Basic searching techniques

The algorithm used to search a list depends to a large extent on the structure of the list.

Two basic searches for arrays are:

1. Sequential search
2. Binary search

Sequential search (aka linear search)

- Is used in an **unordered** list

Steps:

1. Start from the leftmost element of the list and one by one compare the target with each element of the list
2. If the target matches with an element, return the index of the element
3. Otherwise, return -1 indicating that the target is not present in the list

Sequential search

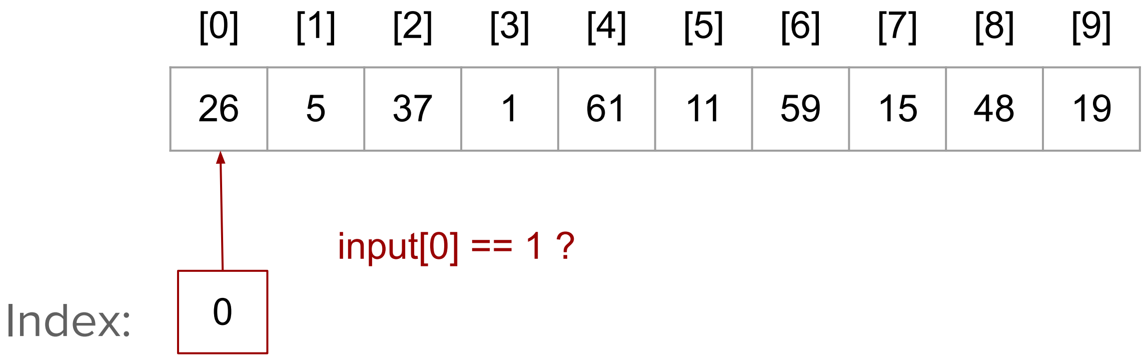
Example: Search for 1 in this unsorted list.

Input:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
26	5	37	1	61	11	59	15	48	19

Target: 1

Sequential search

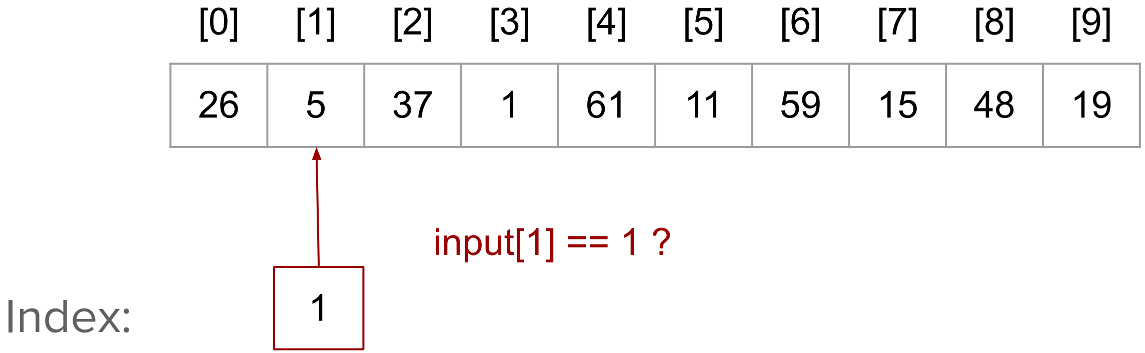


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
26	5	37	1	61	11	59	15	48	19

input[0] == 1 ?

Index: 0

Sequential search

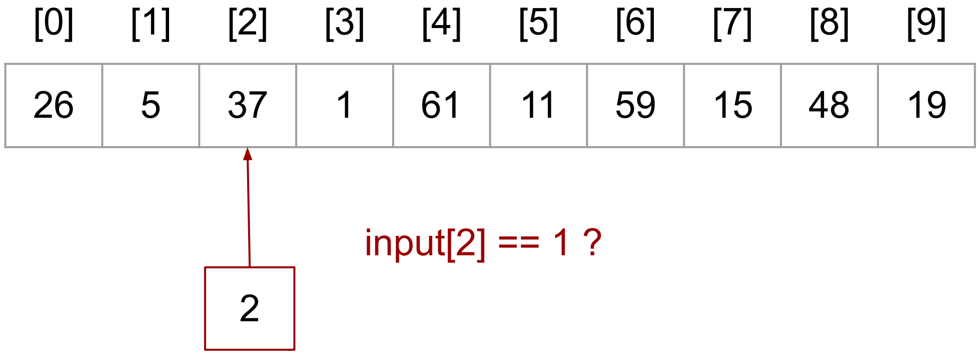


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
26	5	37	1	61	11	59	15	48	19

input[1] == 1 ?

