Ulster University

COM410 Programming in Practice

B3.2 Binary Search

Binary Search of a Sorted Array

- A **binary search** of an array rules out whole sections of the array at each comparison step because the array is sorted

- Binary search of a sorted array, where the desired item (target) = 8
- Repeatedly find the mid point of the array and determine if target is in each half
- In this case the search finds the target

Look at the middle entry, 10:

2	4	5	7	8	**10**	12	15	18	21	24	26
0	1	2	3	4	5	6	7	8	9	10	11

$8 < 10$, so search the left half of the array.

Look at the middle entry, 5:

2	4	**5**	7	8
0	1	2	3	4

$8 > 5$, so search the right half of the array.

Look at the middle entry, 7:

7	8
3	4

$8 > 7$, so search the right half of the array.

Look at the middle entry, 8:

8
4

$8 = 8$, so the search ends. 8 is in the array.

Binary Search of a Sorted Array

- Another binary search of a sorted array, where the desired item (target) = 16

Look at the middle entry, 10:

2	4	5	7	8	**10**	12	15	18	21	24	26
0	1	2	3	4	5	6	7	8	9	10	11

$16 > 10$, so search the right half of the array.

Look at the middle entry, 18:

12	15	**18**	21	24	26
6	7	8	9	10	11

$16 < 18$, so search the left half of the array.

Look at the middle entry, 12:

12	15
6	7

$16 > 12$, so search the right half of the array.

Look at the middle entry, 15:

15
7

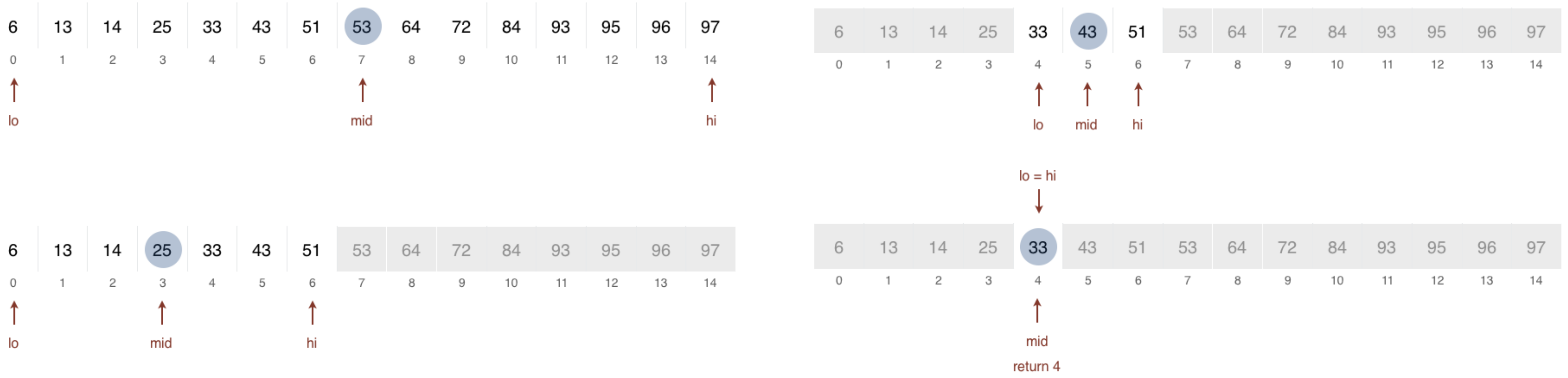
$16 > 15$, so search the right half of the array.

The next subarray is empty, so the search ends. 16 is not in the array.

- No more entries left to consider in the array, so search ends
- Search does not find the target (16 is not in the array)

