</talentlabs>

Lecture 4
Searching Algorithms

</talentlabs>

Agenga

- Common Algorithms
- Searching
- Linear Search
- Binary Search

Common Algorithms

</talentlabs>

Common Algorithms

- In interviews, you don’t always build your own algorithms. Sometimes you only need to memorize or leverage well-known algorithms
- Usually these well-known algorithms are either Searching or Sorting algorithms

Sample Interview Questions:

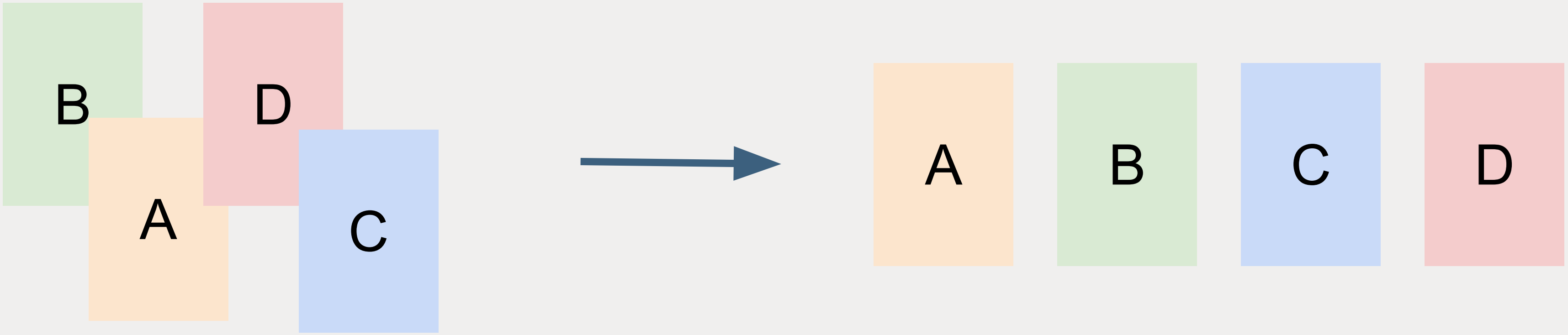
- Can you name and describe 2 sorting algorithms to me?
- Can you describe “Binary Search” algorithm to me?

Searching Algorithm

- Common algorithms on searching for a particular elements in an array.
- There are 2 common types of search algorithms - Linear Search and Binary Search.

Sorting Algorithms

- These algorithms are to solve the problem of “given an array, how do you sort the array in shortest time?”
- These is a long list of sorting algorithms, some of the common ones include “Bubble Sort”, “Merge Sort”, “Quick Sort” and more.



