Recursion + Binary Search

Kelly Rivers and Stephanie Rosenthal

15-110 Fall 2019

Announcements

- Homework 3 Check-in 2
  - How did it go?
- Homework 3 full is due next week!

Learning Objectives

- To understand the purpose and definition of recursion
- To produce recursive algorithms
- To understand and trace a recursive binary search algorithm

Recursion

Recursion is a way to compute by

1) breaking down the problem into smaller pieces
2) calling itself (the same function) on each smaller piece
3) combining the smaller answers back together in some way

Picture of What is Happening

Example:

```
def recursiveAdd(L):
    if L == []:
        return 0
    return L[0]+recursiveAdd(L[1:])
```

Picture of What is Happening

Example:

```
def recursiveAdd(L):
    if L == []:
        return 0
    return L[0]+recursiveAdd(L[1:])
```

recursiveAdd(

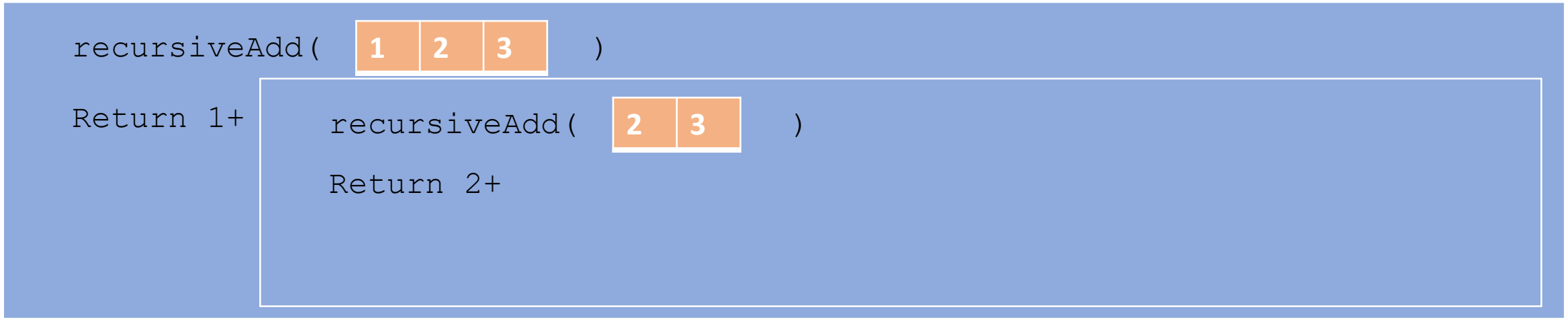
)

Return 1+

Picture of What is Happening

Example:

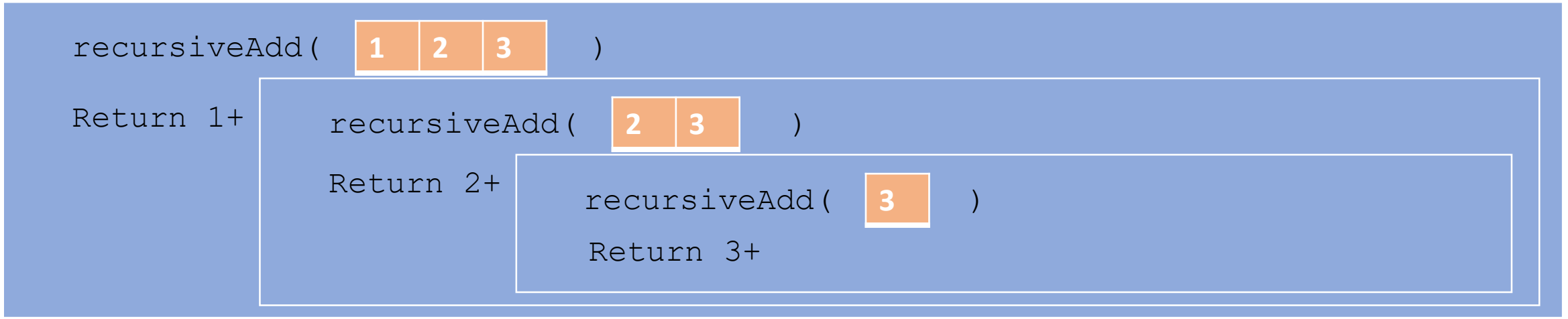
```
def recursiveAdd(L):
    if L == []:
        return 0
    return L[0]+recursiveAdd(L[1:])
```



Picture of What is Happening

Example:

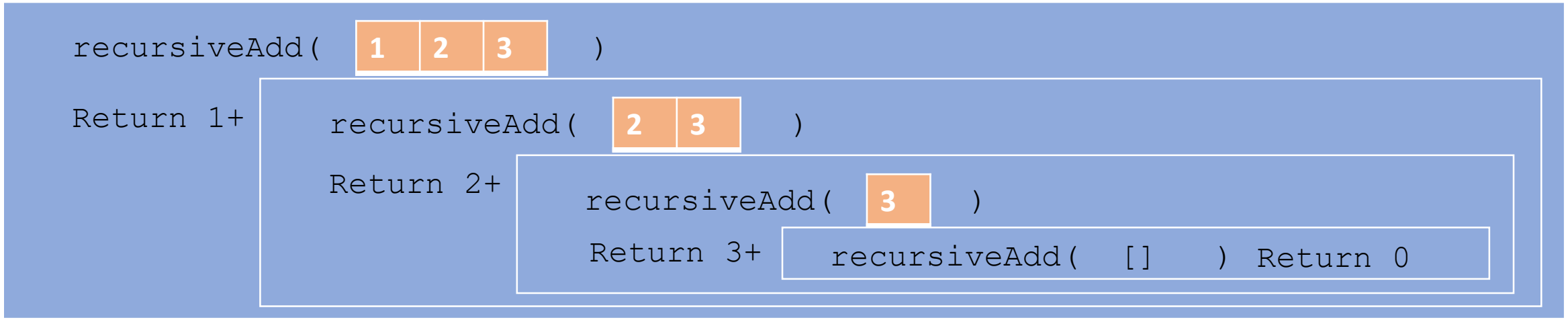
```
def recursiveAdd(L):
    if L == []:
        return 0
    return L[0]+recursiveAdd(L[1:])
```



Picture of What is Happening

Example:

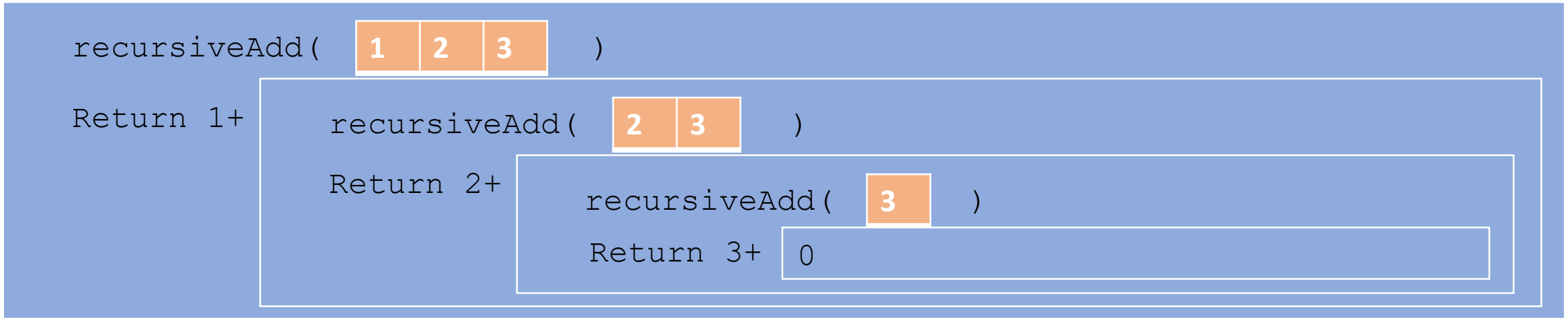
```
def recursiveAdd(L):
    if L == []:
        return 0
    return L[0]+recursiveAdd(L[1:])
```



Picture of What is Happening

Example:

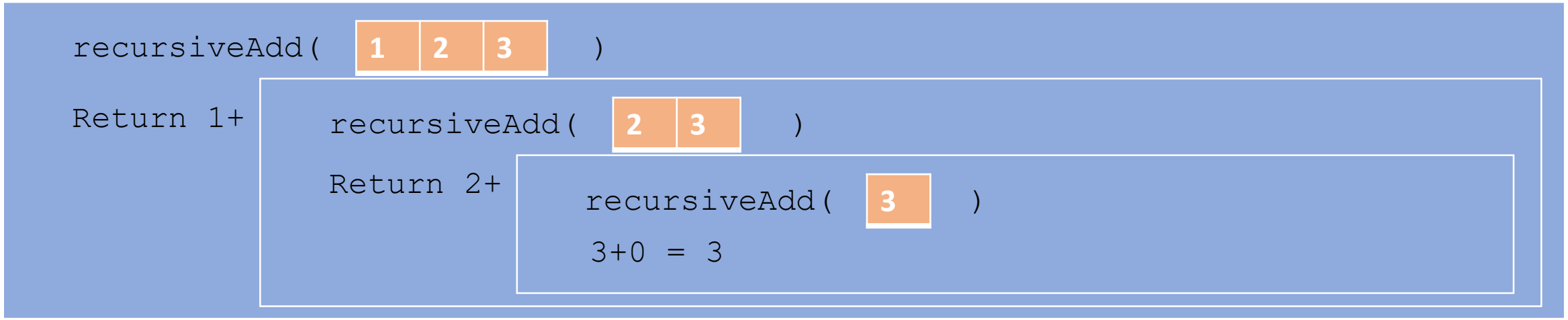
```
def recursiveAdd(L):
    if L == []:
        return 0
    return L[0]+recursiveAdd(L[1:])
```



Picture of What is Happening

Example:

