# ECS32B

Introduction to Data Structures

Binary Search

Lecture 17

## Announcements

- Midterms are being graded. You will get back your scanned graded midterm next week.
- The final will have a more leisurely pace, i.e. more time per question.
- Homework 4 up today and will be due Tuesday May 14 at 11:59pm
- Go to discussion this week. It will cover Midterm solutions and the Homework 4 topics Search and Hashing. If needed, you can try to attend an alternate section.

# Binary Search

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

- Now we will look at two different types of binary search.
    - iterative binary search
    - recursive binary search
- Binary search is only useful if the list is ordered. When a collection of items is sorted, binary search for one item can be much faster than a sequential search.

# Binary search with iteration

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

The search starts with finding the item at the midpoint of the list.

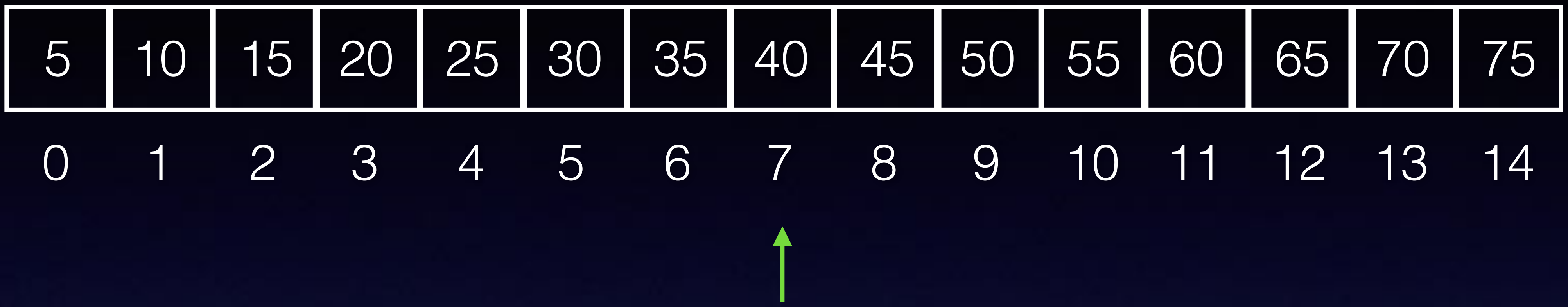
We use **Python integer division** to find the midpoint

first index + last index // 2

This will always **truncate** or **round down** the result to the nearest integer. It is shorthand for converting the result of the division to an integer, which is what we need it to be for a valid list index:

int(first index + last index / 2)

# Calculating the midpoint



| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

For the list above, Python integer division to find the midpoint gives us:

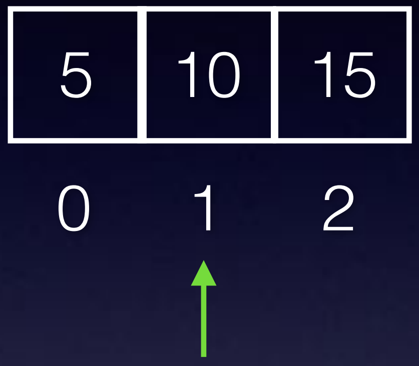
(first index + last index) // 2

(0 + 14) // 2

7

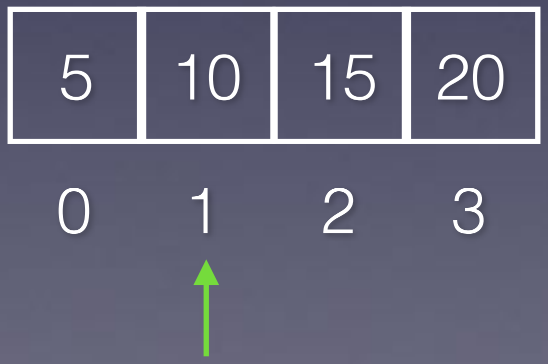
# Calculating the midpoint for iterative binary search

A small example



```
>>(0 + 2) // 2
1
```

A little bigger example

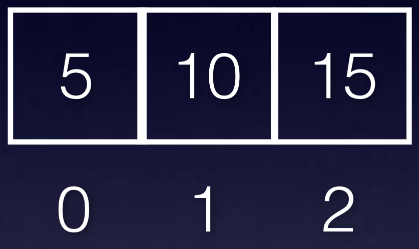


```
>>(0 + 3) // 2
1
```

# Calculating the midpoint for iterative binary search

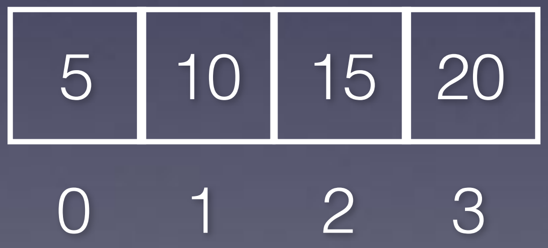
## An equivalent calculation

A small example



```
 >>int( (0 + 2) * 1/2 )
1
```

A little bigger example



```
 >>int( (0 + 2) * 1/2 )
1
```

# Calculating the midpoint for iterative binary search

works at the end of the list

A small example

| | | |
|---|---|---|
| 65 | 70 | 75 |
| 12 | 13 | 14 |

```
 >>int( (12 + 14) * 1/2 )
13
```

A little bigger example

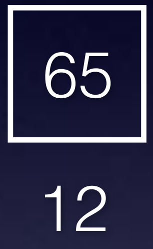
| | | | |
|---|---|---|---|
| 60 | 65 | 70 | 75 |
| 11 | 12 | 13 | 14 |

```
 >>int( (11 + 14) * 1/2 )
12
```

# Calculating the midpoint for iterative binary search

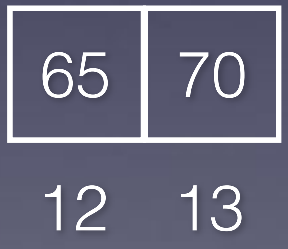
works for very short ranges

A small example



```
 >>int( (12 + 12) * 1/2 )
12
```

A little bigger example



```
 >>int( (12 + 13) * 1/2 )
12
```

# Calculating the midpoint for iterative binary search

Remember to <u>round down</u> with integer division!

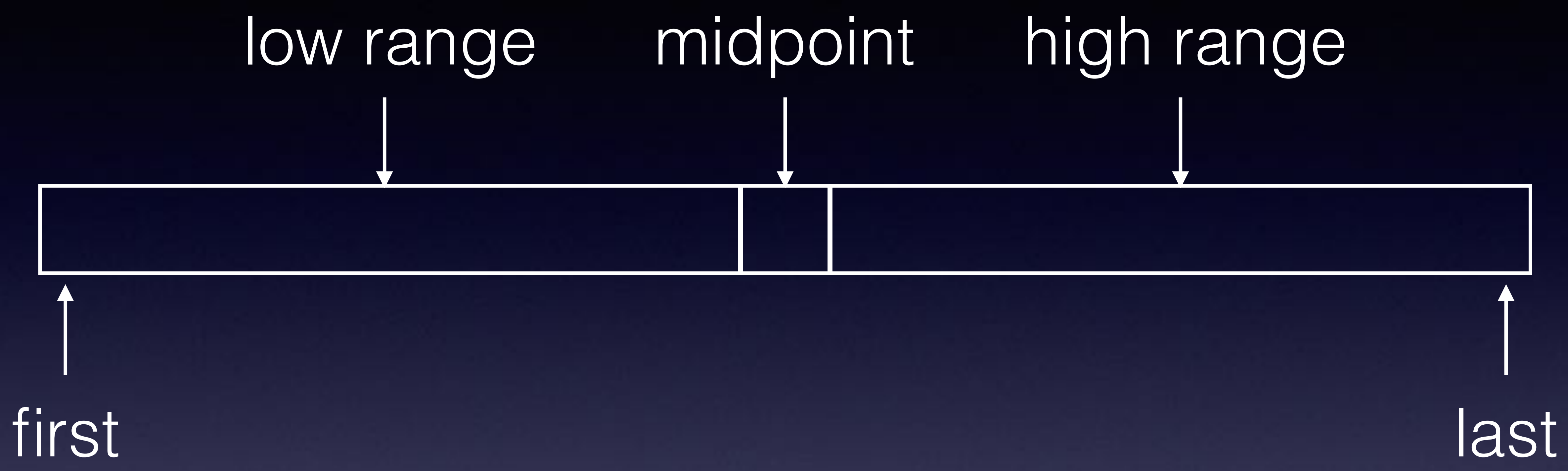
Remember

```
>>(12 + 13) // 2

12
```

Is shorthand for

```
>>int( (12 + 13) * 1/2 )

12
```

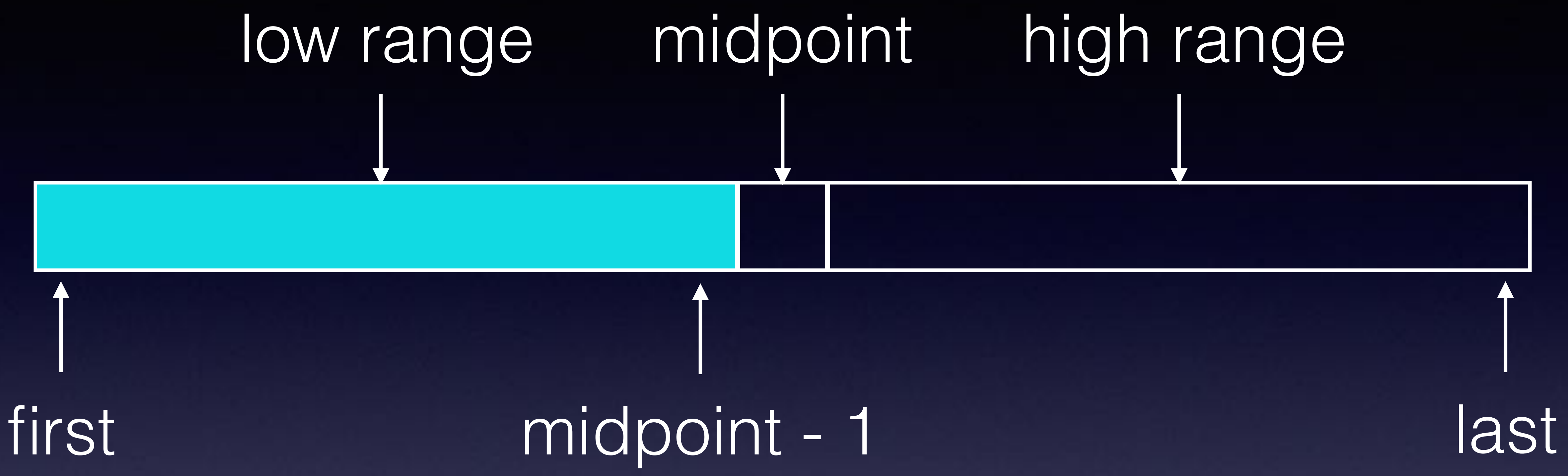
# Divide and conquer



A range specified by a first and last position(inclusive) is divided into three segments, or smaller ranges.