Index: 1

Sequential search

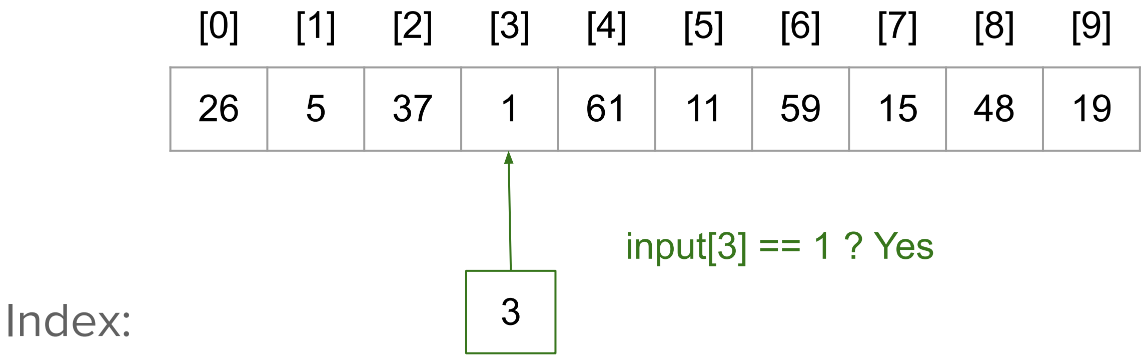


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
26	5	37	1	61	11	59	15	48	19

input[2] == 1 ?

Index: 2

Sequential search



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
26	5	37	1	61	11	59	15	48	19

input[3] == 1 ? Yes

Index: 3

Sequential search performance

Best case, i.e. when the target is the first element in the list:

O(1)

Worst case, i.e. when the target is not present in the list or is the last element of the list:

O(n)

Average case:

O(n)

Binary search

In sequential search, if there are 1000 elements, 1000 comparisons will be made in the worst case.

If the list is sorted, we can use a more efficient algorithm called the **binary search**.

In general, we should use a binary search whenever the list starts to become large (e.g., when the list has more than 16 elements).

Algorithm: binarySearch(a, target)

Input: A sorted list, a, and the element to be searched, target

Output: Index of the target, if present, otherwise -1

Steps:

```
min = 0
max = n - 1
while max ≥ min
    mid = $\lfloor (min + max ) / 2 \rfloor$ # average of max and min
    if a[mid] == target
        return mid  # target found
    else if a[mid] < target
        min = mid + 1
    else
        max = mid - 1
    end if
end while
if max < min, then return -1 # target is not present
end if
```

Binary search

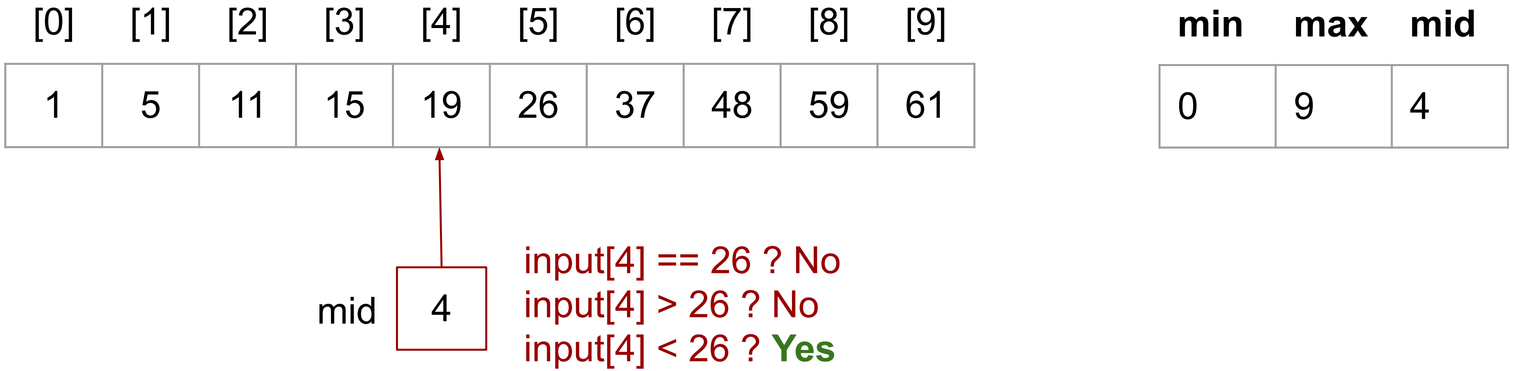
Example: Search for 26 in this list.