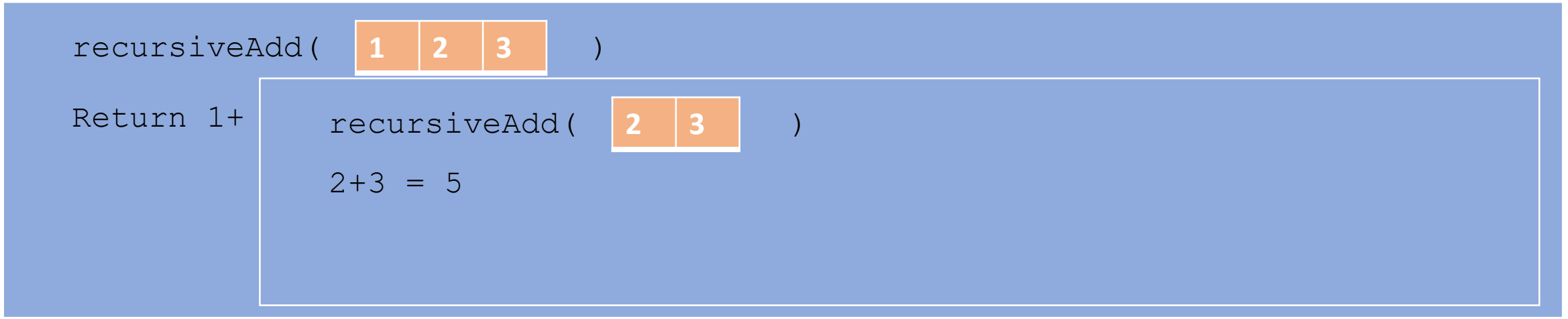
```
def recursiveAdd(L):
    if L == []:
        return 0
    return L[0]+recursiveAdd(L[1:])
```



Picture of What is Happening

Example:

```
def recursiveAdd(L):
    if L == []:
        return 0
    return L[0]+recursiveAdd(L[1:])
```



Picture of What is Happening

Example:

```
def recursiveAdd(L):
    if L == []:
        return 0
    return L[0]+recursiveAdd(L[1:])
```

recursiveAdd(

)

Return 1+5 = 6

Picture of What is Happening

Example:

```
def recursiveAdd(L):
    if L == []:
        return 0
    return L[0]+recursiveAdd(L[1:])
```

recursiveAdd(

)

6

Recursion

General Form:

```
def recursiveFunction(X):
    if X == ?: #base case is some smallest case
        return ____ #something not recursive
    else:
        #call itself on a smaller piece
        result = recursiveFunction(Xsmaller)
        #combine with some value and return
        return _______
```

Activity: Search Elements in List

```
def recursiveSearch(L,item):
```

Activity: Search Elements in List

```
def recursiveSearch(L,item):
```

Base cases:

L = [], what should be returned?

L[0] == item, what should be returned?

Inductive case?

Activity: Search Elements in List

```
def recursiveSearch(L,item):
    if len(L) == 0: #base case is some smallest case
        return False
    if L[0] == item: #base case is some smallest case
        return True
    else:
        #call itself on a smaller piece
        result = recursiveSearch(L[1:],item)
        return result
```

Trace the function

```
def recursiveSearch(L,item):
    if len(L) == 0: #base case is some smallest case
        return False
    if L[0] == item: #base case is some smallest case
        return True
    else:
        #call itself on a smaller piece
        result = recursiveSearch(L[1:],item)
        return result


recursiveSearch(["dog", "cat", "mouse"], "mouse")
```

Trace the function

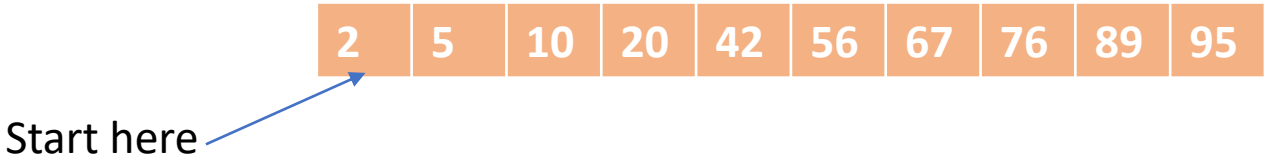
```
def recursiveSearch(L,item):
    if len(L) == 0: #base case is some smallest case
        return False
    if L[0] == item: #base case is some smallest case
        return True
    else:
        #call itself on a smaller piece
        result = recursiveSearch(L[1:],item)
        return result


recursiveSearch([0,10,30,60,200,500],65)
```

Searching Sorted Lists

Linear Search: searching in order from beginning to end

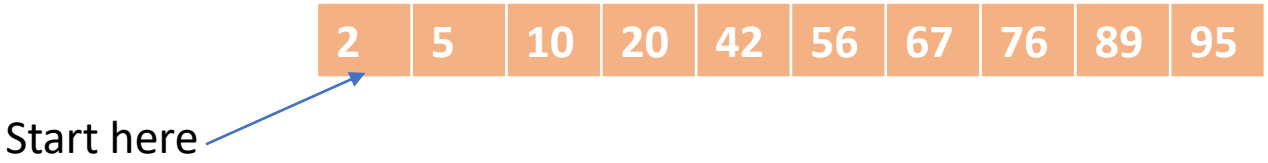
I need 10 comparisons to check for 98



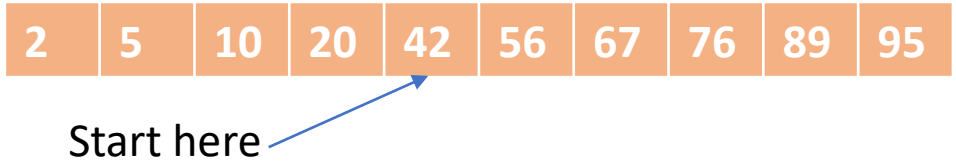
Searching Sorted Lists

Linear Search: searching in order from beginning to end

I need 10 comparisons to check for 98

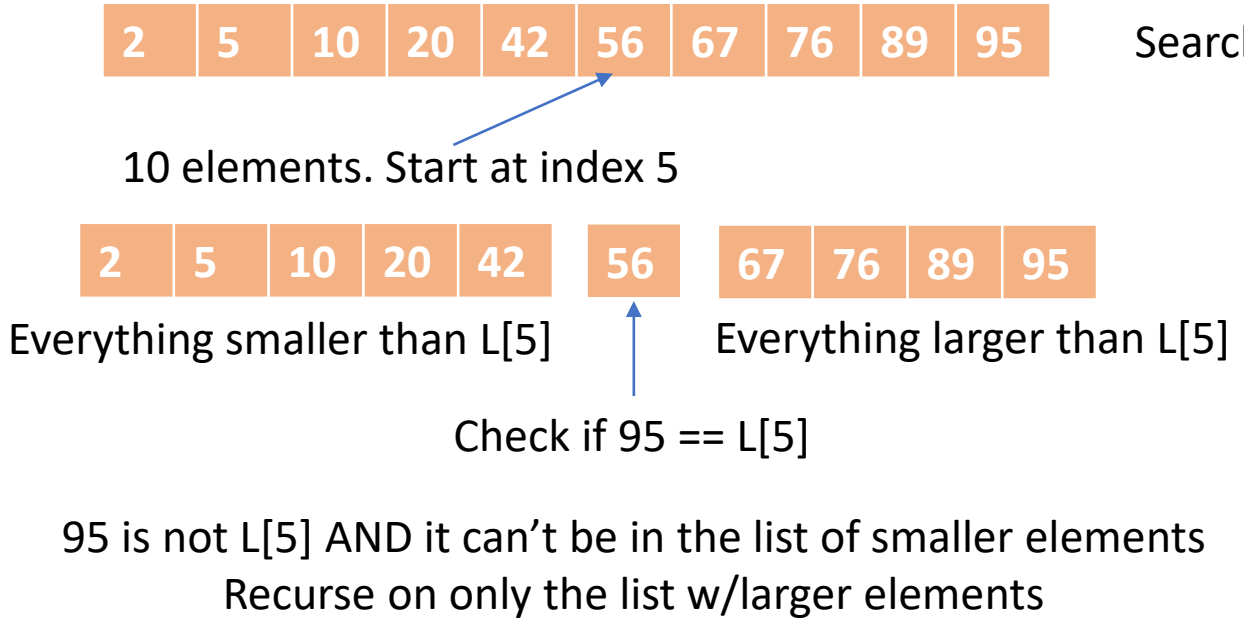


Different Idea: I can eliminate half of the list with 1 comparison if I check the middle element instead of the first element