- low range
- midpoint
- high range

# Divide and conquer

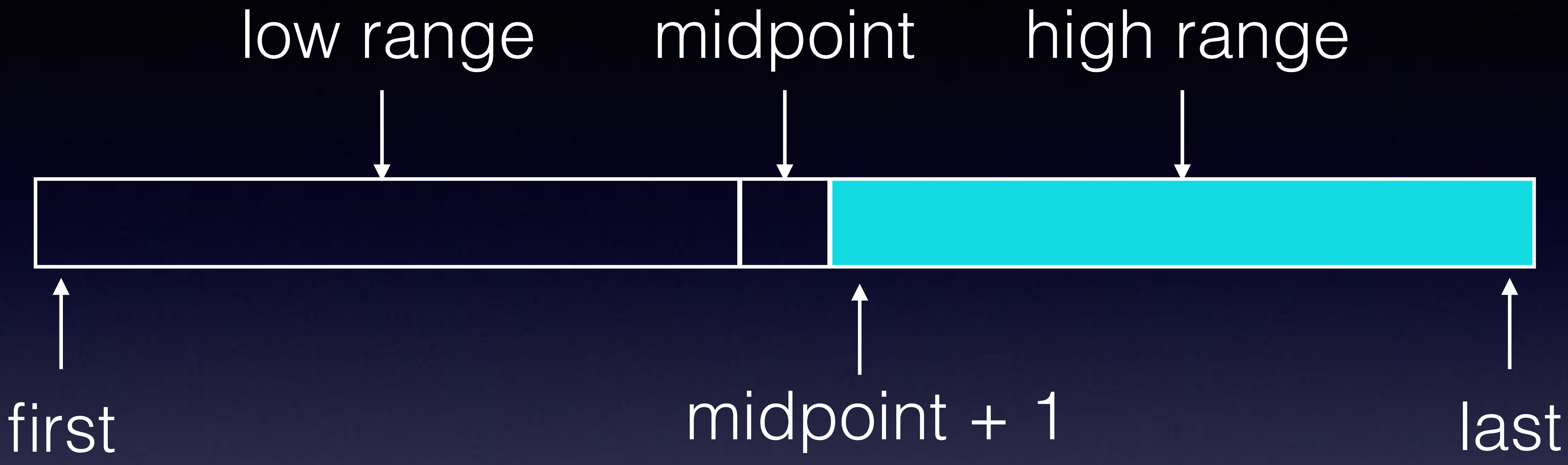


The low range, if it is non-empty, includes the first index to the index just before the midpoint.

The coordinates in the low range are all less than the midpoint coordinate.

The values in the low range are all less than the value at the midpoint coordinate

# Divide and conquer

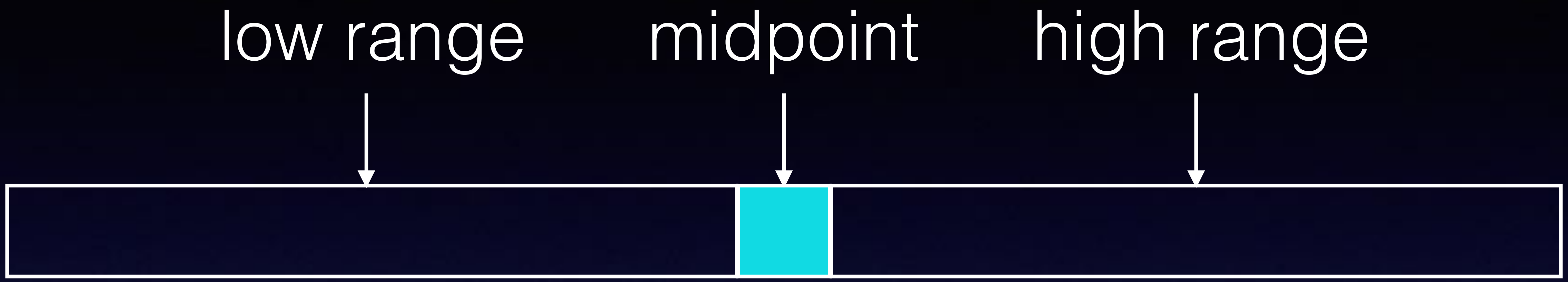


The high range, if it is non-empty, includes the index just after the midpoint to the last index.

The coordinates in the high range are all greater than the midpoint coordinate.

The values in the high range are all greater than the value at the midpoint coordinate

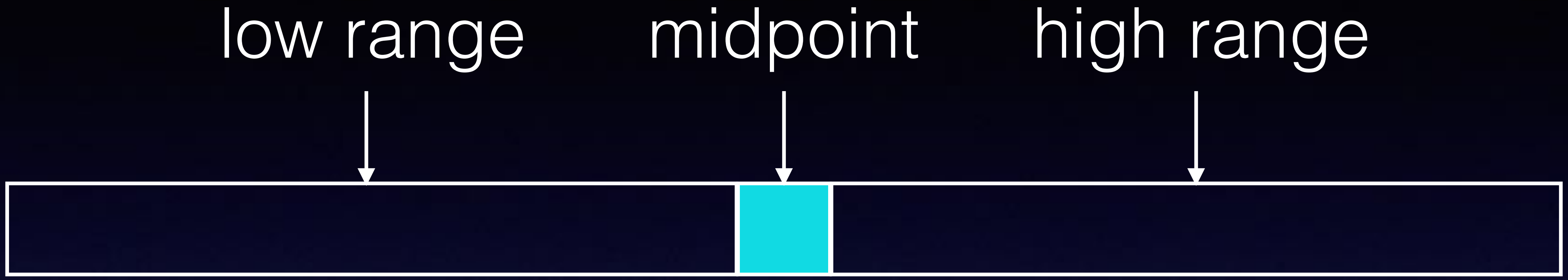
# Divide and conquer



The value at the midpoint coordinate separates the values in the low range from the high range.

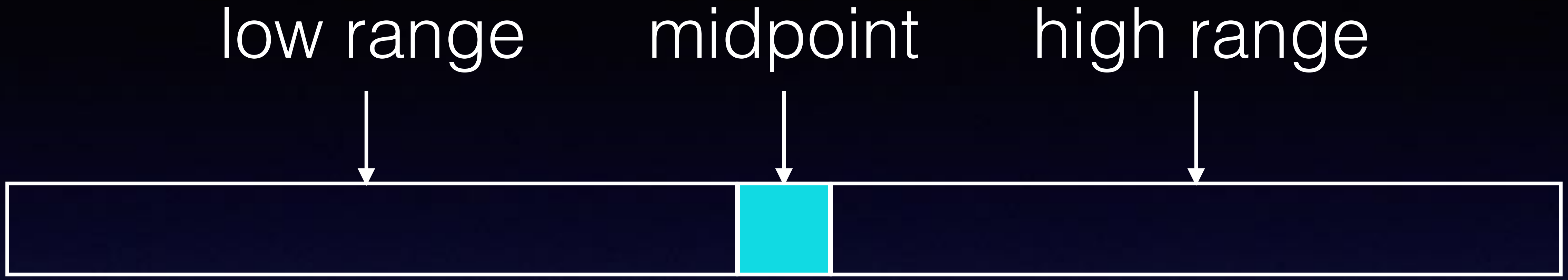
During search a target value is compared to the midpoint value. Three cases arise from this comparison.

# Divide and conquer



Case 1: target value is equal to midpoint value

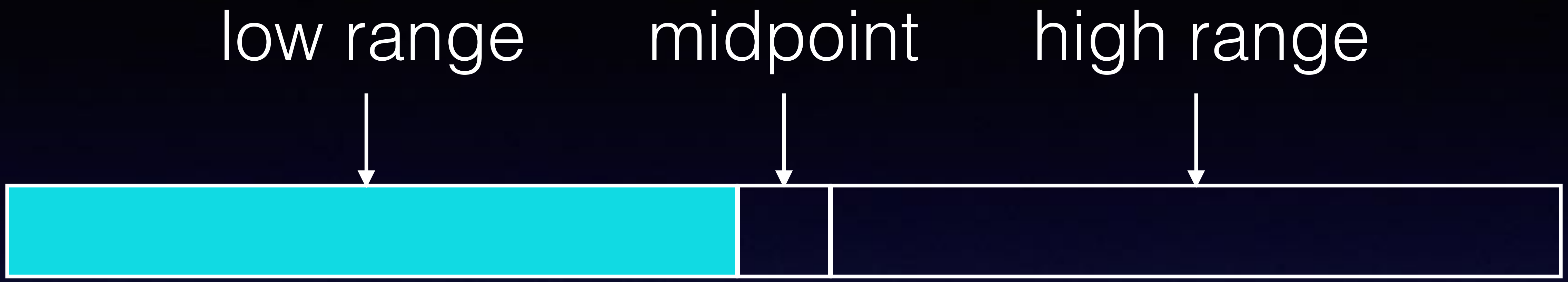
# Divide and conquer



Case 1: target value is equal to midpoint value

Great, we have found it, we return True.

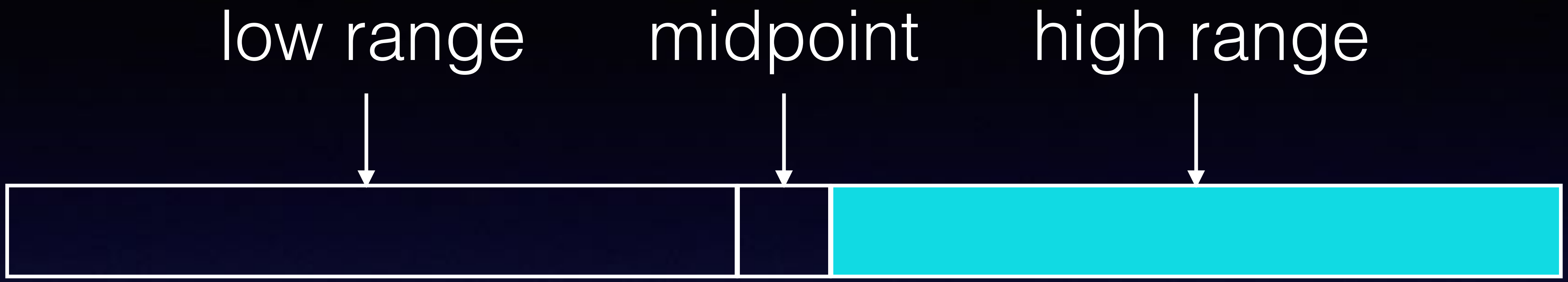
# Divide and conquer



Case 2: target value is *less than* the midpoint value

It must be in the low range!