Input:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	5	11	15	19	26	37	48	59	61

Target: 26

Binary search



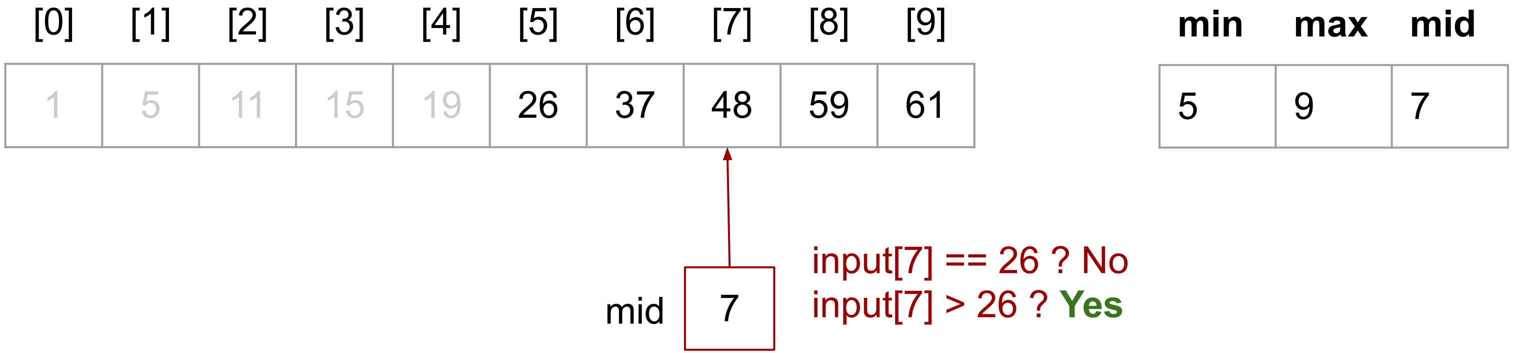
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	5	11	15	19	26	37	48	59	61

min	max	mid
0	9	4

mid 4

input[4] == 26 ? No
input[4] > 26 ? No
input[4] < 26 ? **Yes**

Binary search



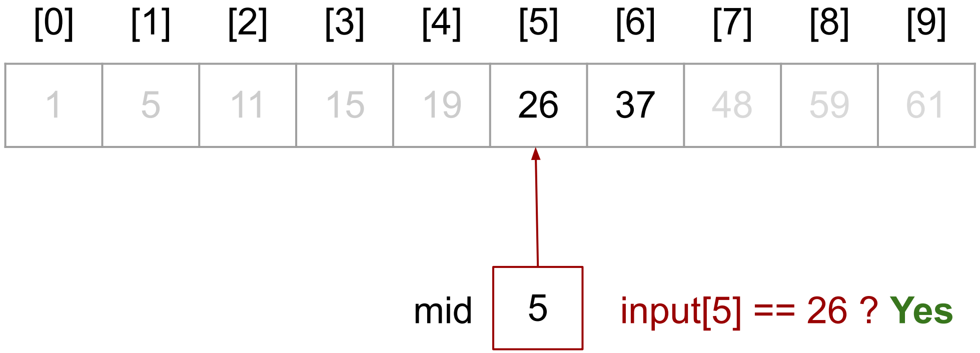
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	5	11	15	19	26	37	48	59	61

min	max	mid
5	9	7

mid 7

input[7] == 26 ? No
input[7] > 26 ? **Yes**

Binary search



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	5	11	15	19	26	37	48	59	61

mid 5

input[5] == 26 ? **Yes**

min	**max**	**mid**
5	6	5

Target found!

Binary search performance

Best case: $O(1)$

Worst case: $O(\log_2 n)$

Average case: $O(\log_2 n)$