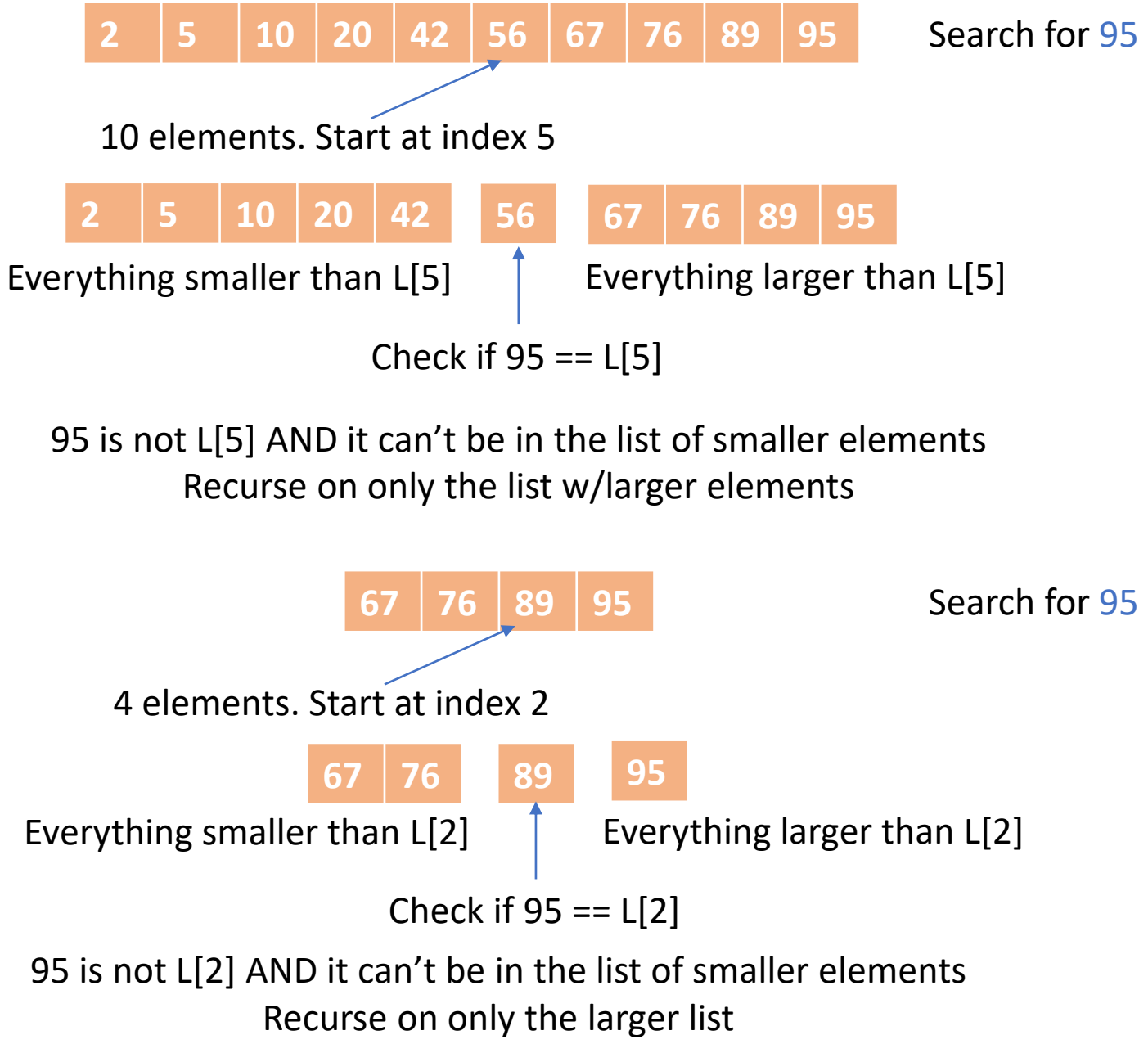
Binary Search

For a sorted list, compare to the middle element and recurse on the side that the element is on



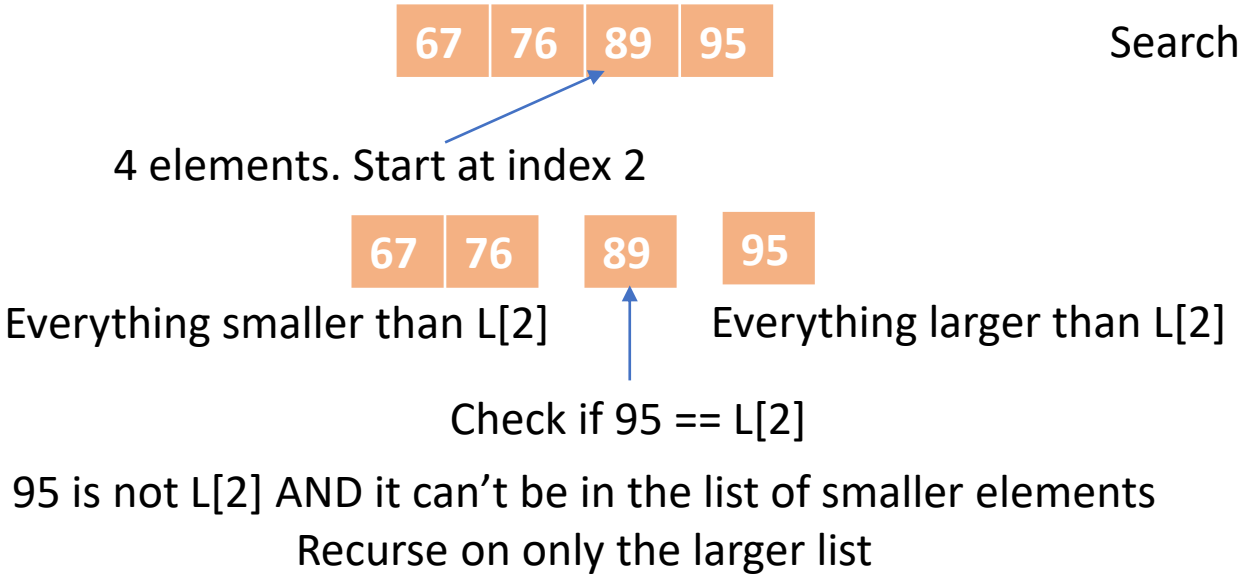
Binary Search

For a sorted list, compare to the middle element and recurse on the side that the element is on



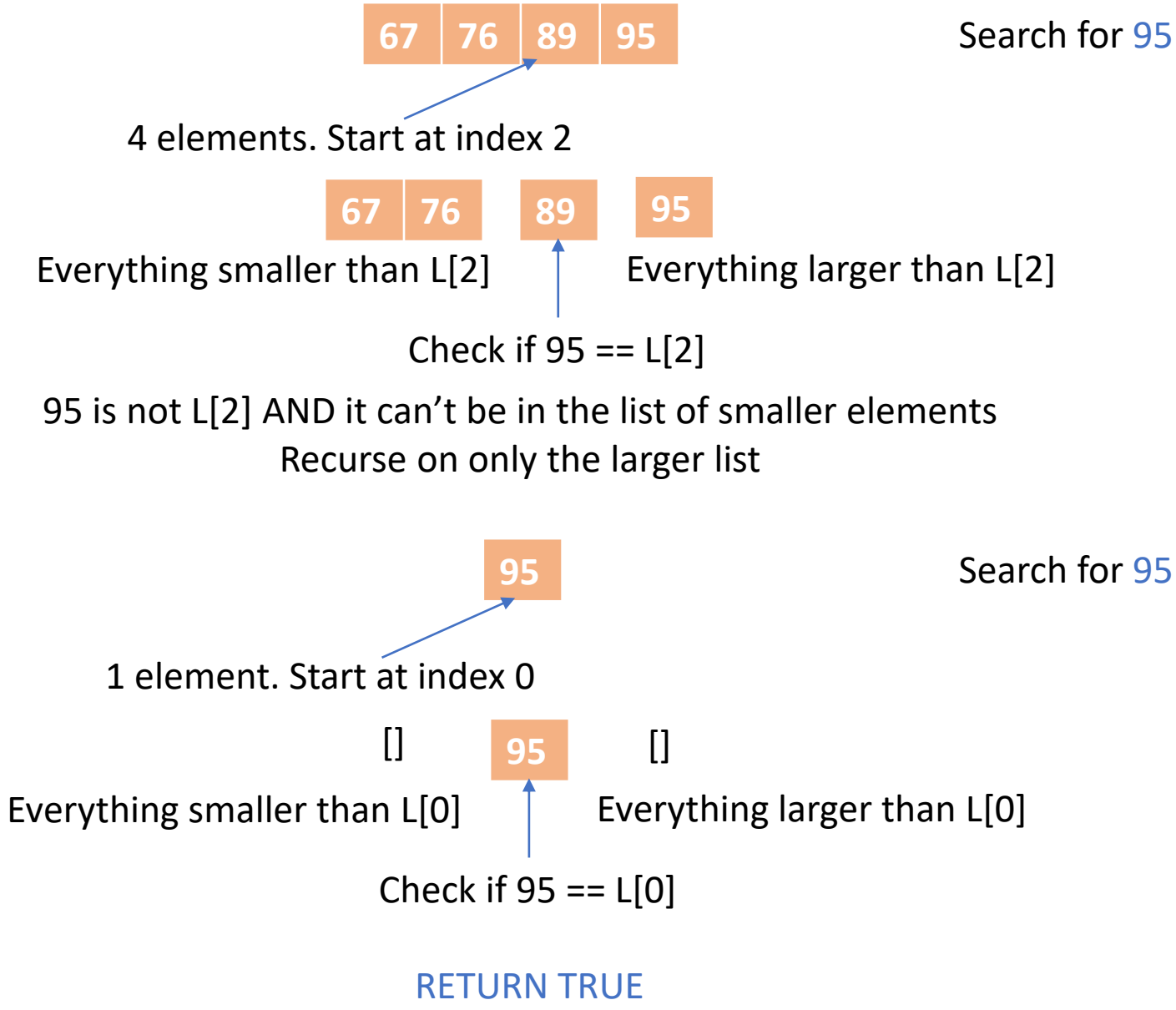
Binary Search

For a sorted list, compare to the middle element and recurse on the side that the element is on



Binary Search

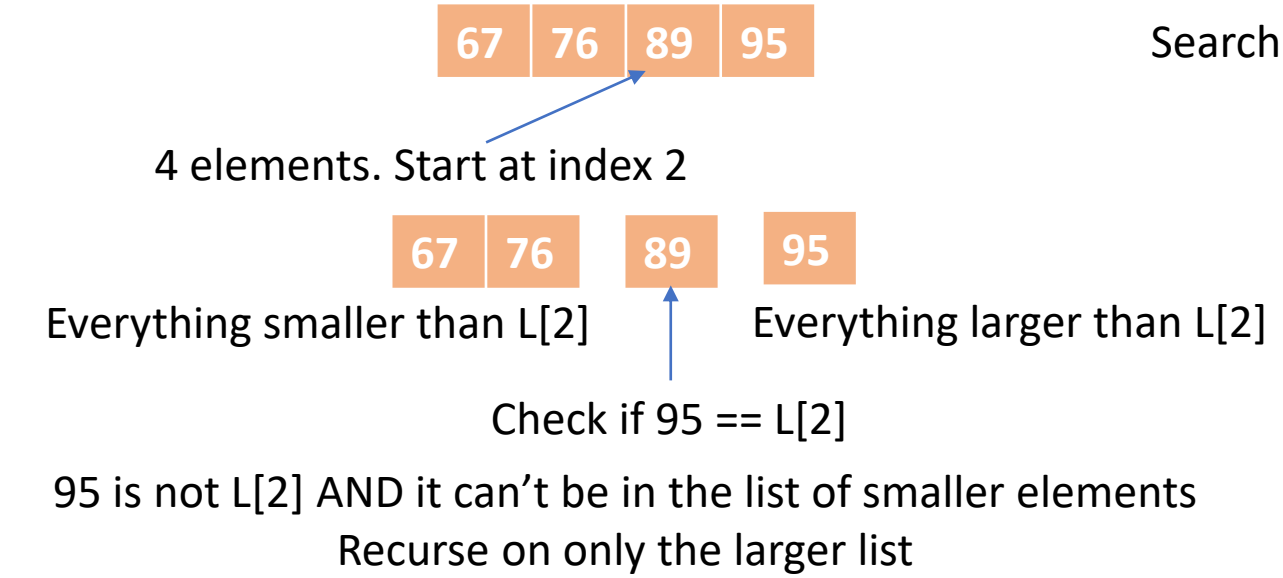
For a sorted list, compare to the middle element and recurse on the side that the element is on