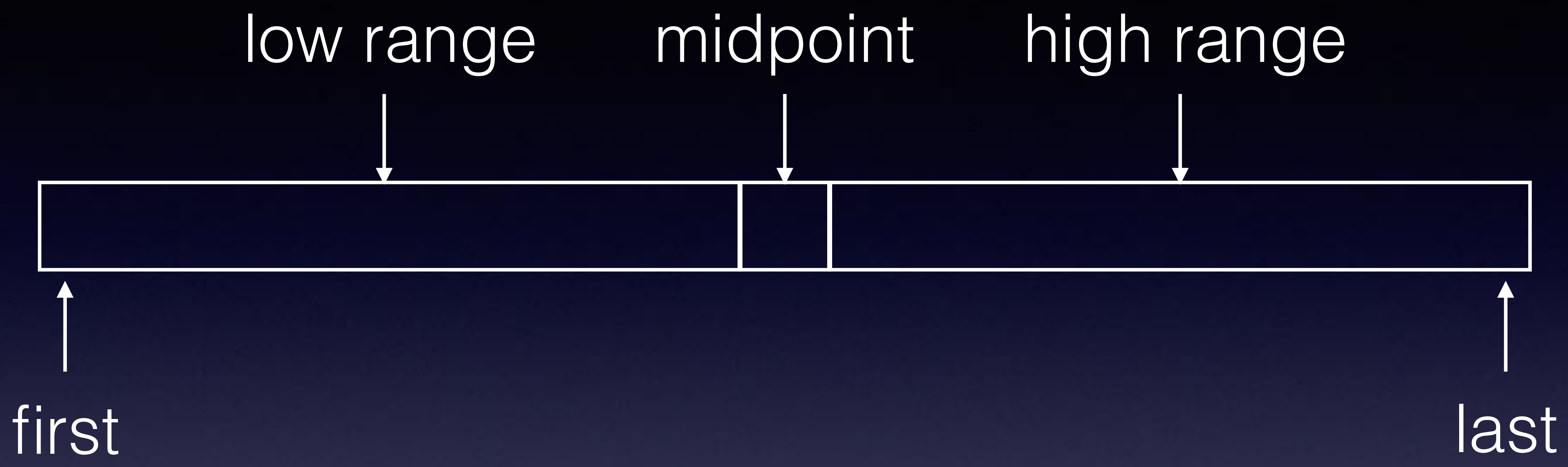
# Divide and conquer



Case 3: target value is *greater than* the midpoint value

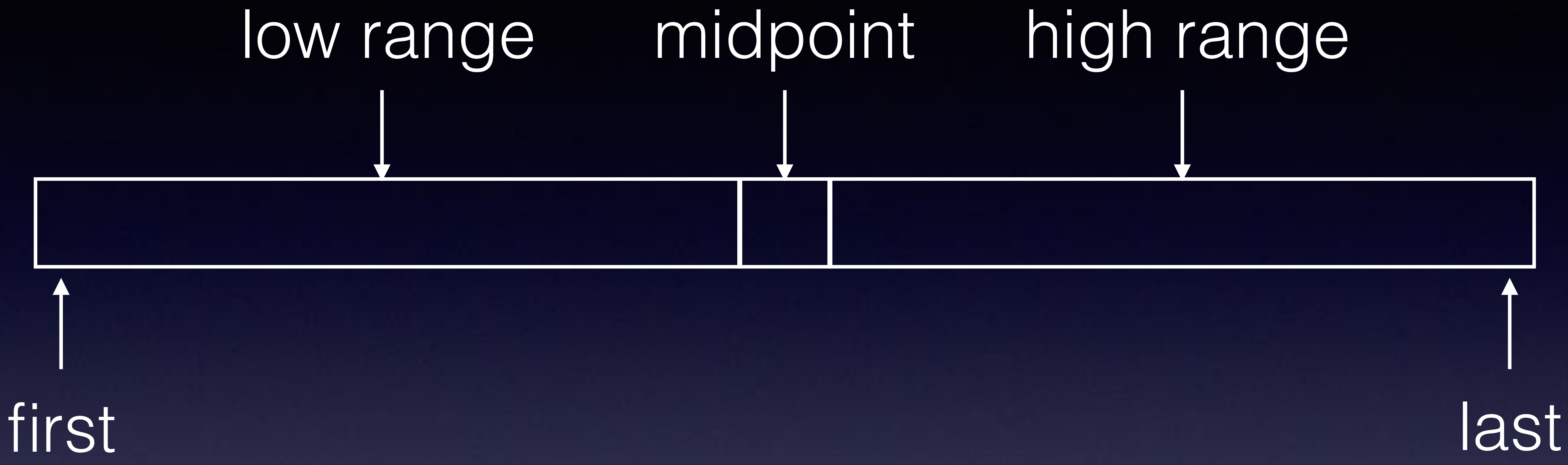
It must be in the high range!

# What happens next?



We start over again on the range that might contain the target value.

# What happens next?

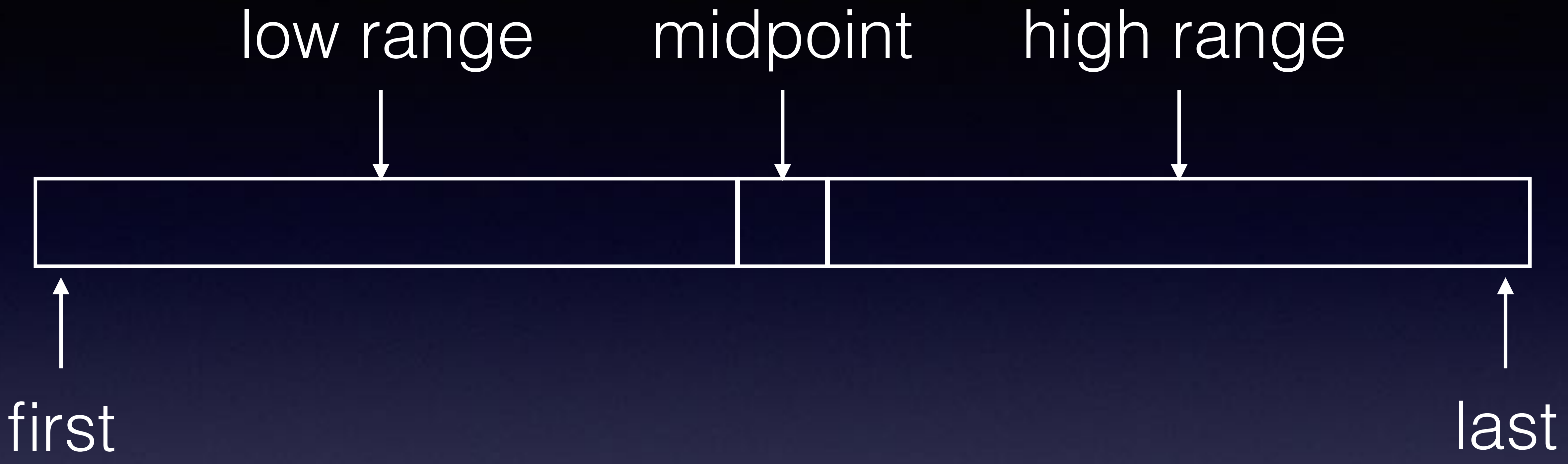


If the new range is empty, then there is no midpoint and we will return False since we have not found our target value.

Otherwise we define these three regions again

- midpoint (if empty return False)
- low range (from first to midpoint - 1, may be empty)
- high range (from midpoint + 1 to last, may be empty)

# What happens next?



This process keeps going iteratively or recursively until we return True, because we found the value, or False, because we ran out of places to look.

# Binary search with iteration

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

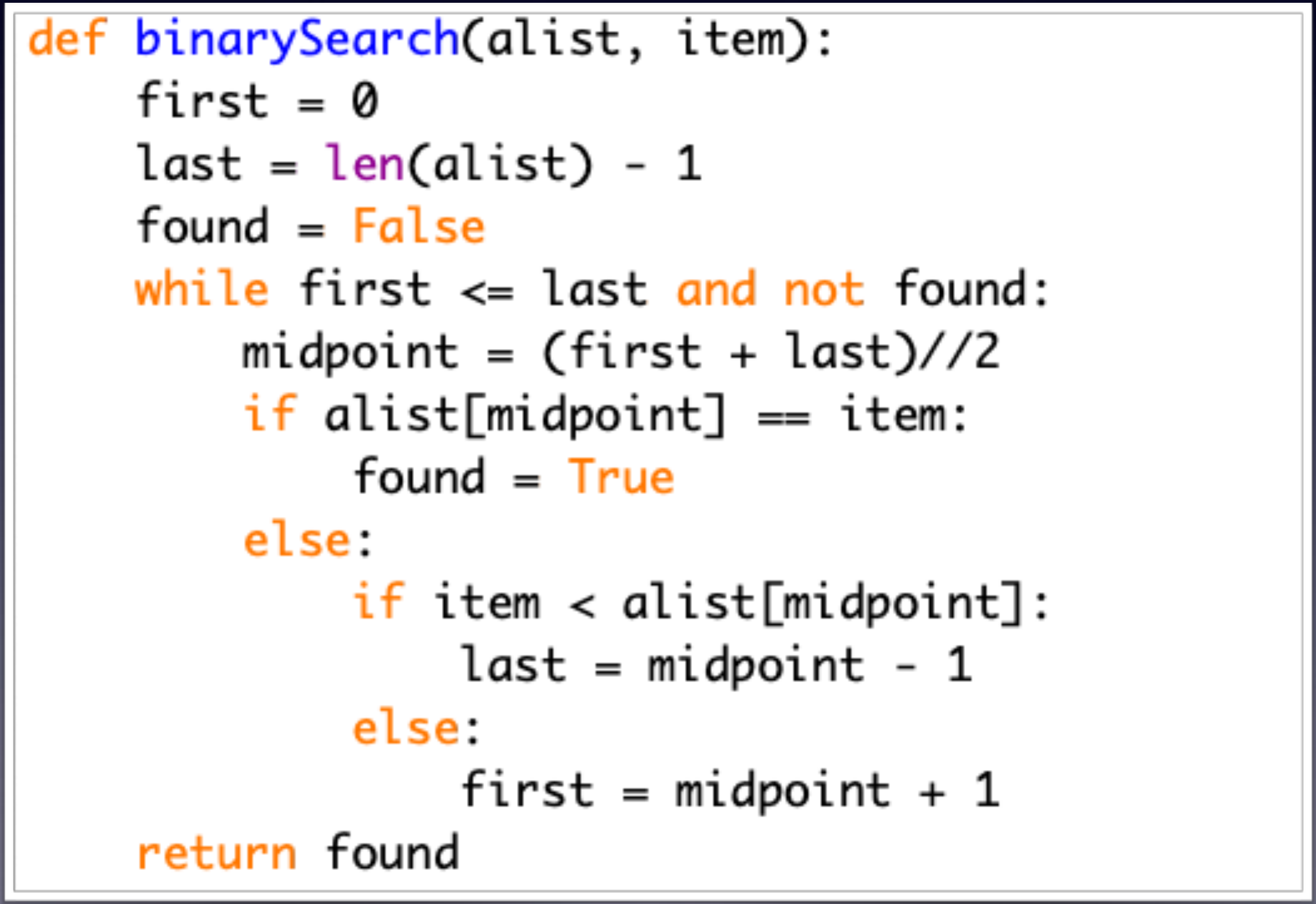
The search starts with finding the item at the midpoint of the entire list. There are three cases when comparing the midpoint value to the target. If equal, then the procedure is done and returns True. If that midpoint value is not the target, then the procedure determines whether the target is less than the midpoint value or greater than (it can't be equal to) the midpoint value.

The loop starts again but on that segment of the list that might contain the target value.

If there is no segment remaining we return False.

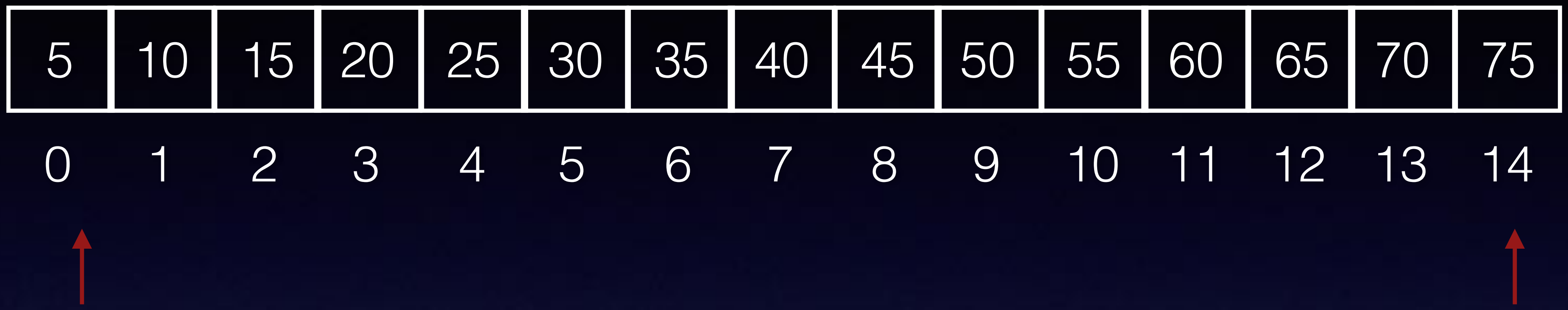
# Binary search with iteration V1

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |



```python
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

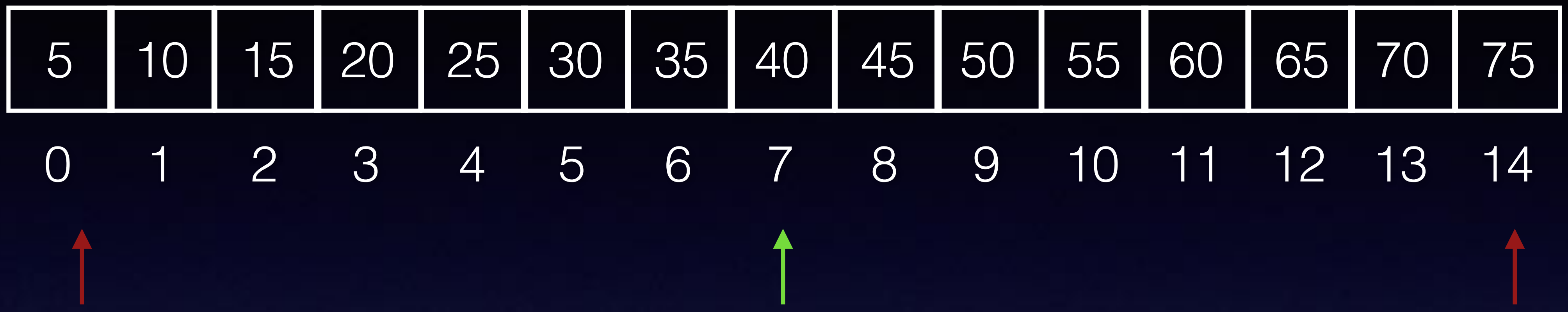
# Binary search with iteration