Searching Algorithm 1 - Linear Search

</talentlabs>

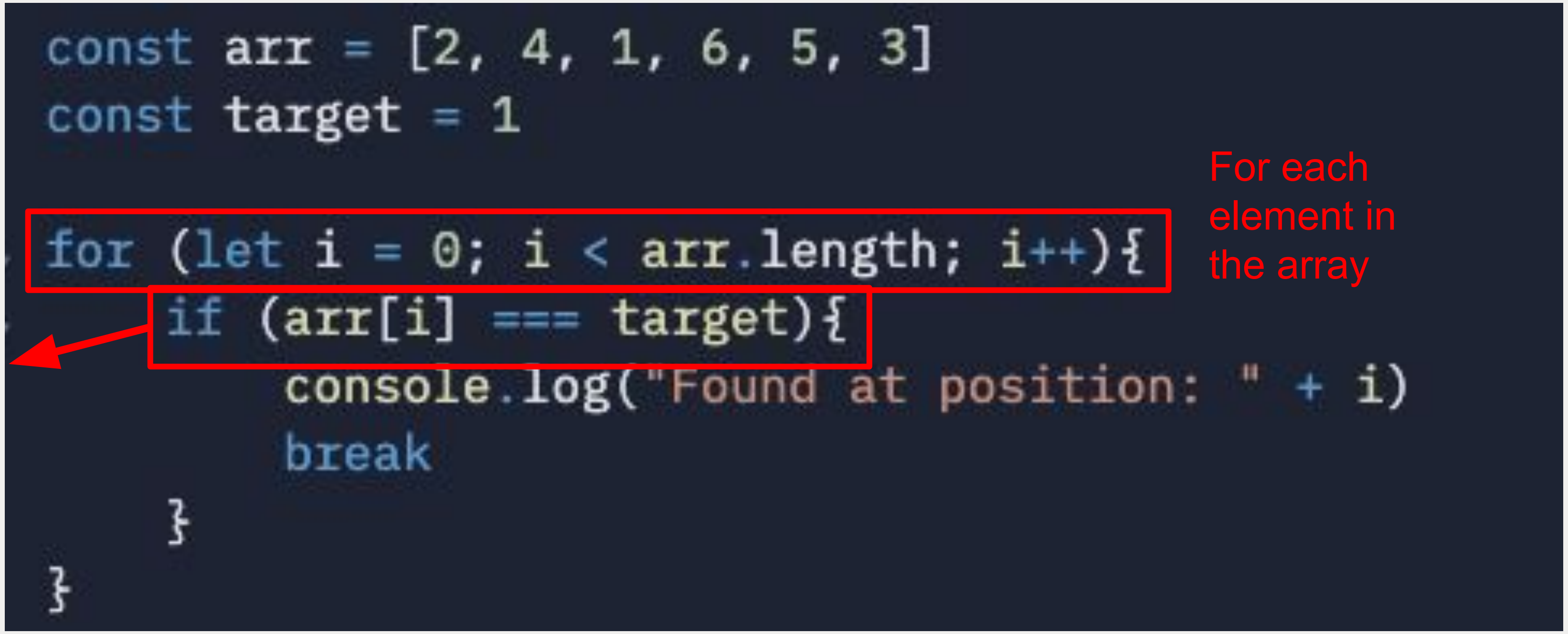
Linear Search

- You already know Linear Search!
- Linear search means checking the elements one by one, from the start to the end, until the target is found

Linear Search Algorithm

1. Use a for loop to loop over the array.
2. For each element, check against the target.
    a. If there is a match, return true or the position.
    b. If it’s not a match, go for the next item
3. If still not found after going through the whole array, return false or -1

Converting into Code



```
const arr = [2, 4, 1, 6, 5, 3]
const target = 1

for (let i = 0; i < arr.length; i++){
    if (arr[i] === target){
        console.log("Found at position: " + i)
        break
    }
}
```

For each element in the array

Check if the element equals to the target

Searching Algorithm 2 - Binary Search

Is there a faster way of searching?

- Linear search is very effective, but the worse case is that we need to search through whole array one by one.
- What if the target is at the end of the array, then we need to go through the whole array.
- Imagine the array length is 10000 instead of 10:
  - How do we make the search quicker and more efficient?
  - Is there a way that we don’t need to go through the whole list?

Michelle:

```
{
  a: [apple, app, astronaut, add],
  b: [ball, boy, baby, basket]
}
```

Ariff:

```
{
  “smallerthan40”: [1, 30, 25]
  “smallthan80”: [50, 41, 60]

}
```

Id (Primary Key)	**Name**	**Classes (Index)**
1	Darren	Python
2	Michelle	JS
3	Thas	Python
4	Ariff	JS

Index of **Classes Column**:

```
{
  “Python”: [1, 3],
  “JS”: [2, 4]
}
```

```
select * from students_table where classes = “JS” order by id
```

