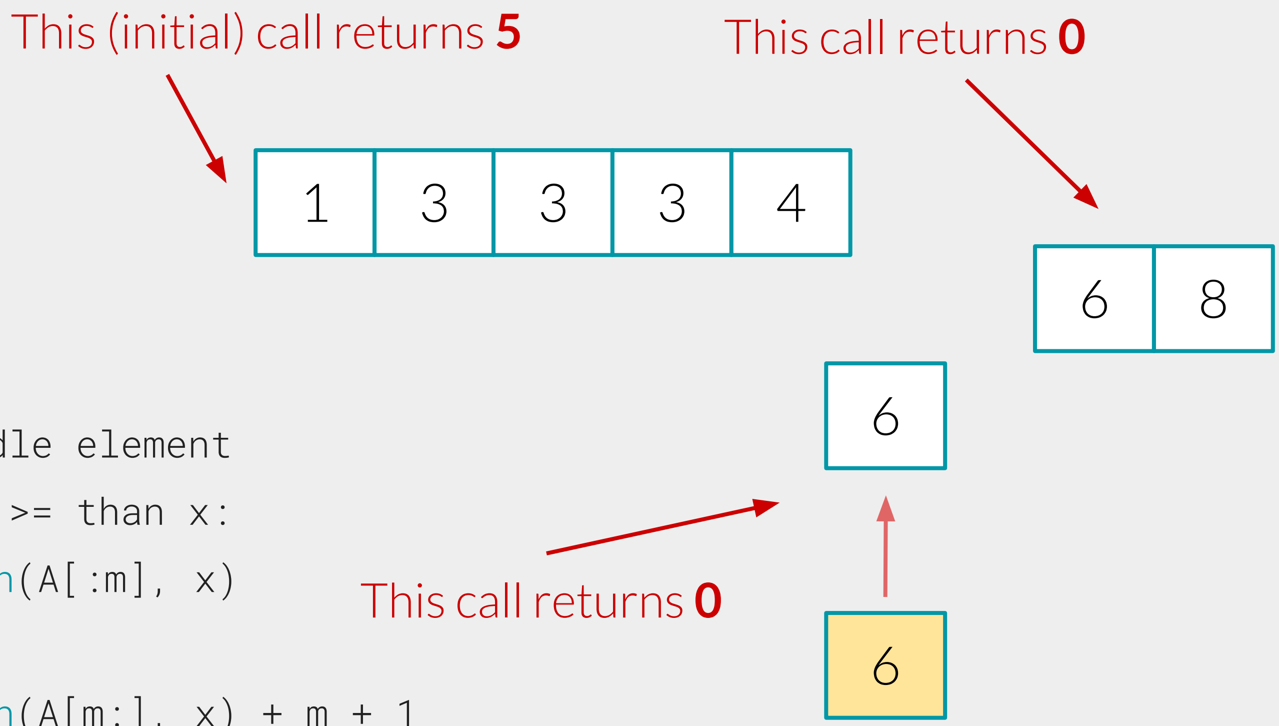
# Binary Search

```
BinarySearch(A, x):
  if A is empty:
    return 0
  n = length of A
  m = index of the middle element
  if middle element is >= than x:
    return BinarySearch(A[:m], x)
  else:
    return BinarySearch(A[m:], x) + m + 1
```



# Binary Search

```
BinarySearch(A, x):
  if A is empty:
    return 0
  n = length of A
  m = index of the middle element
  if middle element is >= than x:
    return BinarySearch(A[:m], x)
  else:
    return BinarySearch(A[m:], x) + m + 1
```



# Binary Search: Complexity

# Substitution Method
# For Solving Recurrences

# MergeSort: Review

```
MergeSort(A):
  if len(A) <= 1:
    return A
  L = MergeSort(A[0:n/2])
  R = MergeSort(A[n/2:n])
  return Merge(L, R)
```

Length n / 2 array

Length n / 2 array

n operations

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) =$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) =$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) = 12$,

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) = 12$, $T(8) = 32$, $T(16) = 80$ …

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) = 12$, $T(8) = 32$, $T(16) = 80 \ldots$

**Our Guess:** $T(n) =$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) = 12$, $T(8) = 32$, $T(16) = 80$ …

**Our Guess:** $T(n) = O(n\log n)$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) = 12$, $T(8) = 32$, $T(16) = 80 \ldots$

**Our Guess:** $T(n) = O(n\log n)$

**Base Case:** For $n = 2$, $T(2) = 4 < C \times 2 \times \log(2) = C \times 2$ for any $C > 2$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) = 12$, $T(8) = 32$, $T(16) = 80$ …