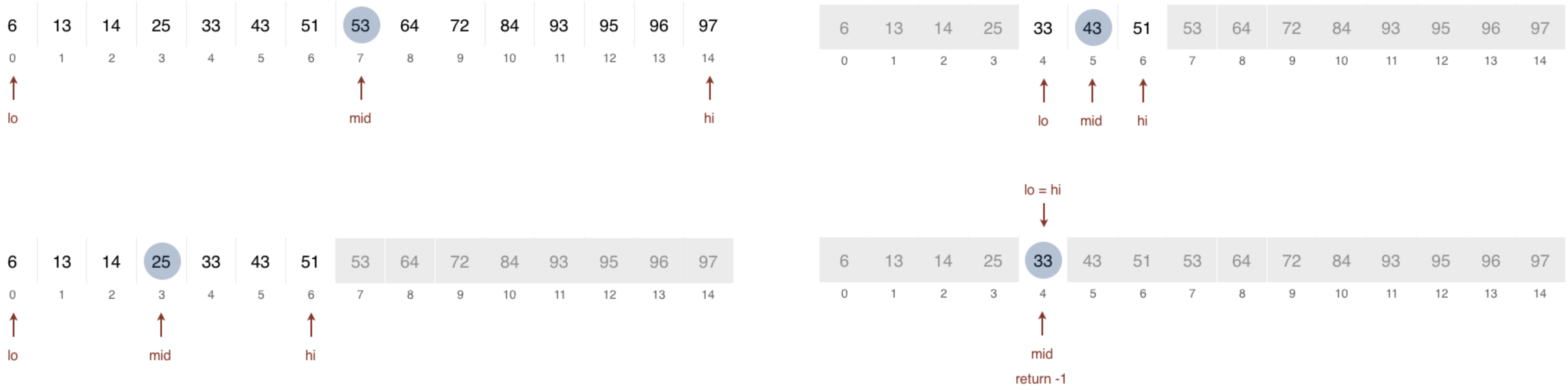
Binary Search of a Sorted Array

- Another example of a successful binary search for target = 33



Binary Search of a Sorted Array

- Another example of an unsuccessful binary search for target = 34



Recursive Binary Search of Sorted Array

- To search elements a[0] through a[n-1] you have to either search a[0] through a[mid-1] or search a[mid+1] through a[n-1]
- Two searches (of portion of the array) are smaller versions of the problem
- Pseudocode for the logic of the binary search:

```
Algorithm binarySearch(array, first, last, entry)
// Returns true if array from position first to position
// last contains element, false otherwise

Set mid to (first + last) / 2
if first > last return false
else if array[mid] == entry] return true
else if  entry < array[mid]
     return binarySearch (array, first, mid – 1, entry)
else return binarySearch (array, mid + 1, last, entry)
```

parameters used to specify the first and last indices of the subranges of the array to be searched

Efficiency of a Binary Search of an Array

- Search eliminates about half of the array from consideration after examining only one element, then another quarter, then an eighth, etc.
- **Best case**: desired item is in the first element checked, so search will be **O(1)**
- **Worst case**: search continues until one item left, splitting the array ***k*** times such that $2^k = n$. Since ***k*** (the number of comparisons) is $\log_2(n)$, search will therefore be **O(log n)**
- **Average case**: search will make one-half of the recursive calls, so will be **O(log n)**

Binary Search of a Sorted Chain?

- How will the mid point of the chain be found? (to find the middle of the chain you must traverse the whole chain to that point)
- Then you must traverse one of the halves to find the middle of that half – far too much work involved!
- It would be better to traverse the list once to build an array and then use the binary search on that.

Sequential Search vs. Binary Search

- You should use a sequential search to search a chain of linked nodes
- If you want to search an array of objects, you need to choose the appropriate technique

	Best Case	Average Case	Worst Case
Sequential search (unsorted data)	O(1)	$O(n)$	$O(n)$
Sequential search (sorted data)	O(1)	$O(n)$	$O(n)$
Binary search (sorted array)	O(1)	$O(\log n)$	$O(\log n)$

- If array is small, a sequential search can be faster (less computation required)
- If array is large and already sorted, a binary search is normally faster

Challenge

- In a new project called **Searching**, create the class `SearchTimeTest` to measure times of search algorithms for sorted arrays
    - Generate random arrays of ascending integers of size 1000, 2000, 4000, 8000, 16000, and 32000 elements. For each array the integers should be in the range 1 to the array size * 10 (e.g. 1K integers in the range up to 10K, 2K integers in the range up to 20K, etc.
    - For each array, generate an array of 1000 search values where the search values are from the same range as the array elements.
    - Measure the time taken by the iterative sequential and recursive binary searches to complete the search for all 1000 search values. Remember to optimize the sequential search for a sorted array.
    - Report the results as shown in the demonstration video.