```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

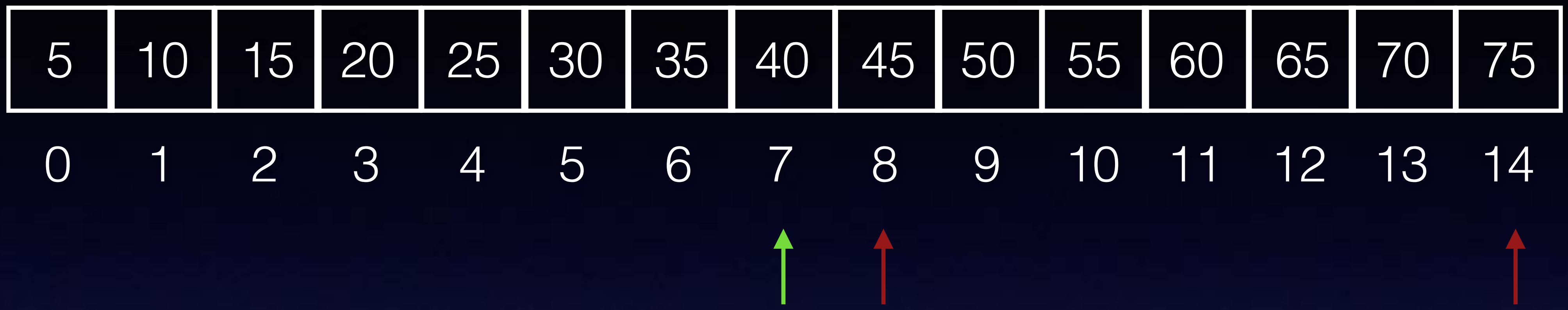
# Binary search with iteration



```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

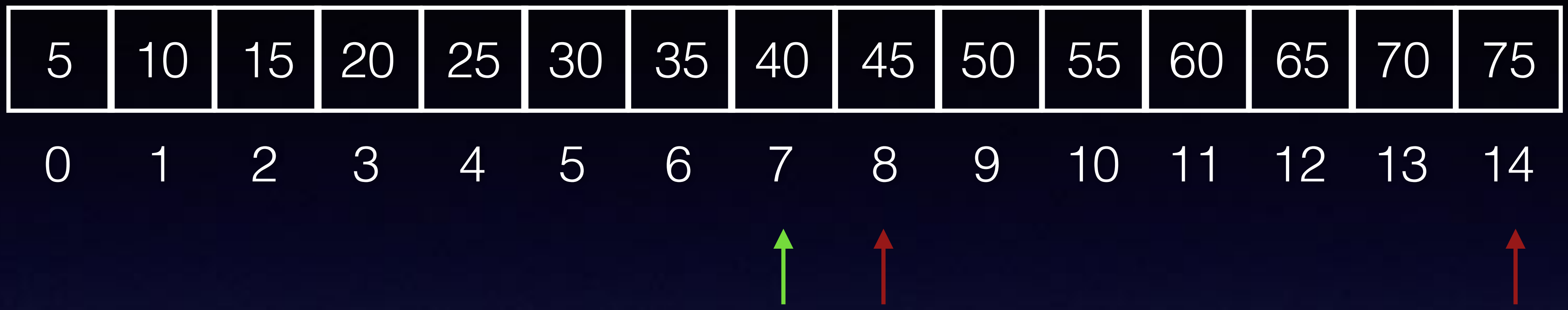
# Binary search with iteration



```python
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

# Binary search with iteration

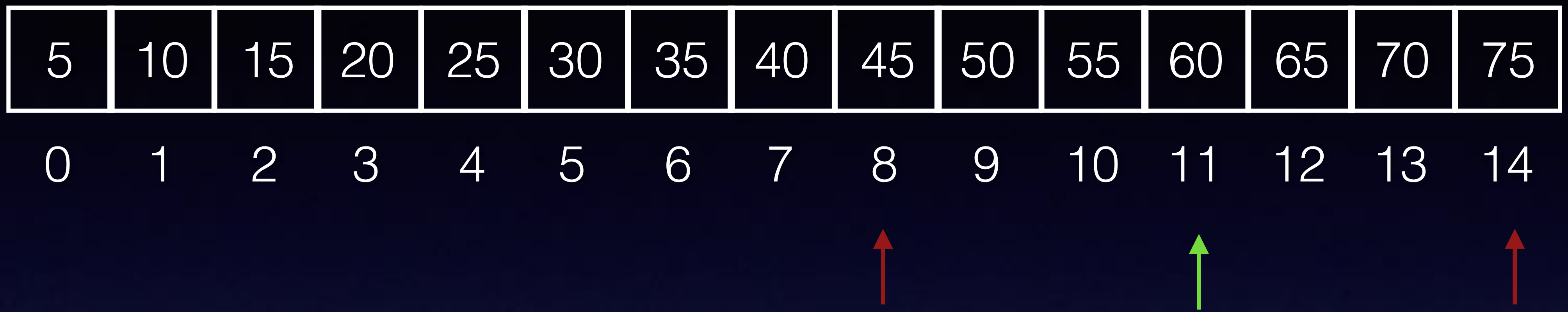


| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

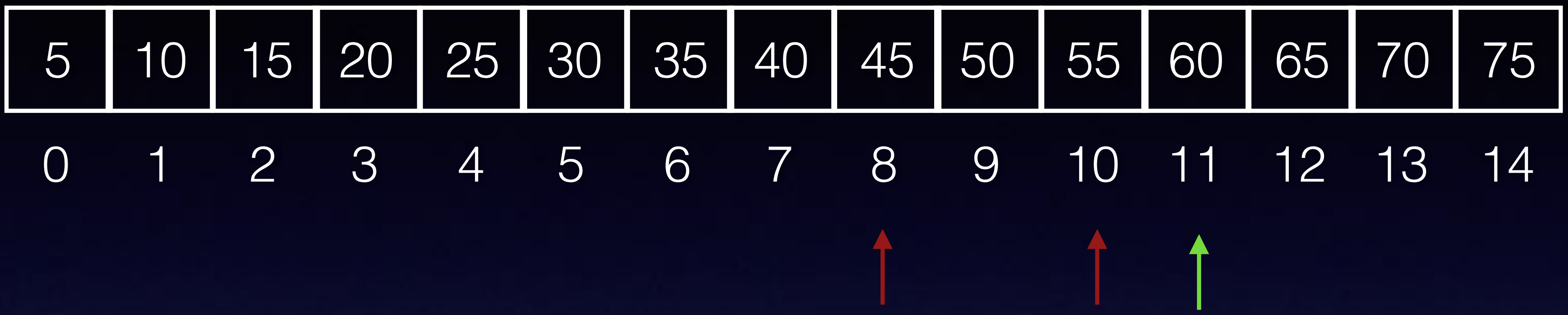
# Binary search with iteration



```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

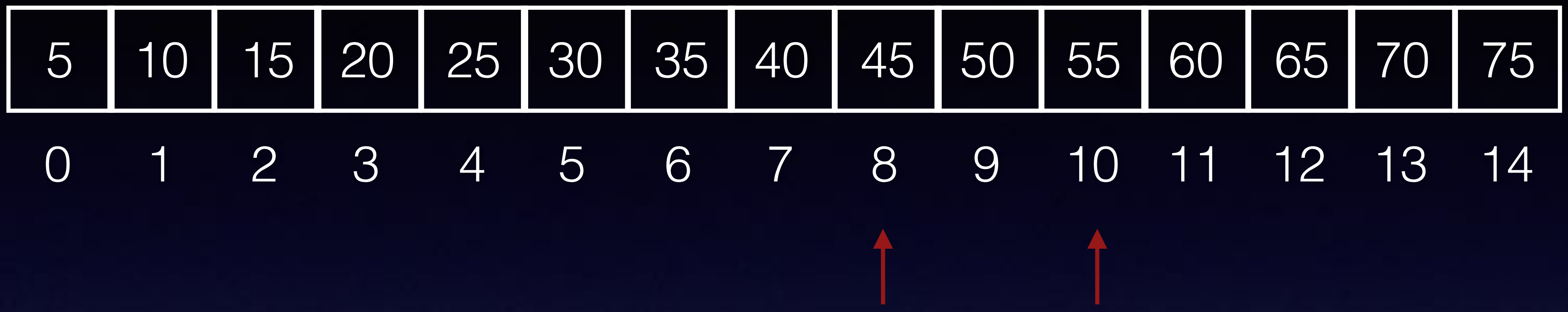
# Binary search with iteration