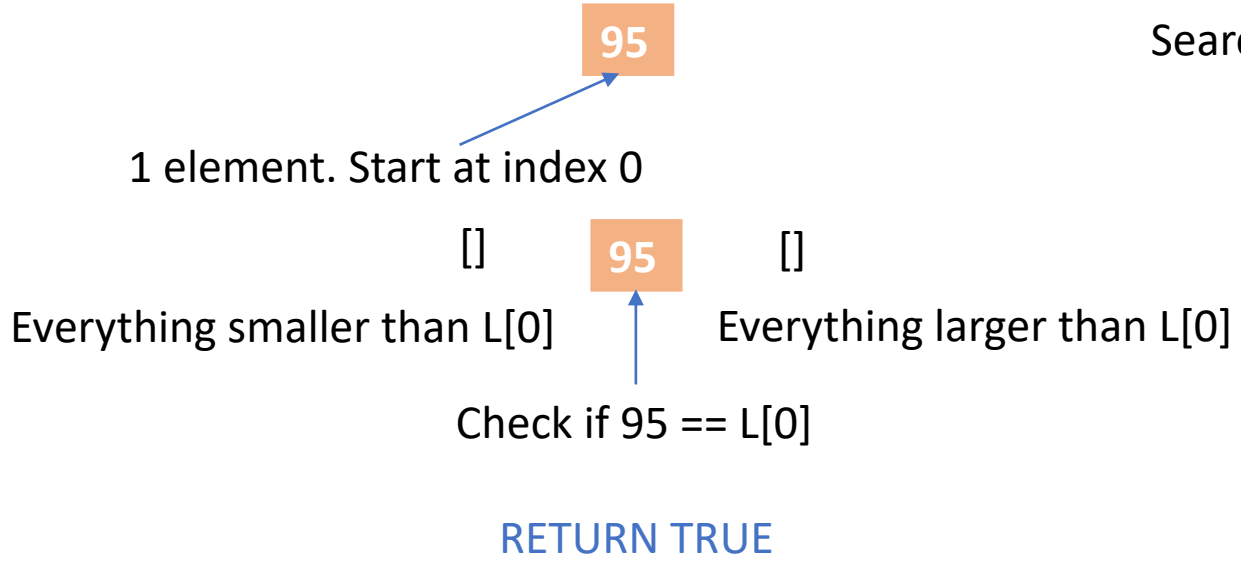


Binary Search

For a sorted list, compare to the middle element and recurse on the side that the element is on

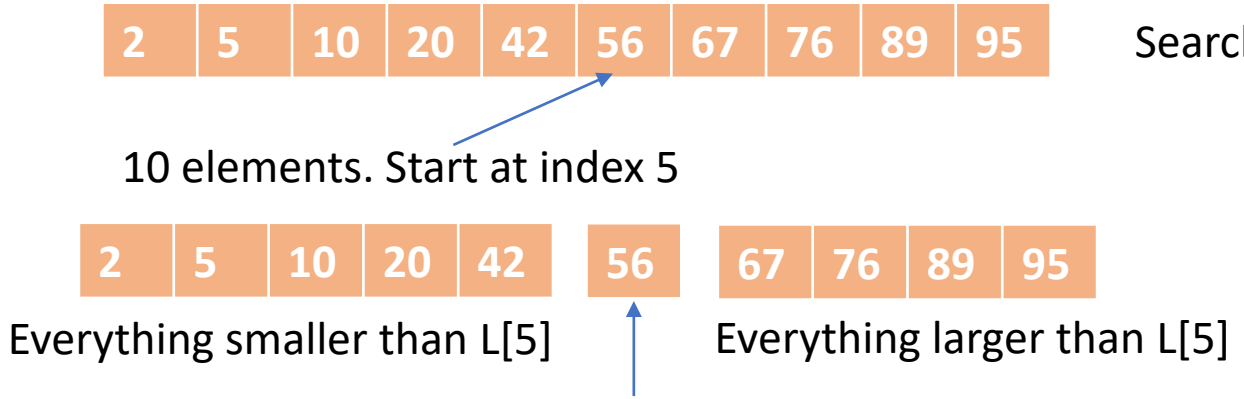
3 checks instead of 10!





Binary Search

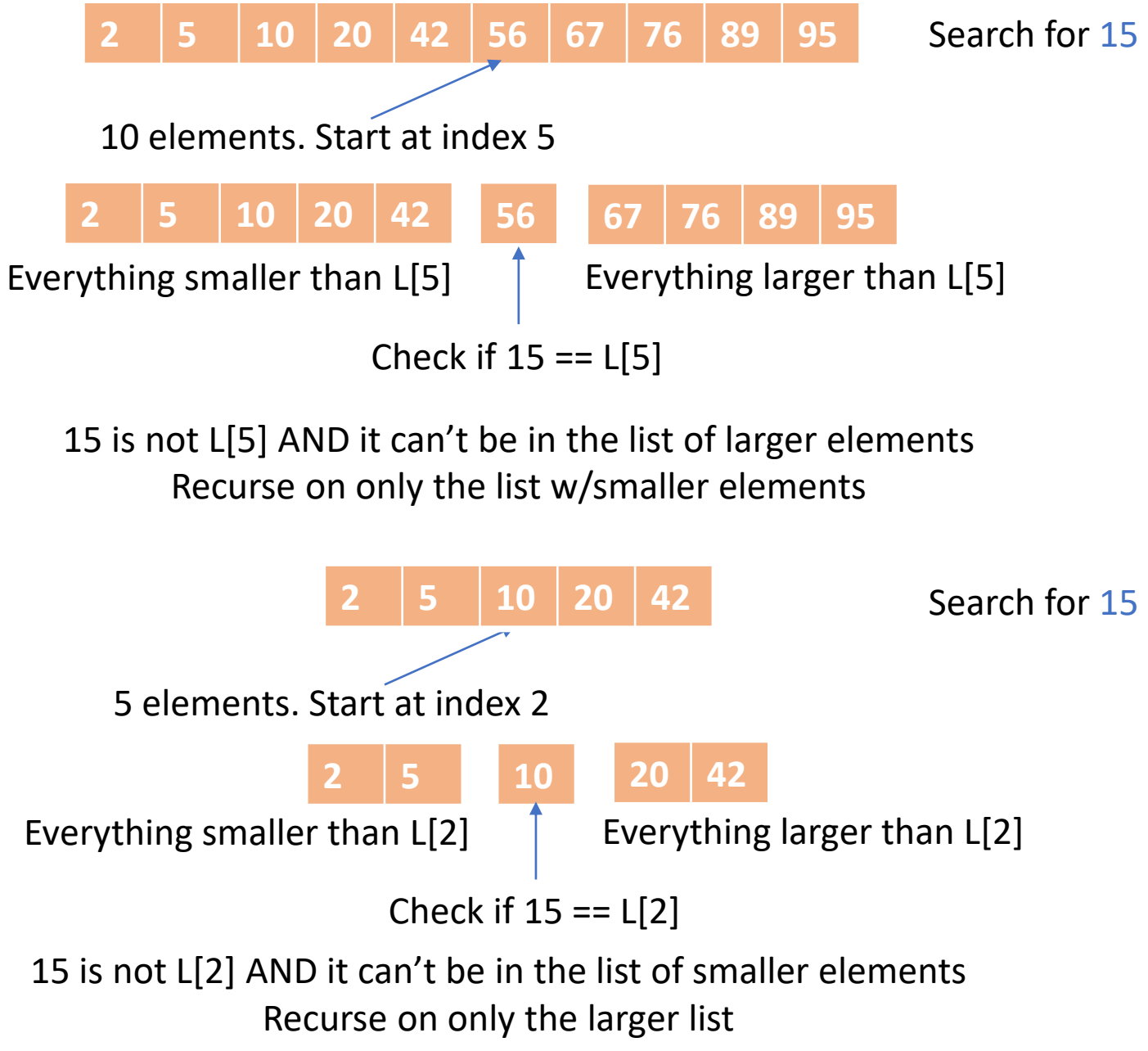
For a sorted list, compare to the middle element and recurse on the side that the element is on



15 is not L[5] AND it can't be in the list of larger elements
Recurse on only the list w/smaller elements

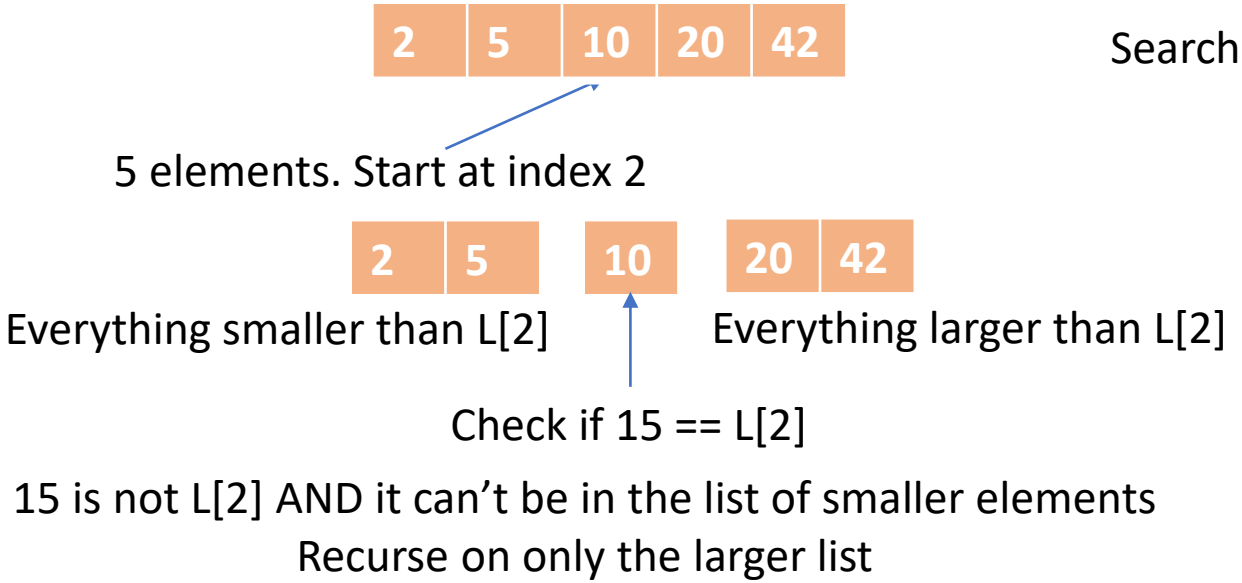
Binary Search

For a sorted list, compare to the middle element and recurse on the side that the element is on