**Our Guess:** $T(n) = O(n\log n)$

**Base Case:** For $n = 2$, $T(2) = 4 < C \times 2 \times \log(2) = C \times 2$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(i \times \log(i)) \Leftrightarrow T(i) < C \times i \times \log(i)$ for some $C$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) = 12$, $T(8) = 32$, $T(16) = 80$ …

**Our Guess:** $T(n) = O(n\log n)$

**Base Case:** For $n = 2$, $T(2) = 4 < C \times 2 \times \log(2) = C \times 2$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(i \times \log(i)) \Leftrightarrow T(i) < C \times i \times \log(i)$ for some $C$

**Inductive Step:** $T(k) = 2 \times T(k / 2) + k$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) = 12$, $T(8) = 32$, $T(16) = 80 \ldots$

**Our Guess:** $T(n) = O(n\log n)$

**Base Case:** For $n = 2$, $T(2) = 4 < C \times 2 \times \log(2) = C \times 2$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(i \times \log(i)) \Leftrightarrow T(i) < C \times i \times \log(i)$ for some $C$

**Inductive Step:** $T(k) = 2 \times T(k / 2) + k$

$< C \times 2 \times (k / 2) \times \log(k/2) + k$ from inductive hypothesis

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) = 12$, $T(8) = 32$, $T(16) = 80 \ldots$

**Our Guess:** $T(n) = O(n\log n)$

**Base Case:** For $n = 2$, $T(2) = 4 < C \times 2 \times \log(2) = C \times 2$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(i \times \log(i)) \Leftrightarrow T(i) < C \times i \times \log(i)$ for some $C$

**Inductive Step:** $T(k) = 2 \times T(k / 2) + k$

$< C \times 2 \times (k / 2) \times \log(k/2) + k$ from inductive hypothesis

$= C \times k \times (\log(k) - 1) + k$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) = 12$, $T(8) = 32$, $T(16) = 80$ …

**Our Guess:** $T(n) = O(n\log n)$

**Base Case:** For $n = 2$, $T(2) = 4 < C \times 2 \times \log(2) = C \times 2$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(i \times \log(i)) \Leftrightarrow T(i) < C \times i \times \log(i)$ for some $C$

**Inductive Step:** $T(k) = 2 \times T(k / 2) + k$

$< C \times 2 \times (k / 2) \times \log(k/2) + k$ from inductive hypothesis

$= C \times k \times (\log(k) - 1) + k$

$= C \times k \times \log(k) - C \times k + k$

# Substitution Method: MergeSort Example

**Recurrence relation**: $T(n) = 2 \times T(n/2) + n$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = 2 \times T(1) + 2 = 4$, $T(4) = 12$, $T(8) = 32$, $T(16) = 80 \ldots$

**Our Guess:** $T(n) = O(n\log n)$

**Base Case:** For $n = 2$, $T(2) = 4 < C \times 2 \times \log(2) = C \times 2$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(i \times \log(i)) \Leftrightarrow T(i) < C \times i \times \log(i)$ for some $C$