```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

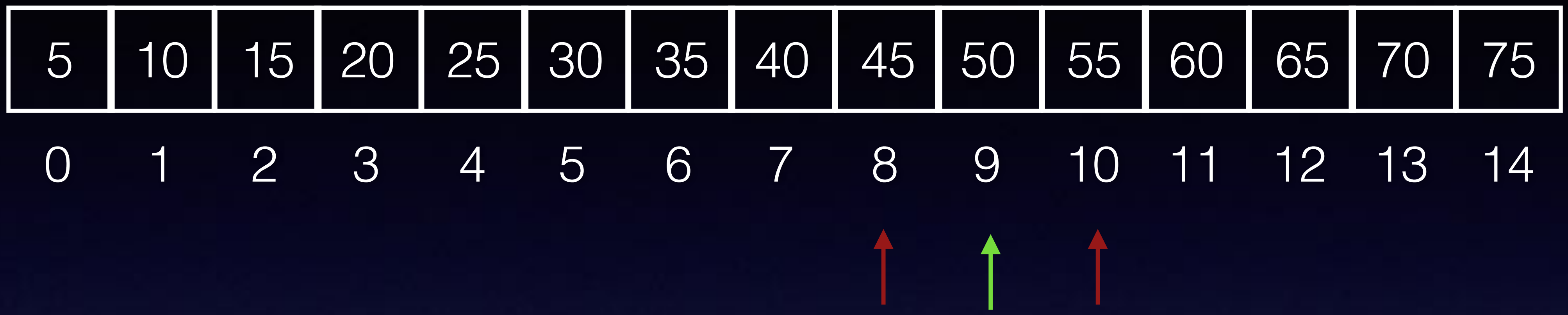
# Binary search with iteration



```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

# Binary search with iteration

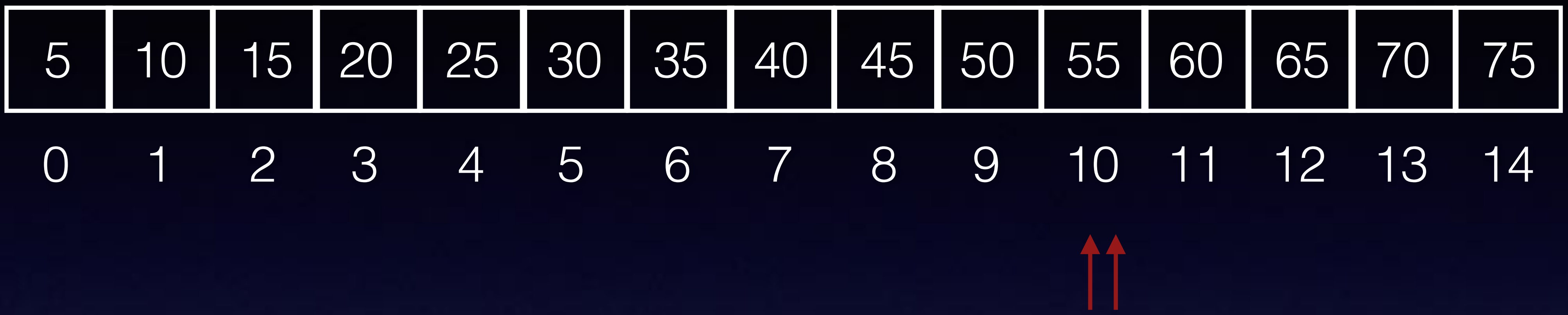


|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

# Binary search with iteration

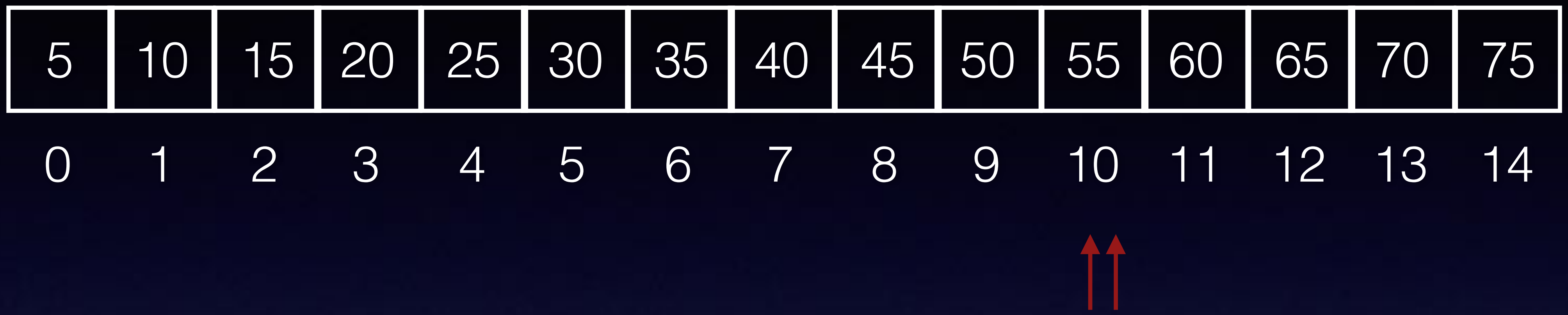


| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

# Binary search with iteration

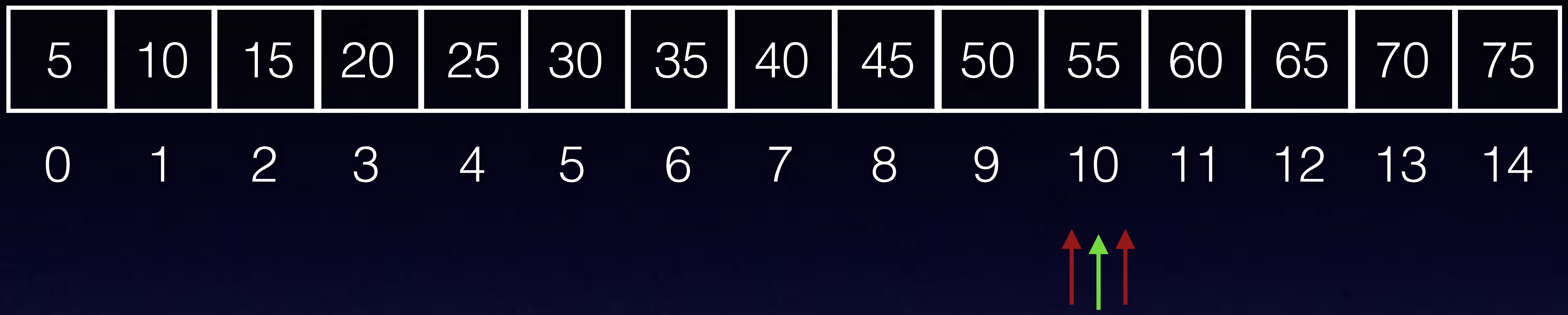


| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

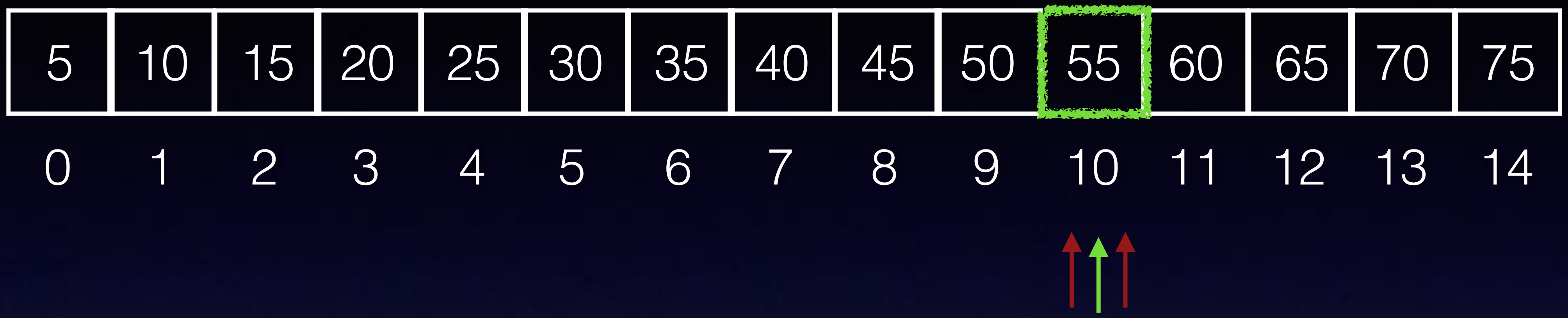
# Binary search with iteration



```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

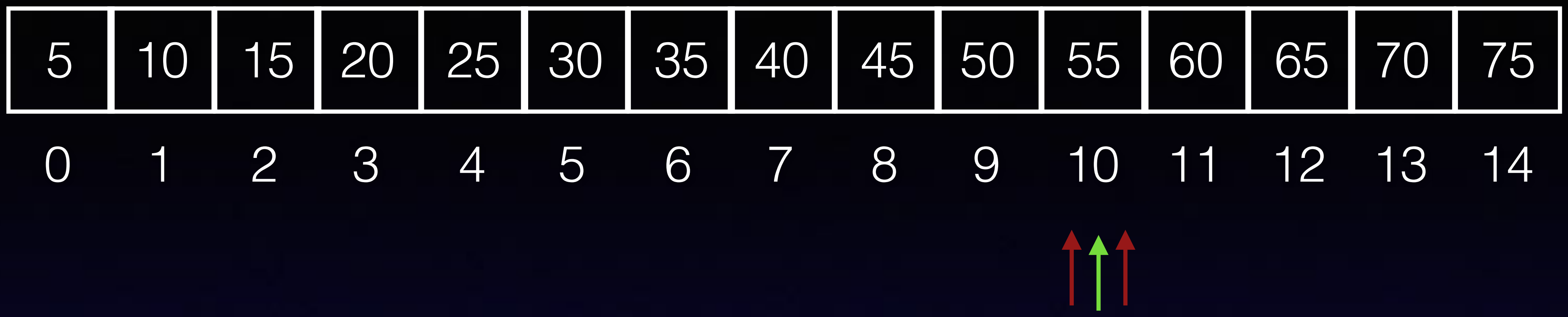
# Binary search with iteration



```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

```
>>> binarySearch(test2, 55)
```

# Binary search with iteration

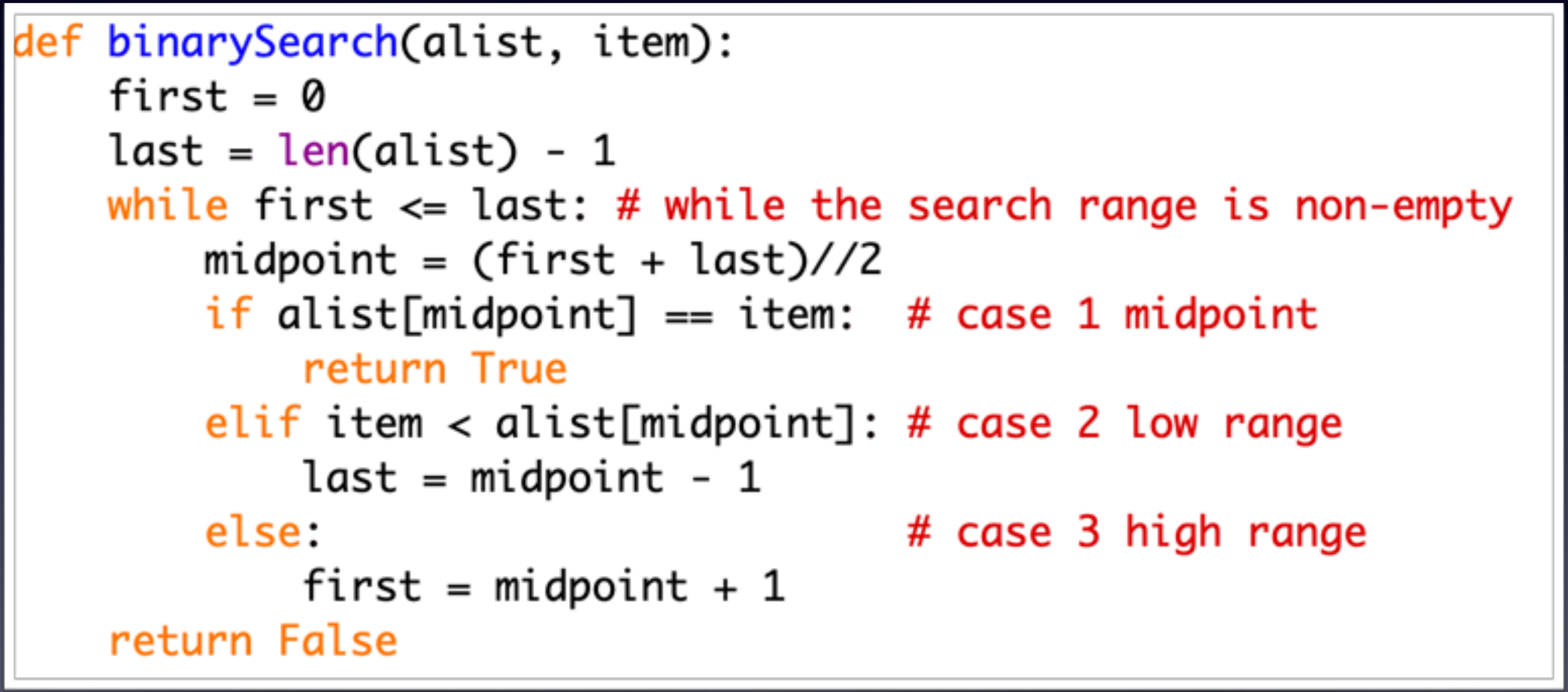


```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

What's the time complexity of binary search? $O(\log_2 n)$

# Binary search with iteration v2

Version 2 is shorter. It uses multiple return statements instead of the boolean variable found.