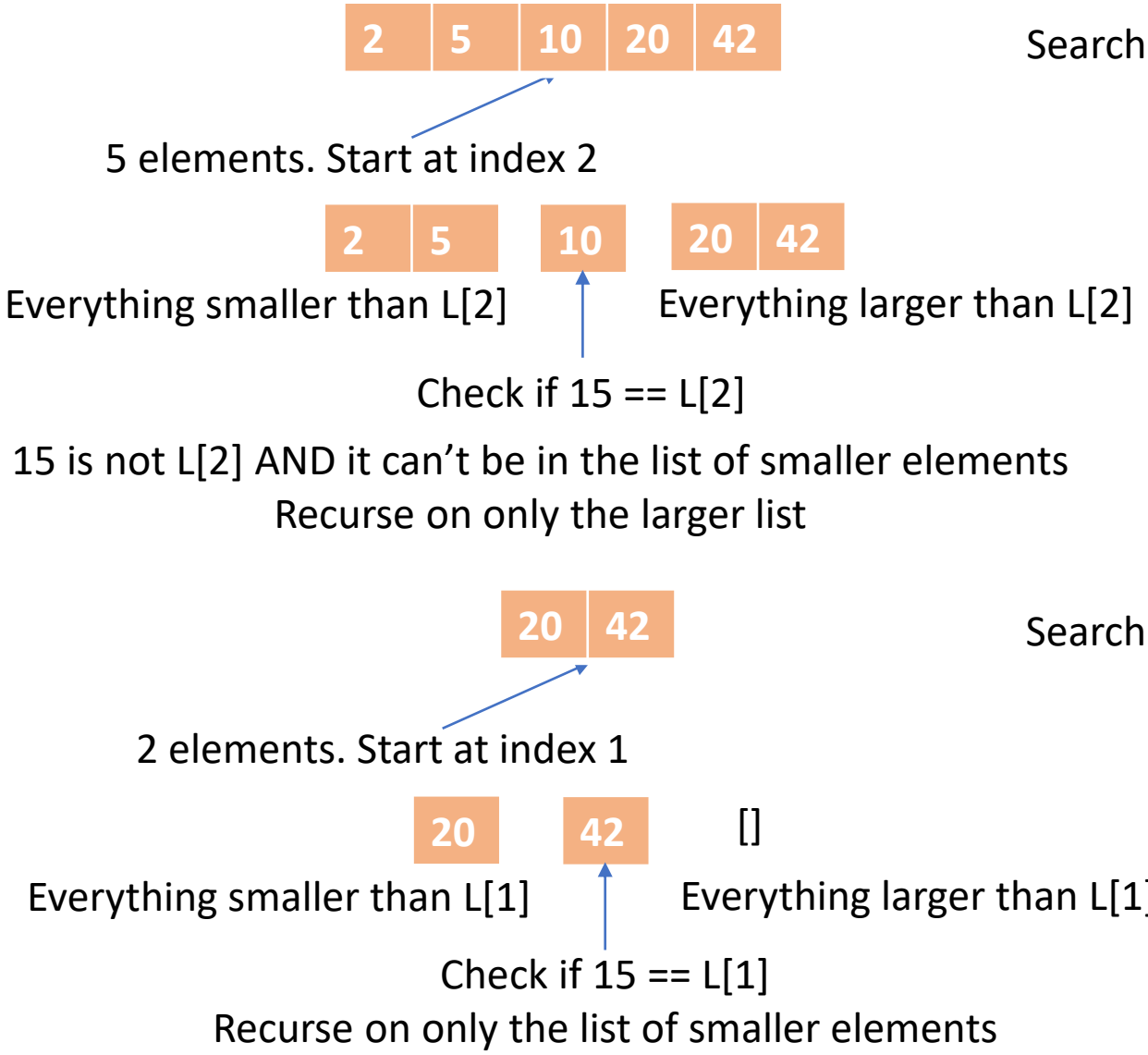
Binary Search

For a sorted list, compare to the middle element and recurse on the side that the element is on



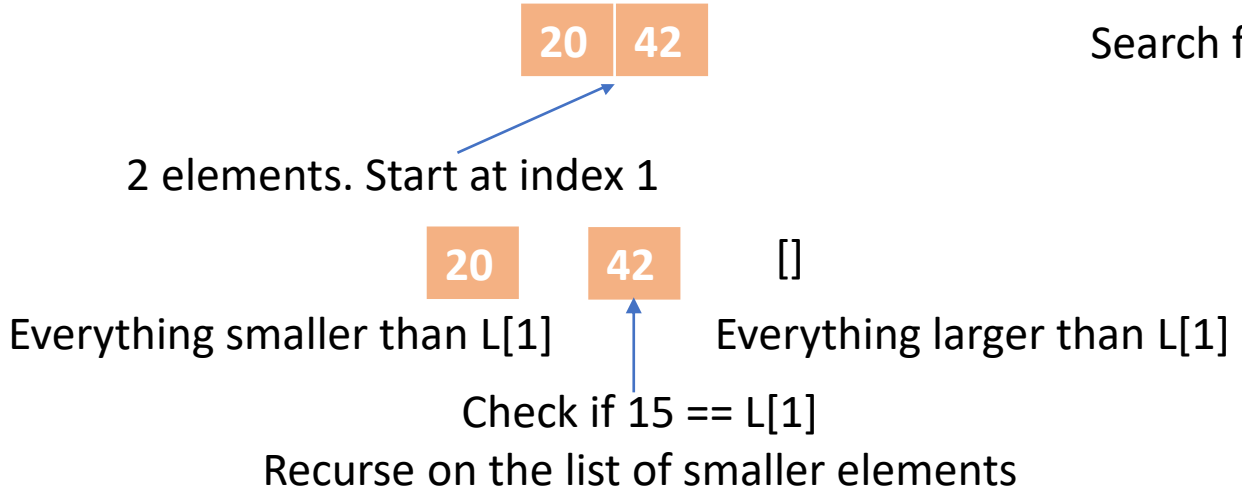
Binary Search

For a sorted list, compare to the middle element and recurse on the side that the element is on



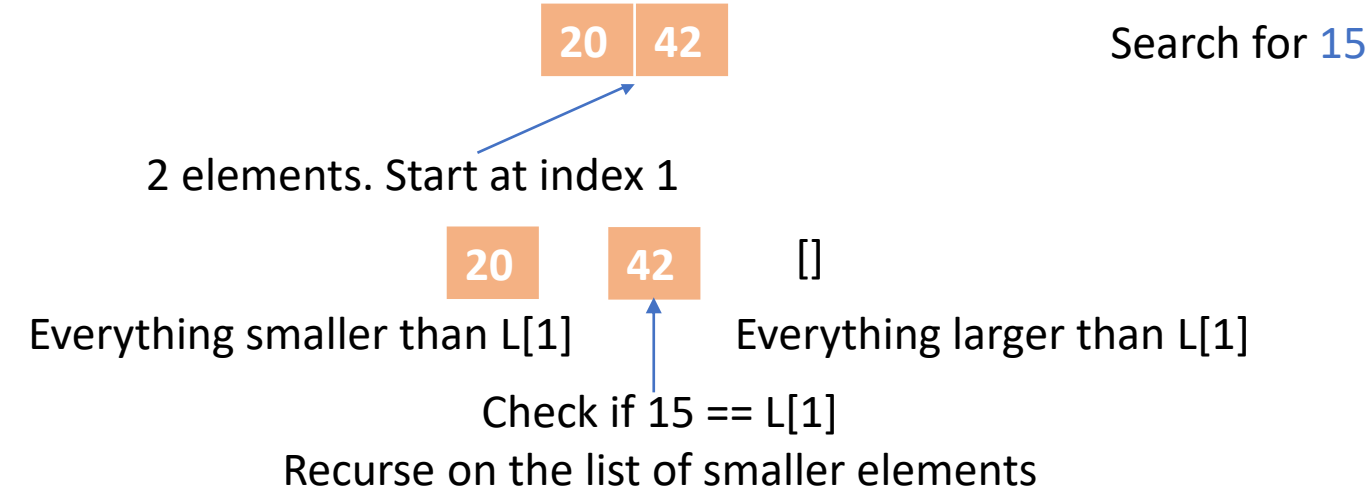
Binary Search

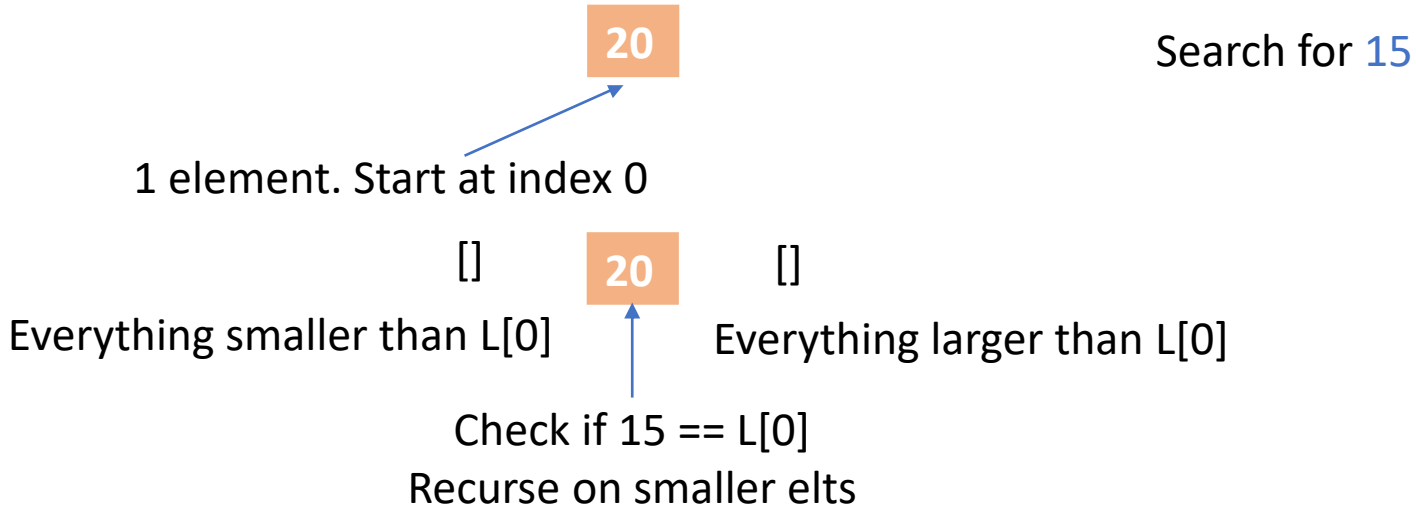
For a sorted list, compare to the middle element and recurse on the side that the element is on