**Inductive Step:** $T(k) = 2 \times T(k / 2) + k$

$< C \times 2 \times (k / 2) \times \log(k/2) + k$ from inductive hypothesis

$= C \times k \times (\log(k) - 1) + k$

$= C \times k \times \log(k) - C \times k + k$

$< C \times k \times \log(k)$ as $C > 2 \Rightarrow C \times (k/2) > k$

# Substitution Method: MergeSort Example

**Recurrence relation**: T(n) = 2 x T(n/2) + n, T(1) = 1

**First Observations:** T(1) = 1, T(2) = 2 x T(1) + 2 = 4, T(4) = 12, T(8) = 32, T(16) = 80 ...

**Our Guess:** T(n) = O(nlogn)

**Base Case:** For n = 2, T(2) = 4 < C x 2 x log(2) = C x 2 for any C > 2

**Inductive Hypothesis:** For all i < k, T(i) = O(i x log(i)) ⇔ T(i) < C x i x log(i) for some C

**Inductive Step:** T(k) = 2 x T(k / 2) + k

< C x 2 x (k / 2) x log(k/2) + k      from inductive hypothesis

= C x k x (log(k) - 1) + k

= C x k x log(k) - C x k + k

< C x k x log(k)      as C > 2 => C x (k/2) > k

= **O( k x log(k) )**

# Substitution Method: MergeSort Example

**Recurrence relation**: T(n) = 2 x T(n/2) + n, T(1) = 1

**First Observations:** T(1) = 1, T(2) = 2 x T(1) + 2 = 4, T(4) = 12, T(8) = 32, T(16) = 80 …

**Our Guess:** T(n) = O(nlogn)

**Base Case:** For n = 2, T(2) = 4 < C x 2 x log(2) = C x 2 for any C > 2

**Inductive Hypothesis:** For all i < k, T(i) = O(i x log(i)) ⇔ T(i) < C x i x log(i) for some C

**Inductive Step:** T(k) = 2 x T(k / 2) + k

< C x 2 x (k / 2) x log(k/2) + k — from inductive hypothesis

= C x k x (log(k) - 1) + k

= C x k x log(k) - C x k + k

< C x k x log(k) — as C > 2 => C x (k/2) > k

= **O( k x log(k) )**

So we conclude the proof. We proved that **T(n) < C x n x log(n)** for **C = 3** and for all **n > 1.**

# Binary Search: Complexity

**Recurrence relation:** $T(n) =$

# Binary Search: Complexity

**Recurrence relation:** $T(n) = T(n/2) + 1$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:**

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) =$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$, $T(4) = 3$, $T(8) = 4$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$, $T(4) = 3$, $T(8) = 4$, $T(16) = 5$, $T(32) = 6 \ldots$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$, $T(4) = 3$, $T(8) = 4$, $T(16) = 5$, $T(32) = 6$ …

**Our Guess:**

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$, $T(4) = 3$, $T(8) = 4$, $T(16) = 5$, $T(32) = 6 \ldots$

**Our Guess:** $T(n) = O(\log n)$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$, $T(4) = 3$, $T(8) = 4$, $T(16) = 5$, $T(32) = 6 \ldots$

**Our Guess:** $T(n) = O(\log n)$

**Base Case:** For $n = 2$, $T(2) = 2 < C \times \log(2) = C$ for any $C > 2$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$, $T(4) = 3$, $T(8) = 4$, $T(16) = 5$, $T(32) = 6$ …

**Our Guess:** $T(n) = O(\log n)$

**Base Case:** For $n = 2$, $T(2) = 2 < C \times \log(2) = C$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(\log i) \Leftrightarrow T(i) < C \times \log(i)$ for some $C$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$, $T(4) = 3$, $T(8) = 4$, $T(16) = 5$, $T(32) = 6$ …

**Our Guess:** $T(n) = O(\log n)$

**Base Case:** For $n = 2$, $T(2) = 2 < C \times \log(2) = C$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(\log i) \Leftrightarrow T(i) < C \times \log(i)$ for some $C$

**Inductive Step:** $T(k) = T(k / 2) + 1$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$, $T(4) = 3$, $T(8) = 4$, $T(16) = 5$, $T(32) = 6$ …

**Our Guess:** $T(n) = O(\log n)$

**Base Case:** For $n = 2$, $T(2) = 2 < C \times \log(2) = C$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(\log i) \Leftrightarrow T(i) < C \times \log(i)$ for some $C$

**Inductive Step:** $T(k) = T(k / 2) + 1$

$< C \times \log(k/2) + 1$ from inductive hypothesis

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$, $T(4) = 3$, $T(8) = 4$, $T(16) = 5$, $T(32) = 6 \ldots$

**Our Guess:** $T(n) = O(\log n)$

**Base Case:** For $n = 2$, $T(2) = 2 < C \times \log(2) = C$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(\log i) \Leftrightarrow T(i) < C \times \log(i)$ for some $C$

**Inductive Step:**
$$\begin{aligned} T(k) &= T(k / 2) + 1 \\ &< C \times \log(k/2) + 1 && \text{from inductive hypothesis} \\ &= C \times (\log(k) - 1) + 1 \end{aligned}$$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1, T(1) = 1$

**First Observations:** $T(1) = 1, T(2) = T(1) + 1 = 2, T(4) = 3, T(8) = 4, T(16) = 5, T(32) = 6 \ldots$

**Our Guess:** $T(n) = O(\log n)$

**Base Case:** For $n = 2$, $T(2) = 2 < C \times \log(2) = C$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(\log i) \Leftrightarrow T(i) < C \times \log(i)$ for some $C$

**Inductive Step:** $T(k) = T(k / 2) + 1$

$< C \times \log(k/2) + 1$ from inductive hypothesis

$= C \times (\log(k) - 1) + 1$

$= C \times \log(k) - C + 1$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1, T(1) = 1$