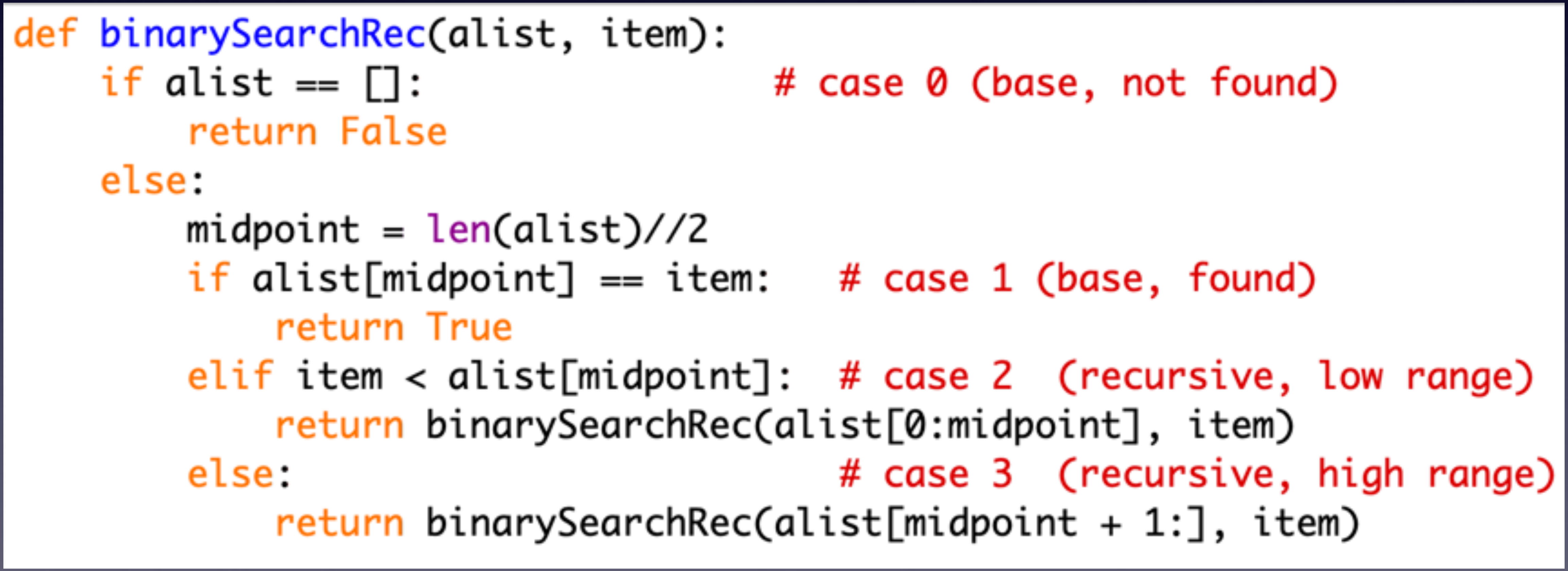
```python
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    while first <= last: # while the search range is non-empty
        midpoint = (first + last)//2
        if alist[midpoint] == item:  # case 1 midpoint
            return True
        elif item < alist[midpoint]: # case 2 low range
            last = midpoint - 1
        else:                        # case 3 high range
            first = midpoint + 1
    return False
```

The nested if is also replaced with an if-elif-else.

Convince yourself it implements exactly the same algorithm.

# Binary search with recursion

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |



```
def binarySearchRec(alist, item):
    if alist == []:                    # case 0 (base, not found)
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:    # case 1 (base, found)
            return True
        elif item < alist[midpoint]:   # case 2  (recursive, low range)
            return binarySearchRec(alist[0:midpoint], item)
        else:                          # case 3  (recursive, high range)
            return binarySearchRec(alist[midpoint + 1:], item)
```

Notice that calculating the midpoint is different. Why?

# Calculating the midpoint

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

For the list above, Python integer division to find the midpoint gives us:

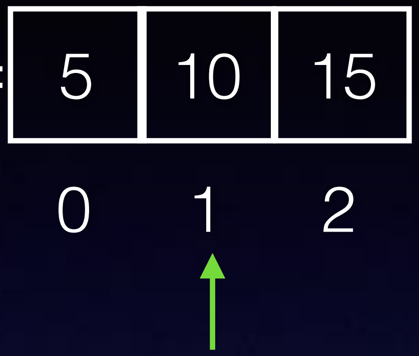
length(alist) // 2

15 // 2

7

# Calculating the midpoint

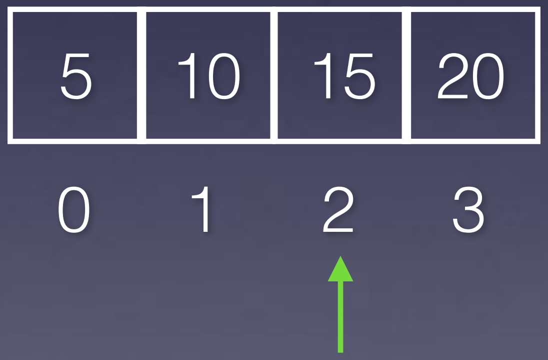
Small example: alist=



```
>>len(alist) // 2
1
```

same as the iterative version!

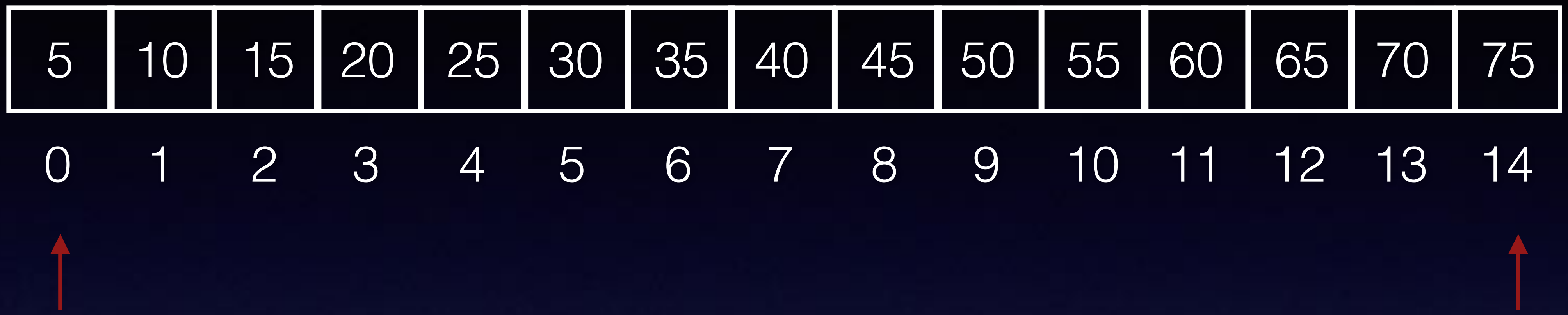
Little bigger example: alist=



```
>>len(alist) // 2
2
```

different from the iterative version!

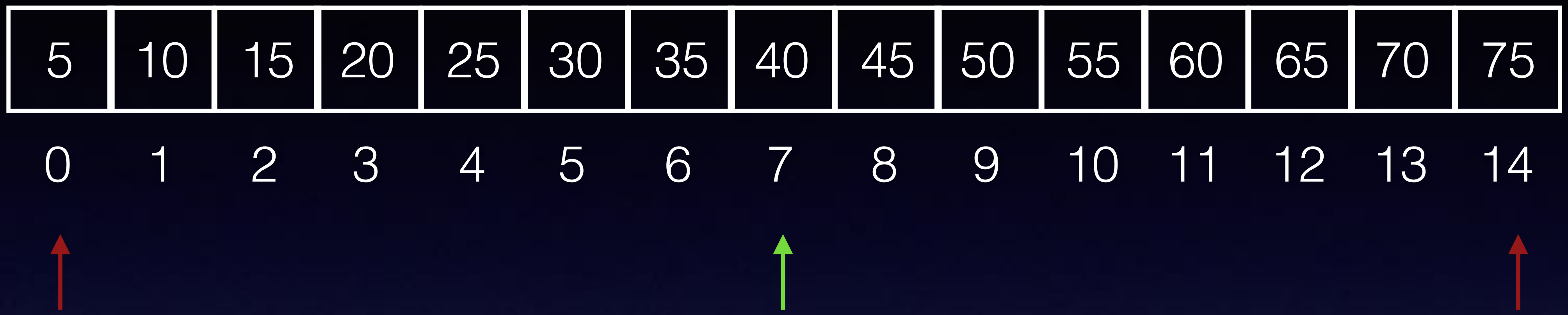
# Binary search with recursion



```
def binarySearchRec(alist, item):
    if alist == []:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:
            return True
        elif item < alist[midpoint]:
            return binarySearchRec(alist[:midpoint], item)
        else:
            return binarySearchRec(alist[midpoint + 1:], item)
```

```
>>> binarySearchRec([5, 10, ..., 70, 75], 55)
```

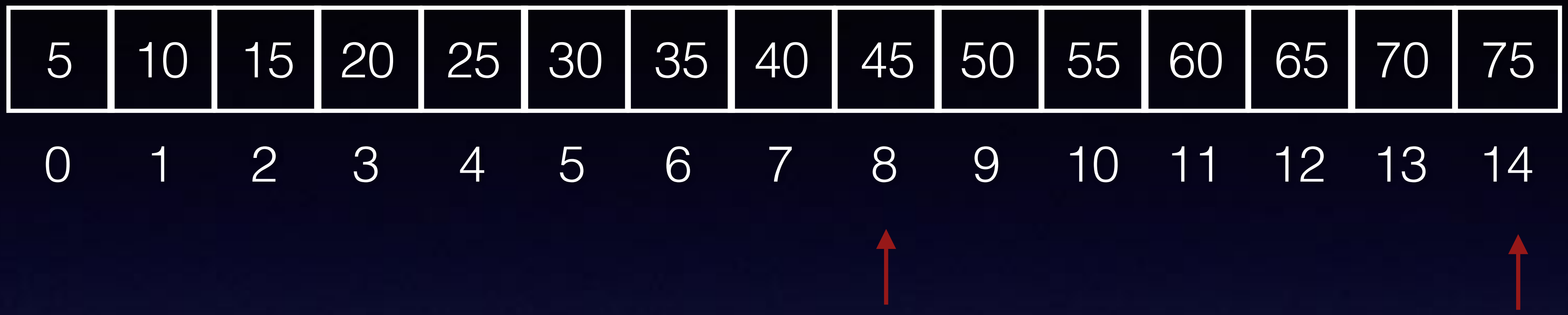
# Binary search with recursion



```
def binarySearchRec(alist, item):
    if alist == []:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:
            return True
        elif item < alist[midpoint]:
            return binarySearchRec(alist[:midpoint], item)
        else:
            return binarySearchRec(alist[midpoint + 1:], item)
```

```
>>> binarySearchRec([5, 10, ..., 70, 75], 55)
```

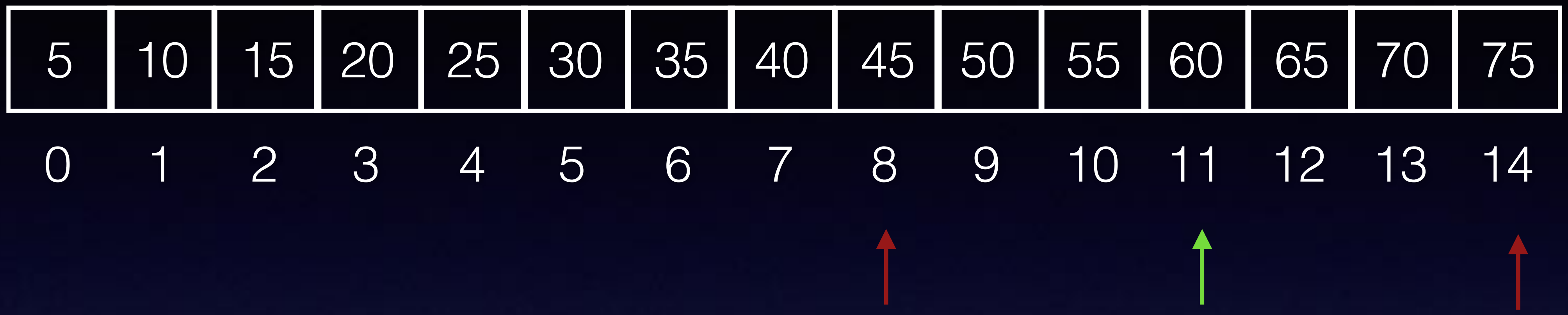
# Binary search with recursion



```
def binarySearchRec(alist, item):
    if alist == []:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:
            return True
        elif item < alist[midpoint]:
            return binarySearchRec(alist[:midpoint], item)
        else:
            return binarySearchRec(alist[midpoint + 1:], item)
```

```
>>> binarySearchRec([5, 10, ..., 70, 75], 55)
     binarySearchRec([45, 50, ..., 70, 75], 55)
```