Binary Search

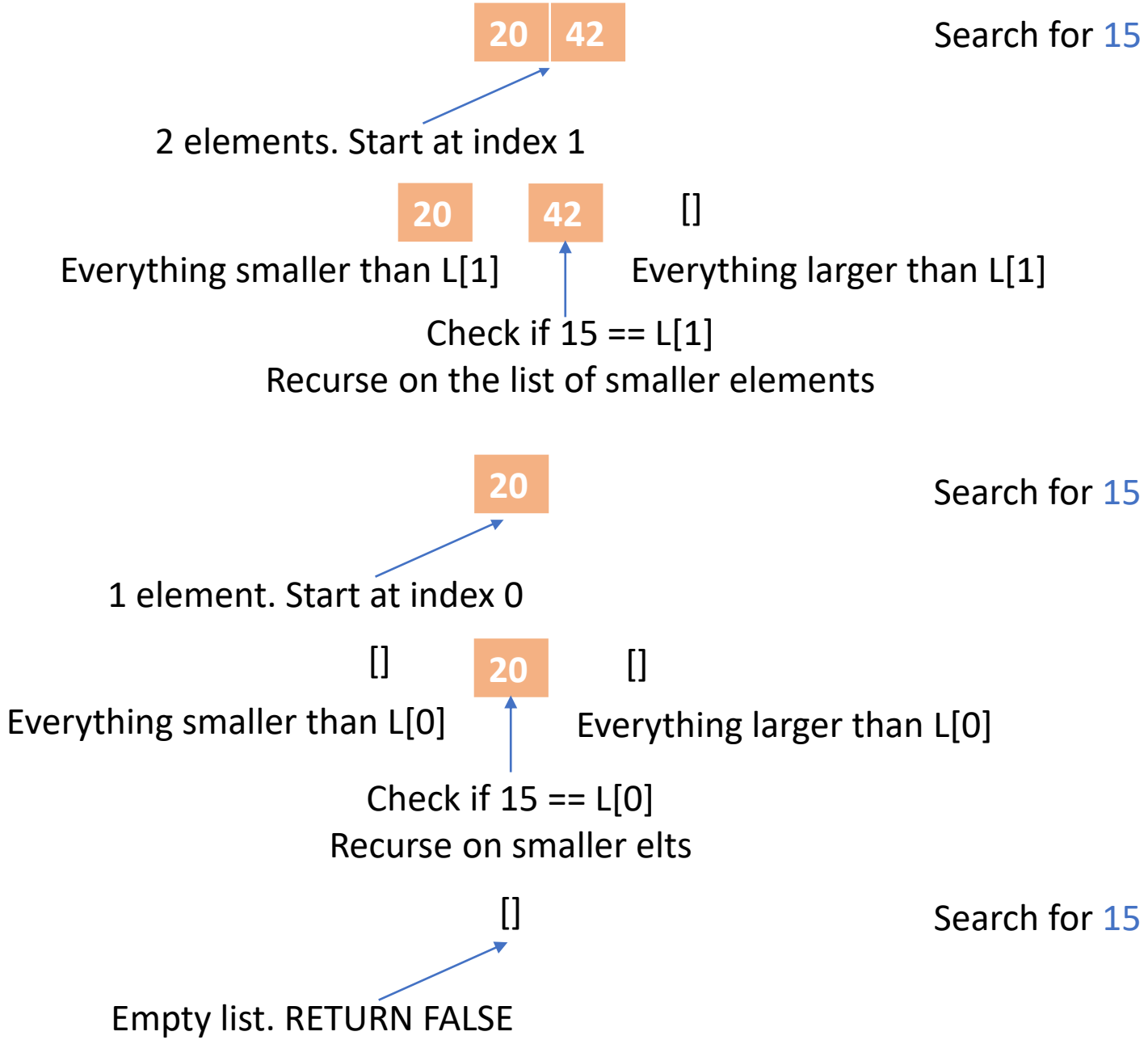
For a sorted list, compare to the middle element and recurse on the side that the element is on





Binary Search

For a sorted list, compare to the middle element and recurse on the side that the element is on



Binary Search: Describe in your own words

Describe what is happening in 30 words or less

Binary Search: Base Cases

What are the base cases?

Binary Search: Base Cases

What are the base cases?

If the list is empty, we know NOTHING can be in the list. Return False

If the middle element is our element, we know we found it. Return True

Binary Search

```
def binarySearch(L,item):
    if len(L) == 0: #base case is some smallest case
        return False
    middle = len(L)//2
    if L[middle] == item: #base case is some smallest case
        return True
```

Base Cases

Binary Search: Induction Step

If the list is not empty, and the middle element isn’t the right one,

What is the induction or recursive step?

Binary Search: Induction Step

If the list is not empty, and the middle element isn't the right one,

Determine if the element would be in the list of smaller elements or the list of larger elements, and recurse on that sub-list.

Binary Search

```
def binarySearch(L,item):
    if len(L) == 0: #base case is some smallest case
        return False
    middle = len(L)//2
    if L[middle] == item: #base case is some smallest case
        return True
    elif L[middle] > item:
        return binarySearch(L[:middle],item)
    else:
        return binarySearch(L[middle+1:],item)
```

Base Cases

Induction Step

Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
            if len(L) == 0: return False
            middle = len(L)//2
            if L[middle] == item: return True
            elif L[middle] > item: return binarySearch(L[:middle],item)
            else: return binarySearch(L[middle+1:],item)
```

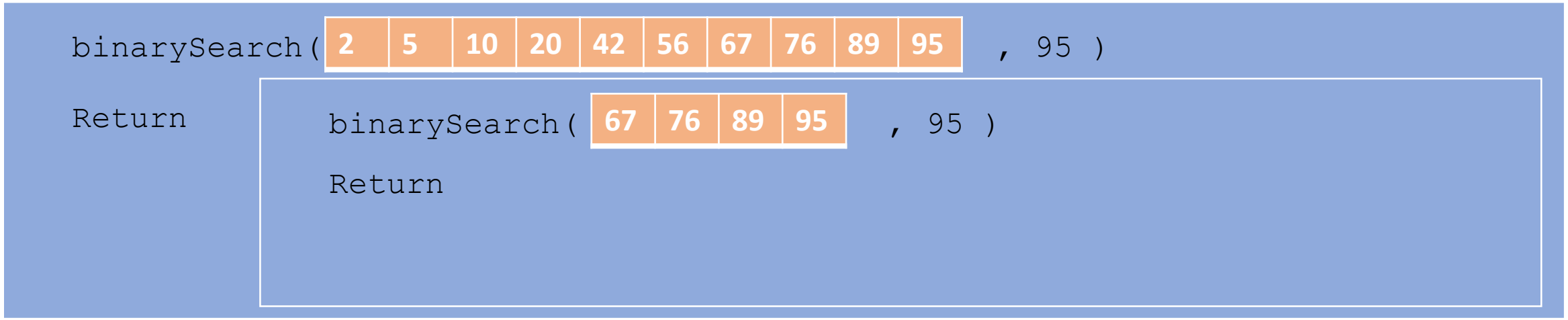
binarySearch(

, 95)

Return

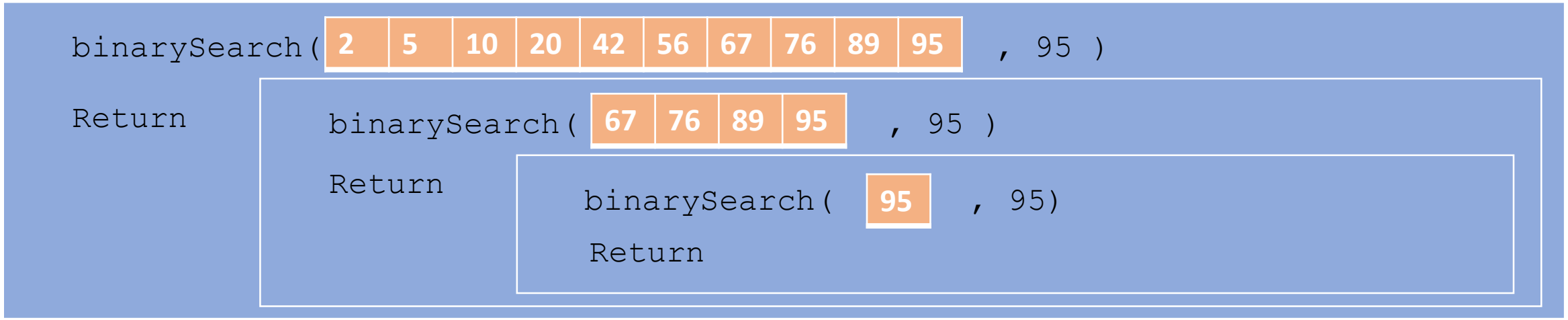
Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
                if len(L) == 0: return False
                middle = len(L)//2
                if L[middle] == item: return True
                elif L[middle] > item: return binarySearch(L[:middle],item)
                else: return binarySearch(L[middle+1:],item)
```