**First Observations:** $T(1) = 1, T(2) = T(1) + 1 = 2, T(4) = 3, T(8) = 4, T(16) = 5, T(32) = 6 \ldots$

**Our Guess:** $T(n) = O(\log n)$

**Base Case:** For $n = 2$, $T(2) = 2 < C \times \log(2) = C$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(\log i) \Leftrightarrow T(i) < C \times \log(i)$ for some $C$

**Inductive Step:** $T(k) = T(k / 2) + 1$

$$
\begin{aligned}
&< C \times \log(k/2) + 1 && \text{from inductive hypothesis} \\
&= C \times (\log(k) - 1) + 1 && \\
&= C \times \log(k) - C + 1 && \\
&< C \times \log(k) && \text{as } C > 2
\end{aligned}
$$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1, T(1) = 1$

**First Observations:** $T(1) = 1, T(2) = T(1) + 1 = 2, T(4) = 3, T(8) = 4, T(16) = 5, T(32) = 6 \ldots$

**Our Guess:** $T(n) = O(\log n)$

**Base Case:** For $n = 2$, $T(2) = 2 < C \times \log(2) = C$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(\log i) \Leftrightarrow T(i) < C \times \log(i)$ for some $C$

**Inductive Step:** $T(k) = T(k / 2) + 1$

$$
\begin{aligned}
&< C \times \log(k/2) + 1 && \text{from inductive hypothesis} \\
&= C \times (\log(k) - 1) + 1 \\
&= C \times \log(k) - C + 1 \\
&< C \times \log(k) && \text{as } C > 2 \\
&= O(\log(k))
\end{aligned}
$$

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$, $T(4) = 3$, $T(8) = 4$, $T(16) = 5$, $T(32) = 6$ …

**Our Guess:** $T(n) = O(\log n)$

**Base Case:** For $n = 2$, $T(2) = 2 < C \times \log(2) = C$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(\log i) \Leftrightarrow T(i) < C \times \log(i)$ for some $C$

**Inductive Step:**
$$
\begin{aligned}
T(k) &= T(k / 2) + 1 \\
&< C \times \log(k/2) + 1 && \text{from inductive hypothesis} \\
&= C \times (\log(k) - 1) + 1 \\
&= C \times \log(k) - C + 1 \\
&< C \times \log(k) && \text{as } C > 2 \\
&= O(\log(k))
\end{aligned}
$$

So we conclude the proof. We proved that **$T(n) < C \times \log(n)$** for **$C = 3$** and for all **$n > 1$.**

# Substitution Method

**Recurrence relation**: $T(n) = T(n/2) + 1$, $T(1) = 1$

**First Observations:** $T(1) = 1$, $T(2) = T(1) + 1 = 2$, $T(4) = 3$, $T(8) = 4$, $T(16) = 5$, $T(32) = 6$ …

**Our Guess:** $T(n) = O(\log n)$

**Base Case:** For $n = 2$, $T(2) = 2 < C \times \log(2) = C$ for any $C > 2$

**Inductive Hypothesis:** For all $i < k$, $T(i) = O(\log i) \Leftrightarrow T(i) < C \times \log(i)$ for some $C$

**Inductive Step:**

$$
\begin{aligned}
T(k) &= T(k / 2) + 1 \\
&< C \times \log(k/2) + 1 && \text{from inductive hypothesis} \\
&= C \times (\log(k) - 1) + 1 \\
&= C \times \log(k) - C + 1 \\
&< C \times \log(k) && \text{as } C > 2 \\
&= O(\log(k))
\end{aligned}
$$

So we conclude the proof. We proved that **$T(n) < C \times \log(n)$** for **$C = 3$** and for all **$n > 1$.**
Which means that **$T(n) = O(\log n)$ !**

# Binary Search: Complexity

**Recurrence relation:** $T(n) = T(n/2) + 1$

# Binary Search: Complexity

**Recurrence relation:** $T(n) = T(n/2) + 1$

**Complexity:** $O(\log n)$

# Binary Search: Use Examples

Find number of occurrences of a value in a sorted array.

# Binary Search: Use Examples

Find number of occurrences of a value in a sorted array.

Binary Search Trees!

# Binary Search: Use Examples

Find number of occurrences of a value in a sorted array.

Binary Search Trees!

Searching for a word in a dictionary (a real one).

# Binary Search: Use Examples

Find number of occurrences of a value in a sorted array.

Binary Search Trees!

Searching for a word in a dictionary (a real one).

Literally everywhere.