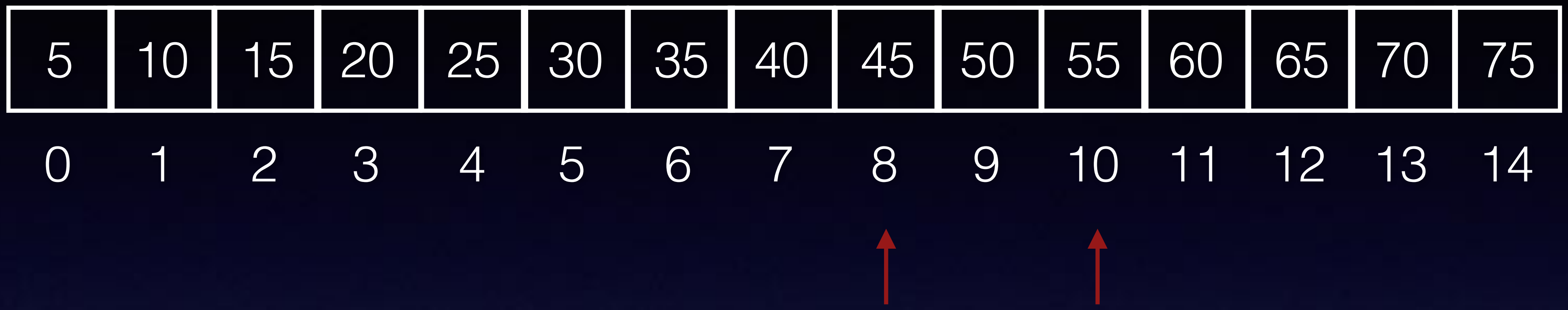
# Binary search with recursion



```
def binarySearchRec(alist, item):
    if alist == []:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:
            return True
        elif item < alist[midpoint]:
            return binarySearchRec(alist[:midpoint], item)
        else:
            return binarySearchRec(alist[midpoint + 1:], item)
```

```
>>> binarySearchRec([5, 10, ..., 70, 75], 55)
     binarySearchRec([45, 50, ..., 70, 75], 55)
```

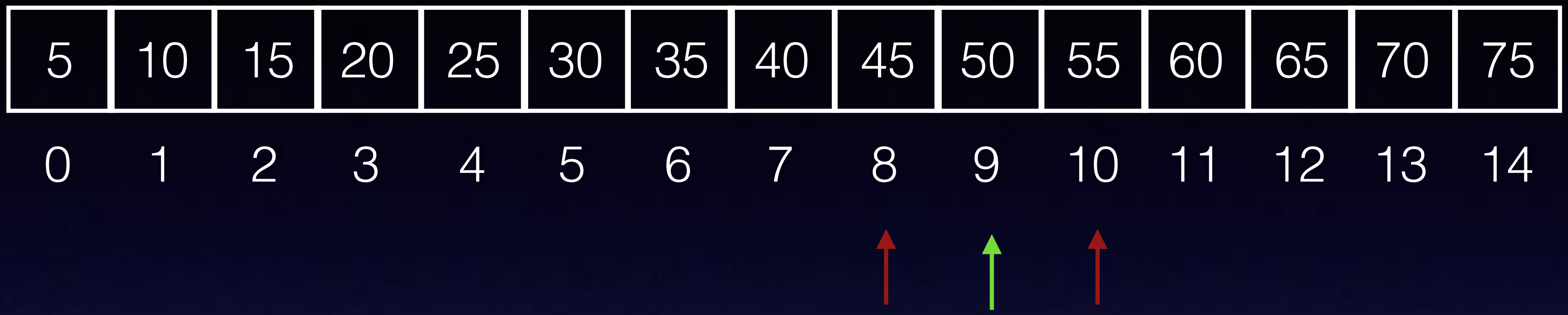
# Binary search with recursion



```python
def binarySearchRec(alist, item):
    if alist == []:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:
            return True
        elif item < alist[midpoint]:
            return binarySearchRec(alist[:midpoint], item)
        else:
            return binarySearchRec(alist[midpoint + 1:], item)
```

```
>>> binarySearchRec([5, 10, ..., 70, 75], 55)
     binarySearchRec([45, 50, ..., 70, 75], 55)
      binarySearchRec([45, 50, 55], 55)
```

# Binary search with recursion

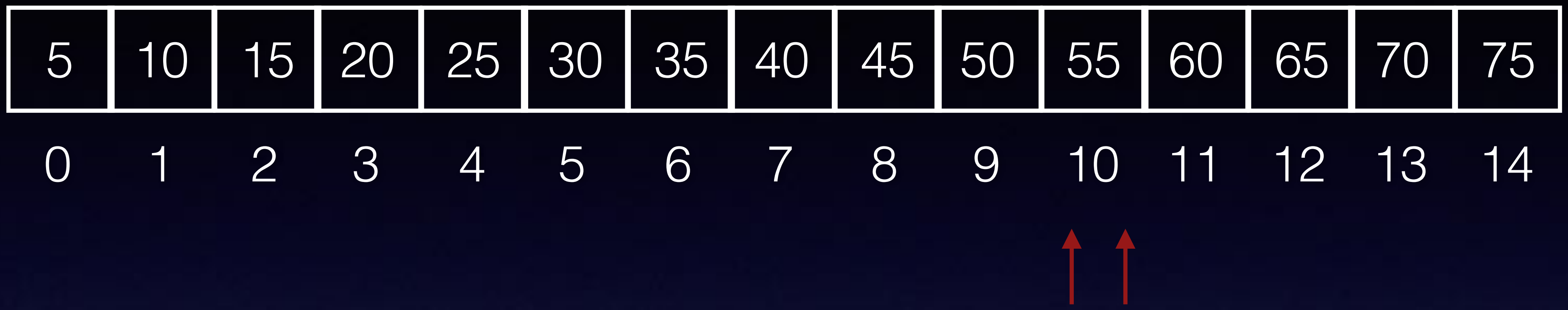


|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

```
def binarySearchRec(alist, item):
    if alist == []:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:
            return True
        elif item < alist[midpoint]:
            return binarySearchRec(alist[:midpoint], item)
        else:
            return binarySearchRec(alist[midpoint + 1:], item)
```

```
>>> binarySearchRec([5, 10, ..., 70, 75], 55)
     binarySearchRec([45, 50, ..., 70, 75], 55)
      binarySearchRec([45, 50, 55], 55)
```

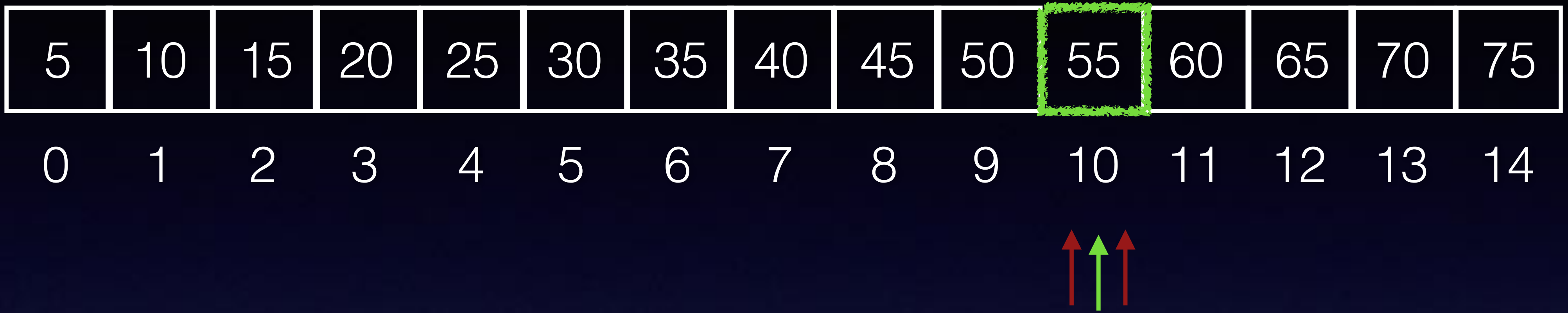
# Binary search with recursion



```
def binarySearchRec(alist, item):
    if alist == []:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:
            return True
        elif item < alist[midpoint]:
            return binarySearchRec(alist[:midpoint], item)
        else:
            return binarySearchRec(alist[midpoint + 1:], item)
```

```
>>> binarySearchRec([5, 10, ..., 70, 75], 55)
     binarySearchRec([45, 50, ..., 70, 75], 55)
      binarySearchRec([45, 50, 55], 55)
```

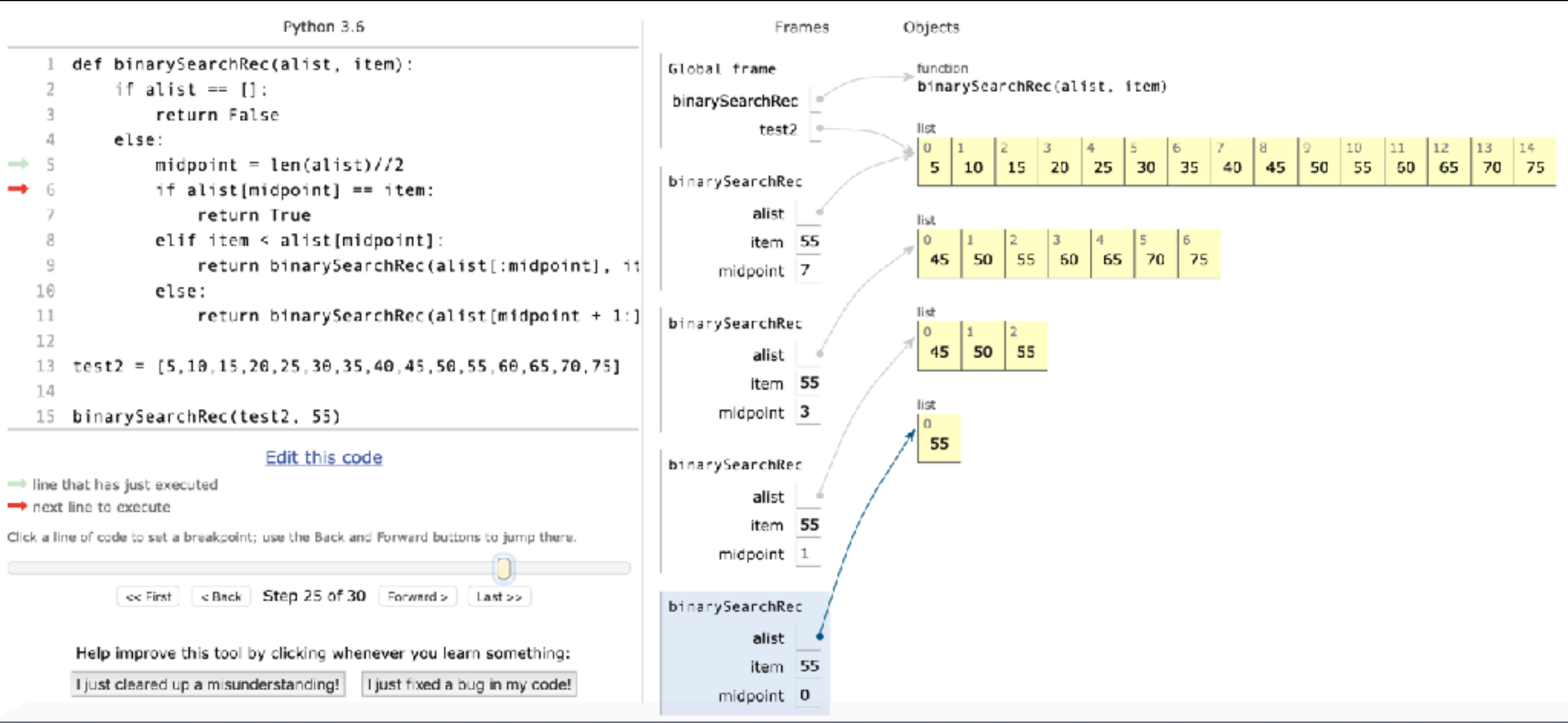
# Binary search with recursion



```
def binarySearchRec(alist, item):
    if alist == []:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:
            return True
        elif item < alist[midpoint]:
            return binarySearchRec(alist[:midpoint], item)
        else:
            return binarySearchRec(alist[midpoint + 1:], item)
```

```
>>> binarySearchRec([5, 10, ..., 70, 75], 55)
     binarySearchRec([45, 50, ..., 70, 75], 55)
      binarySearchRec([45, 50, 55], 55)
       binarySearchRec([55], 55)
```

# Python Tutor



Python 3.6

```
def binarySearchRec(alist, item):
    if alist == []:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:
            return True
        elif item < alist[midpoint]:
            return binarySearchRec(alist[:midpoint], it
        else:
            return binarySearchRec(alist[midpoint + 1:]

test2 = [5,10,15,20,25,30,35,40,45,50,55,60,65,70,75]

binarySearchRec(test2, 55)
```

Edit this code

line that has just executed

next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

<< First | < Back | Step 25 of 30 | Forward > | Last >>

Help improve this tool by clicking whenever you learn something:

I just cleared up a misunderstanding! | I just fixed a bug in my code!