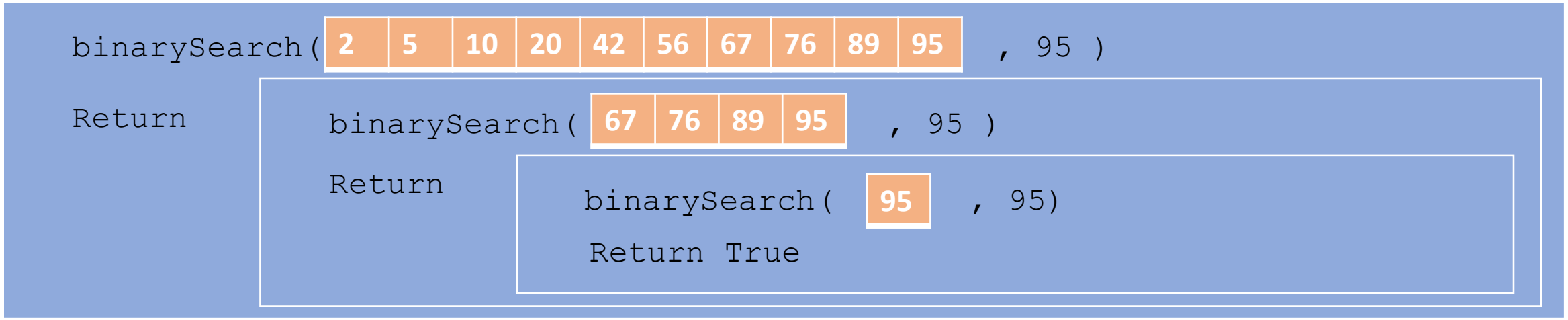
Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
                if len(L) == 0: return False
                middle = len(L)//2
                if L[middle] == item: return True
                elif L[middle] > item: return binarySearch(L[:middle],item)
                else: return binarySearch(L[middle+1:],item)
```



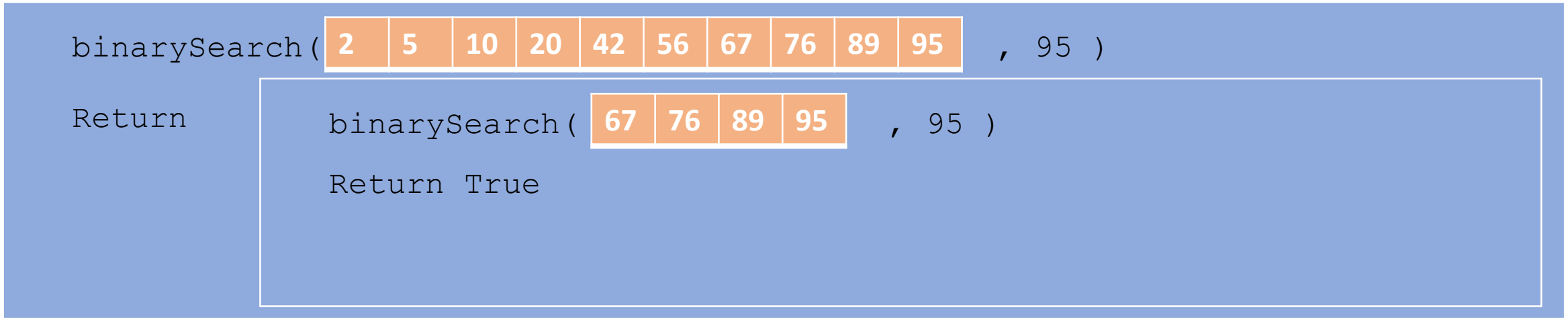
Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
            if len(L) == 0: return False
            middle = len(L)//2
            if L[middle] == item: return True
            elif L[middle] > item: return binarySearch(L[:middle],item)
            else: return binarySearch(L[middle+1:],item)
```



Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
                if len(L) == 0: return False
                middle = len(L)//2
                if L[middle] == item: return True
                elif L[middle] > item: return binarySearch(L[:middle],item)
                else: return binarySearch(L[middle+1:],item)
```



Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
            if len(L) == 0: return False
            middle = len(L)//2
            if L[middle] == item: return True
            elif L[middle] > item: return binarySearch(L[:middle],item)
            else: return binarySearch(L[middle+1:],item)
```

```
binarySearch( 2 | 5 | 10 | 20 | 42 | 56 | 67 | 76 | 89 | 95 , 95 )

Return True
```

Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
                if len(L) == 0: return False
                middle = len(L)//2
                if L[middle] == item: return True
                elif L[middle] > item: return binarySearch(L[:middle],item)
                else: return binarySearch(L[middle+1:],item)
```

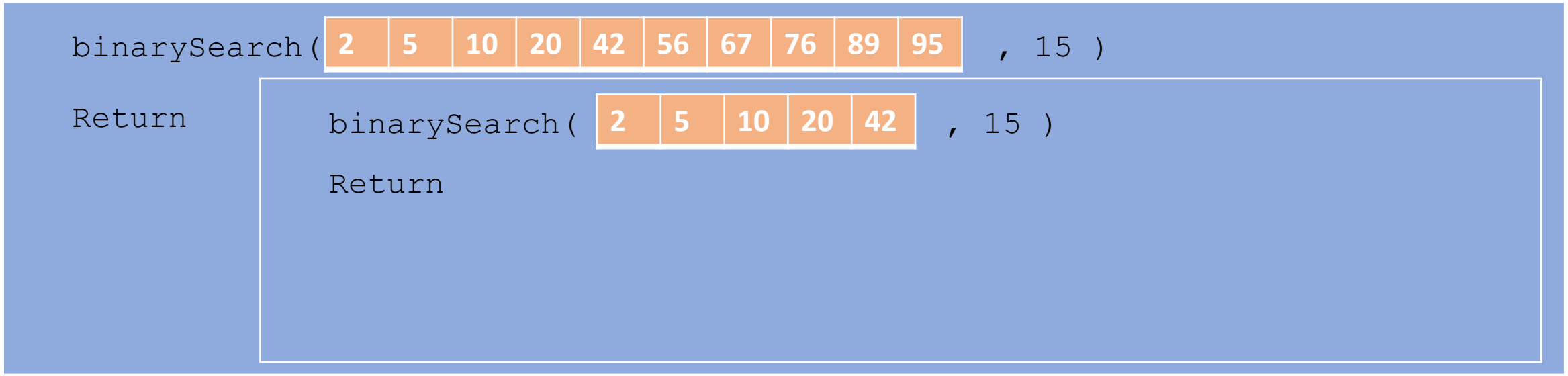
binarySearch(

, 15)

Return

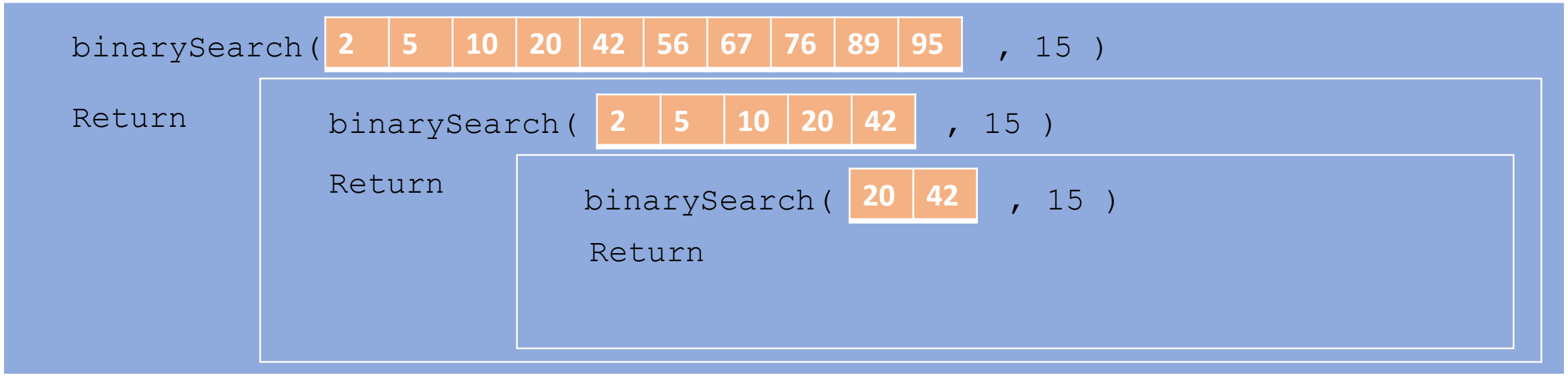
Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
                if len(L) == 0: return False
                middle = len(L)//2
                if L[middle] == item: return True
                elif L[middle] > item: return binarySearch(L[:middle],item)
                else: return binarySearch(L[middle+1:],item)
```