Frames | Objects

```
Global frame
  binarySearchRec
  test2

binarySearchRec
  alist
  item      55
  midpoint  7

binarySearchRec
  alist
  item      55
  midpoint  3

binarySearchRec
  alist
  item      55
  midpoint  1

binarySearchRec
  alist
  item      55
  midpoint  0
```

function
binarySearchRec(alist, item)

list

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 |

list

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 45 | 50 | 55 | 60 | 65 | 70 | 75 |

list

| 0 | 1 | 2 |
|---|---|---|
| 45 | 50 | 55 |

list

| 0 |
|---|
| 55 |

Note the extra lists needed.

# Which is better?

Recursive

```
def binarySearchRec(alist, item):
    if alist == []:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:
            return True
        elif item < alist[midpoint]:
            return binarySearchRec(alist[:midpoint], item)
        else:
            return binarySearchRec(alist[midpoint + 1:], item)
```

# Which is better?

## Iterative

```
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False
    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```

## Which is better?

For binary search on an ordered list, a recursive search algorithm is arguably a more simple and elegant solution than the loop-based approach.

The recursive approach uses up extra space.

# Recursion vs. Iteration

Both involve the repetition of a sequence of statements.

An iterative solution exists for any problem solvable by recursion.

An iterative solution may be more efficient.

A recursive solution is sometimes easier to understand.

## Recursion vs. Iteration

Although the iterative and recursive solutions to binary search in a sorted list achieve the same result with roughly the same number of steps, there is technically more overhead for the recursive solution.

This difference is small, however.  Generally, if it is easier to conceptualize a problem as recursive, it should be coded as such.

# Binary search with recursion

```
def binarySearchRec(alist, item):
    if alist == []:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint] == item:
            return True
        elif item < alist[midpoint]:
            return binarySearchRec(alist[:midpoint], item)
        else:
            return binarySearchRec(alist[midpoint + 1:], item)
```

What about the slices? Again, there's a cost when creating a new list. We could get rid of that. Your book suggests passing start and end indexes along with the list. We leave this as a homework assignment.

# Binary Search Quiz

Suppose you have the following sorted list [3, 5, 6, 8, 11, 12, 14, 15, 17, 18] and are using the **recursive binary search algorithm**. Which group of numbers correctly shows the sequence of comparisons used to find the key 8.

(A) 11, 5, 6, 8
(B) 12, 6, 11, 8
(C) 3, 5, 6, 8
(D) 18, 12, 6, 8

# Binary Search Quiz

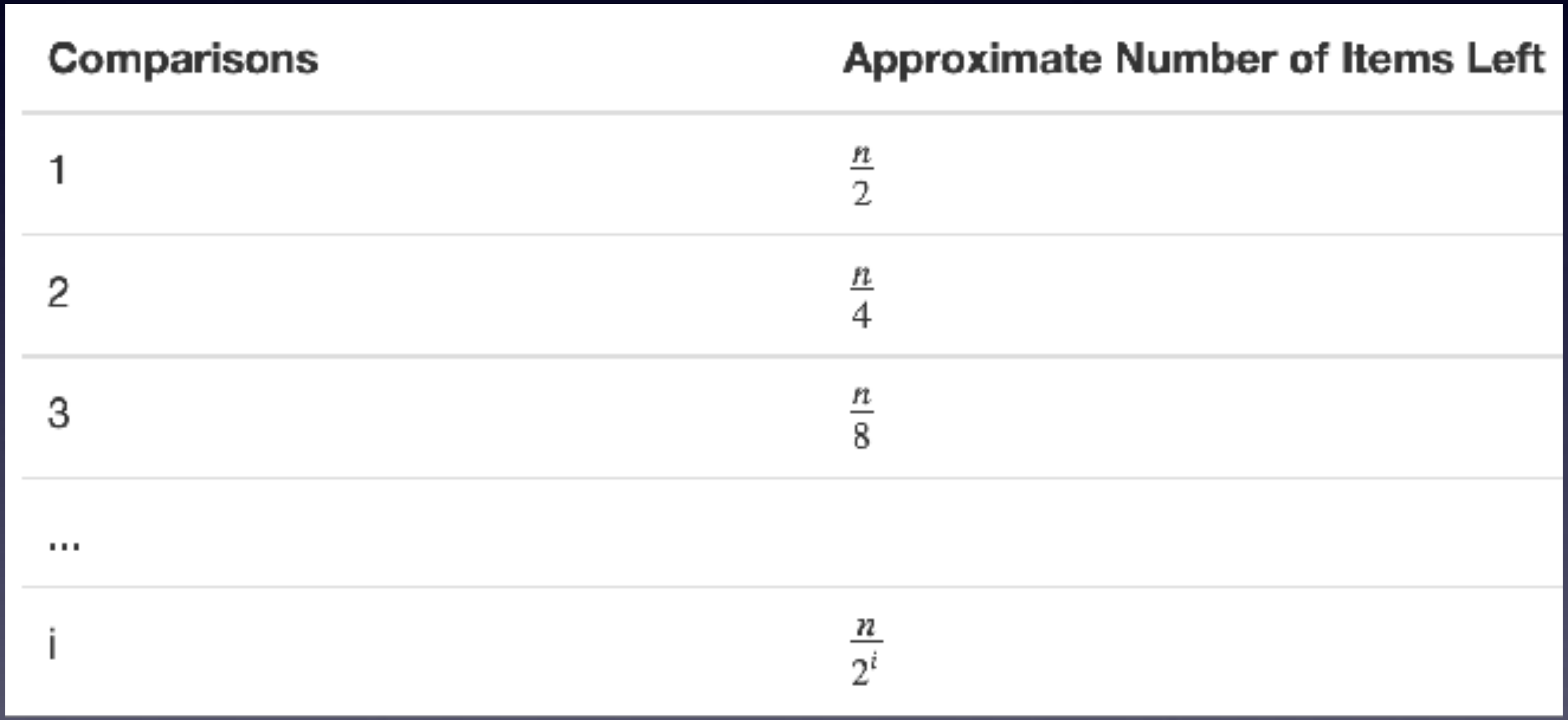
Suppose you have the following sorted list [3, 5, 6, 8, 11, 12, 14, 15, 17, 18] and are using the recursive binary search algorithm. Which group of numbers correctly shows the sequence of comparisons used to find the key 8.

(A) 11, 5, 6, 8
**(B) 12, 6, 11, 8**
(C) 3, 5, 6, 8
(D) 18, 12, 6, 8

What about if using iterative binary search?

# Binary Search Analysis

Number of items left to search

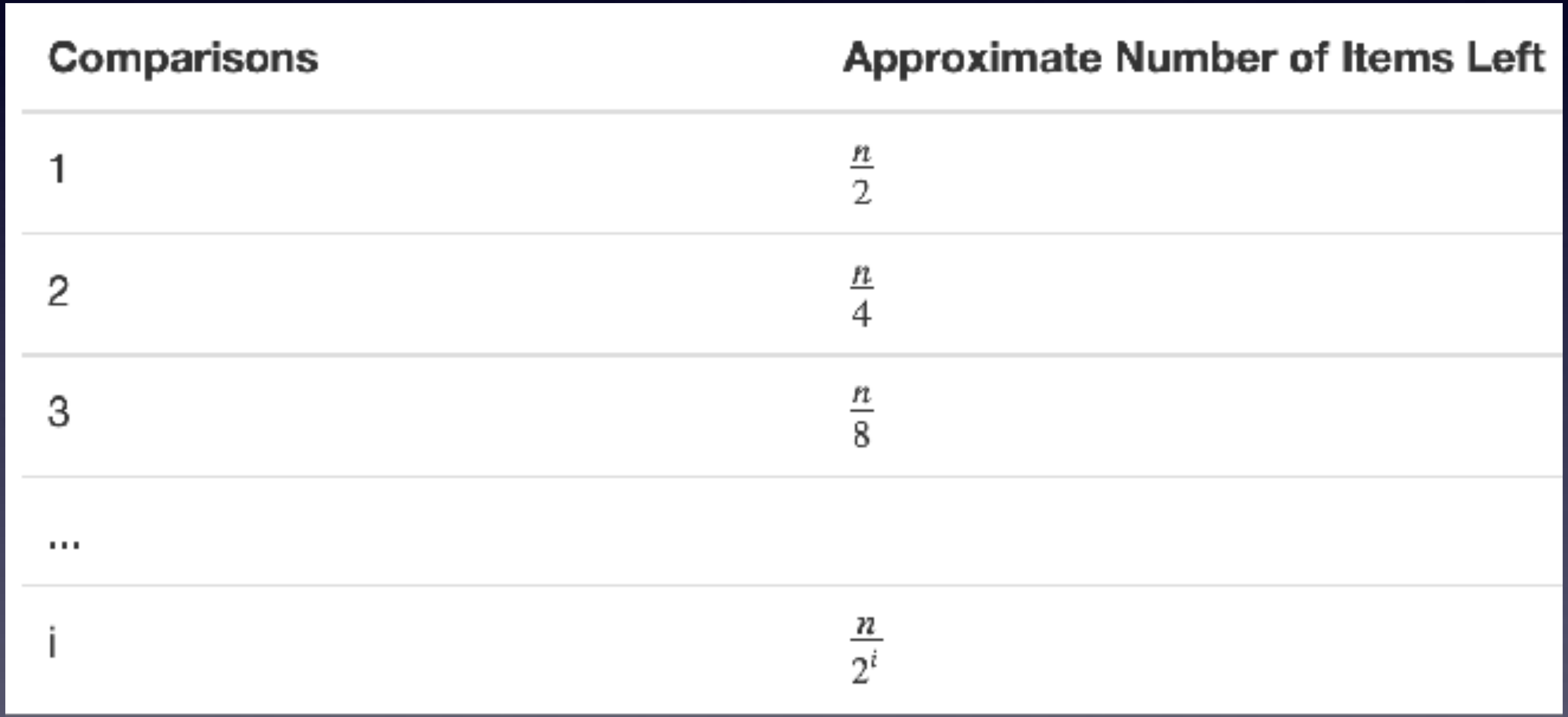


| Comparisons | Approximate Number of Items Left |
|---|---|
| 1 | $\frac{n}{2}$ |
| 2 | $\frac{n}{4}$ |
| 3 | $\frac{n}{8}$ |
| ... | |
| i | $\frac{n}{2^i}$ |

We are done when items left to search is 1

# Binary Search Analysis

Number of items left to search



| Comparisons | Approximate Number of Items Left |
|---|---|
| 1 | $\frac{n}{2}$ |
| 2 | $\frac{n}{4}$ |
| 3 | $\frac{n}{8}$ |
| ... | |
| i | $\frac{n}{2^i}$ |

That gives us this equality, where i is the number of comparisons.

$$\frac{n}{2^i} = 1$$

# Binary Search Analysis

Solving for i in terms of n:

$$\frac{n}{2^i} = 1$$

$$n = 2^i$$

$$log_2(n) = log_2(2^i)$$

$$log_2(n) = i$$

$$number\ of\ comparisons = log_2(n)$$