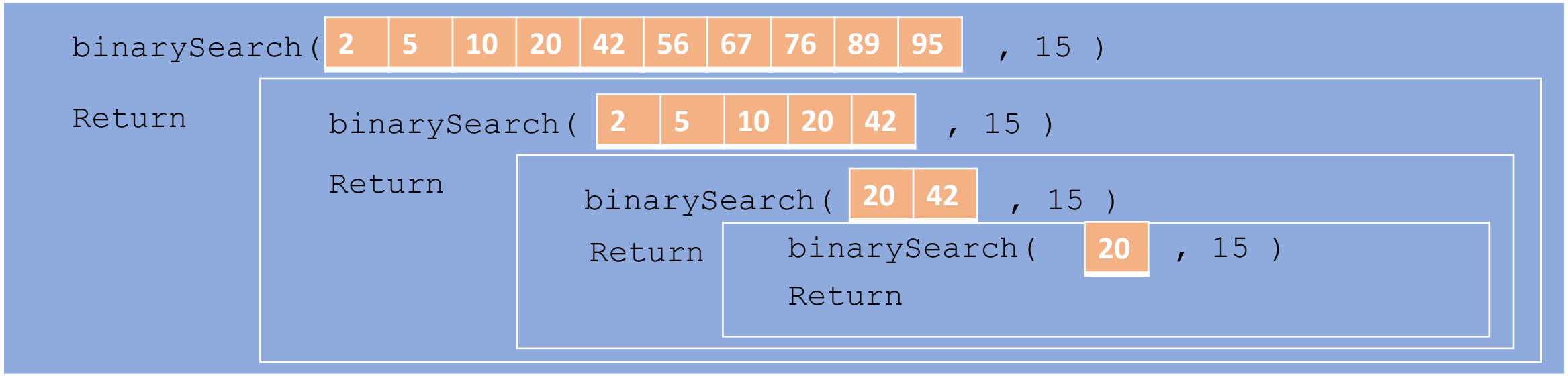
Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
            if len(L) == 0: return False
            middle = len(L)//2
            if L[middle] == item: return True
            elif L[middle] > item: return binarySearch(L[:middle],item)
            else: return binarySearch(L[middle+1:],item)
```



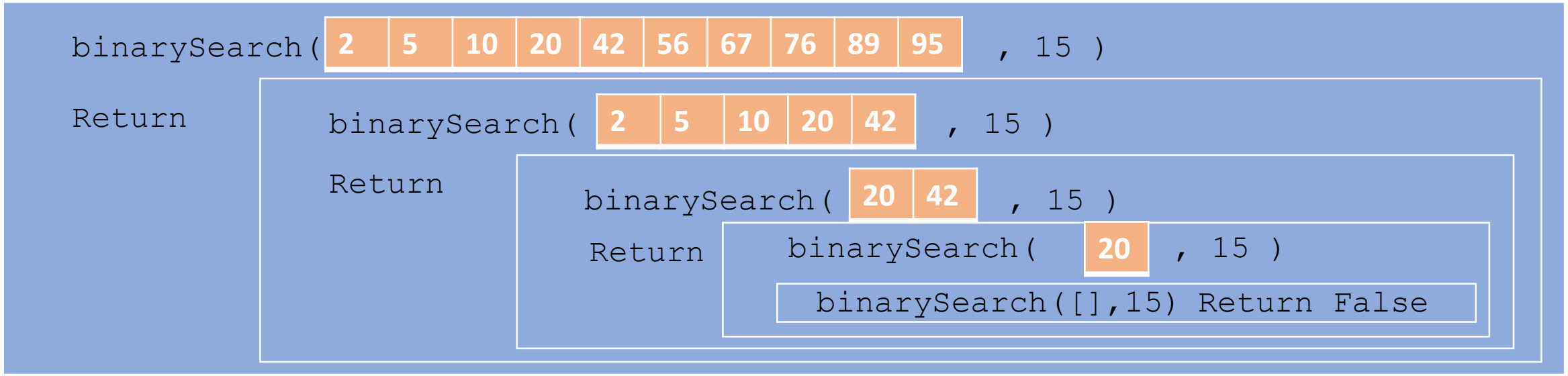
Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
                if len(L) == 0: return False
                middle = len(L)//2
                if L[middle] == item: return True
                elif L[middle] > item: return binarySearch(L[:middle],item)
                else: return binarySearch(L[middle+1:],item)
```



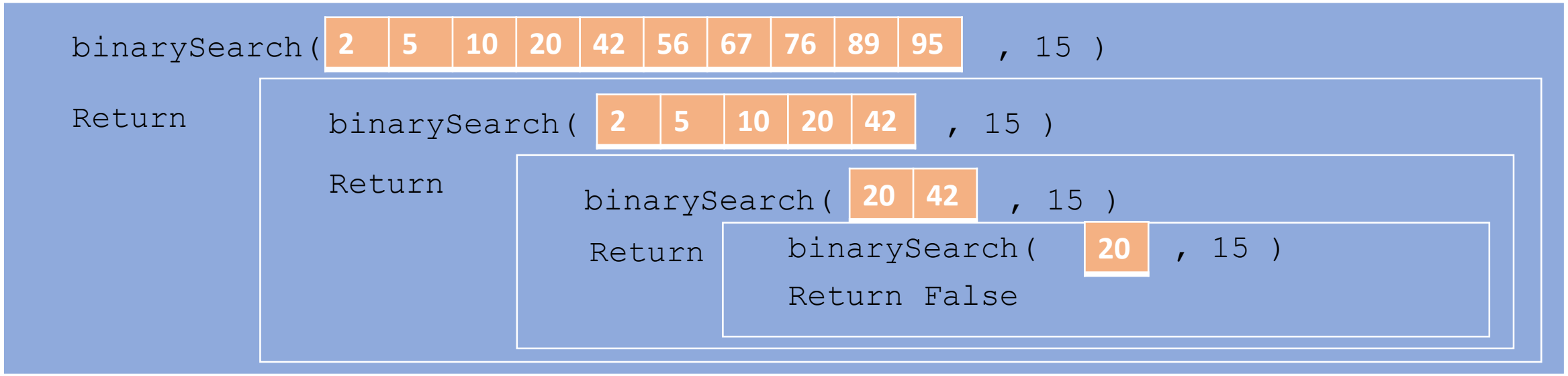
Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
                if len(L) == 0: return False
                middle = len(L)//2
                if L[middle] == item: return True
                elif L[middle] > item: return binarySearch(L[:middle],item)
                else: return binarySearch(L[middle+1:],item)
```



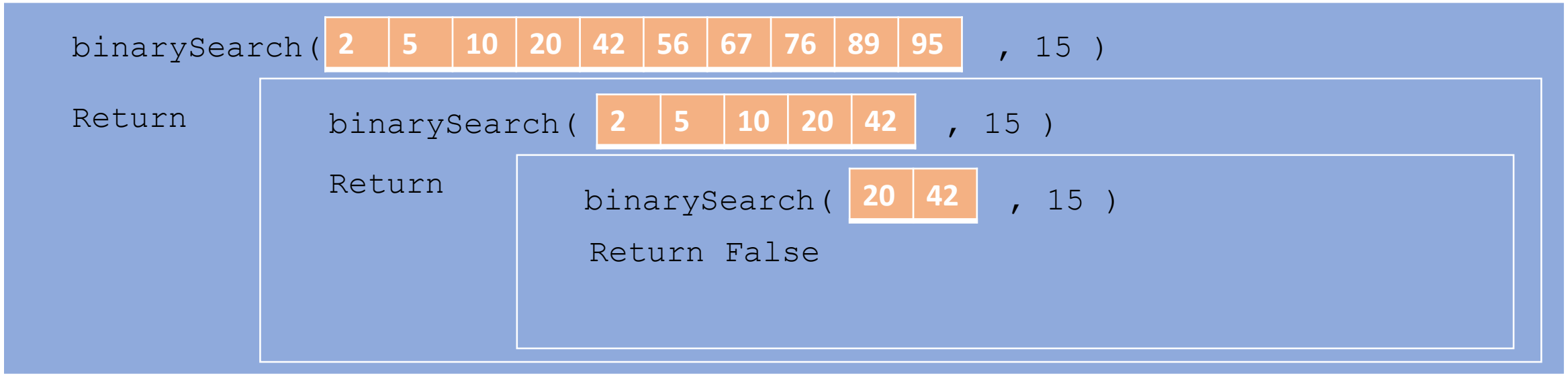
Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
                if len(L) == 0: return False
                middle = len(L)//2
                if L[middle] == item: return True
                elif L[middle] > item: return binarySearch(L[:middle],item)
                else: return binarySearch(L[middle+1:],item)
```



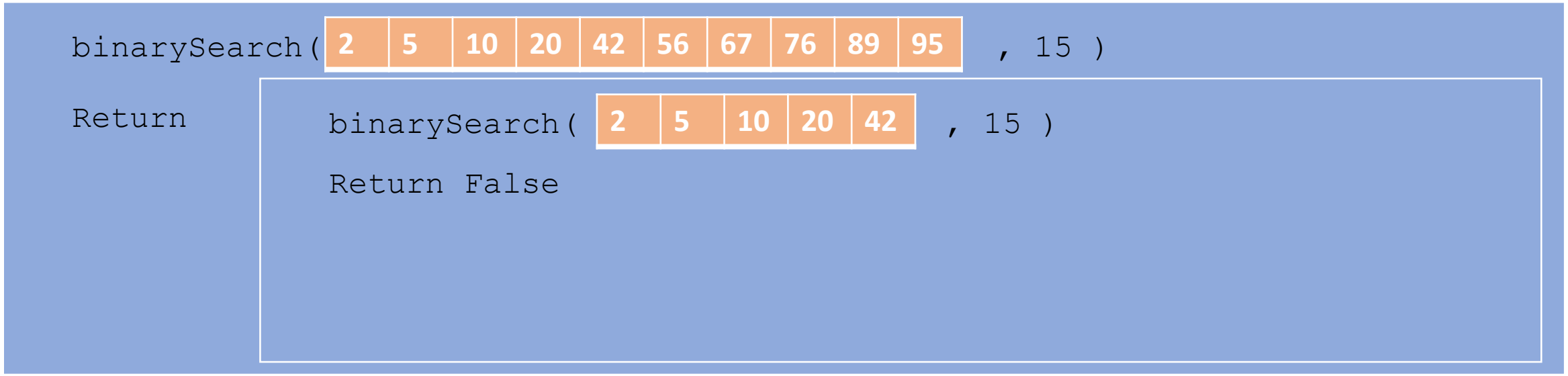
Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
            if len(L) == 0: return False
            middle = len(L)//2
            if L[middle] == item: return True
            elif L[middle] > item: return binarySearch(L[:middle],item)
            else: return binarySearch(L[middle+1:],item)
```



Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
                if len(L) == 0: return False
                middle = len(L)//2
                if L[middle] == item: return True
                elif L[middle] > item: return binarySearch(L[:middle],item)
                else: return binarySearch(L[middle+1:],item)
```



Trace Binary Search

```
def binarySearch(L,item): #newlines removed for space
            if len(L) == 0: return False
            middle = len(L)//2
            if L[middle] == item: return True
            elif L[middle] > item: return binarySearch(L[:middle],item)
            else: return binarySearch(L[middle+1:],item)
```

```
binarySearch( 2 | 5 | 10 | 20 | 42 | 56 | 67 | 76 | 89 | 95 , 15 )

Return False
```

Big Picture

If we can make an assumption about what our list looks like, we can search faster (with fewer comparisons) than linear search.

- Dividing in half is a really good way to reduce the comparisons
- Recursion allows us to solve the problem on half the size

Next time we'll talk about how to formally make those comparisons between different implementations of algorithms.