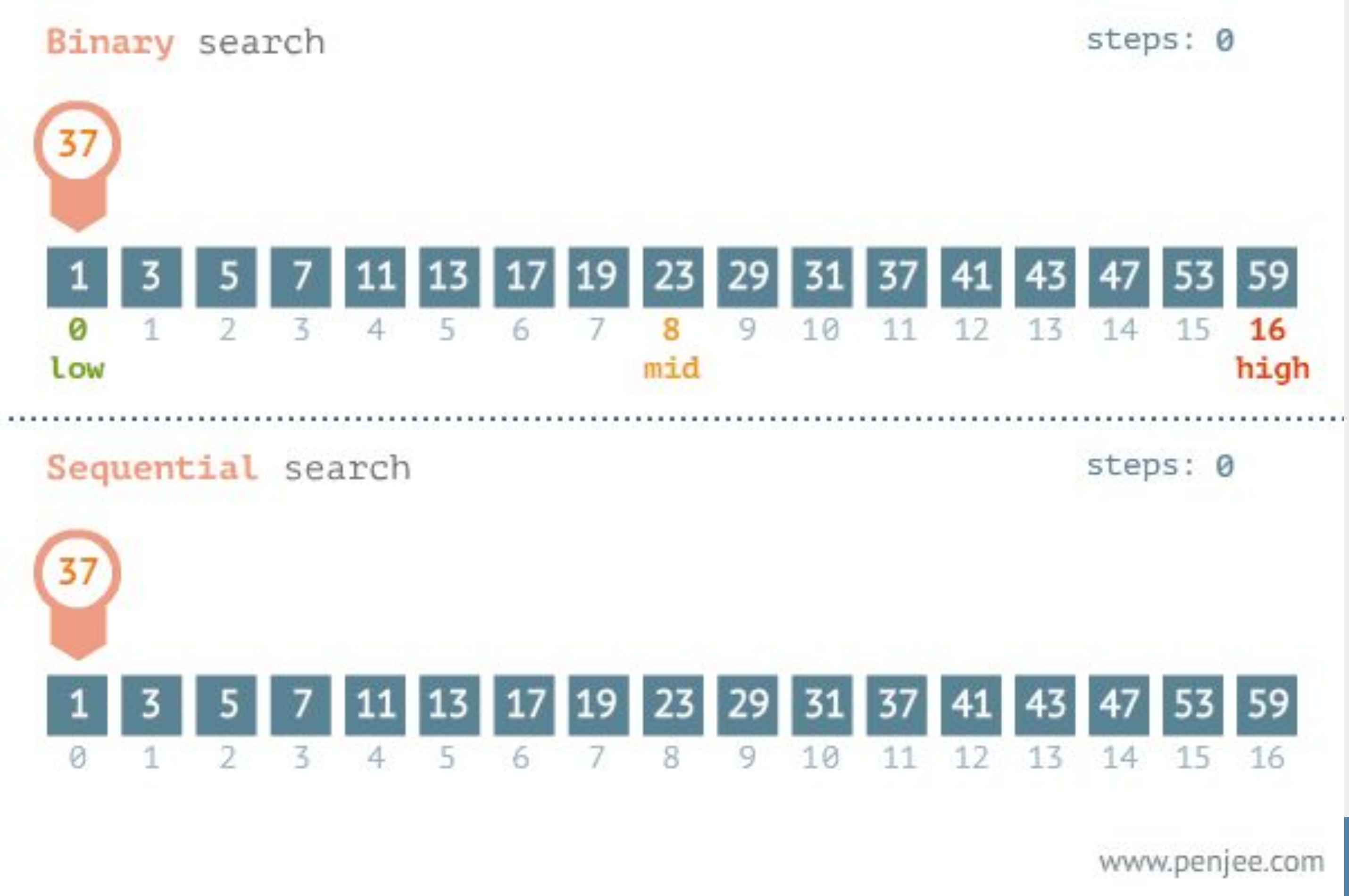
Consider this scenario

- What the numbers are sorted first?

Question Statement becomes:
How to find an element in a sorted array?

Example:
Find “4” in [1, 2, 4, 5, 8, 10, 12, 19]

Consider this scenario



Binary search

steps: 0

37

1	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
low								mid								high

Sequential search

steps: 0

37

1	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

www.penjee.com

Source: https://blog.penjee.com/wp-content/uploads/2015/04/binary-and-linear-search-animations.gif

Binary Search

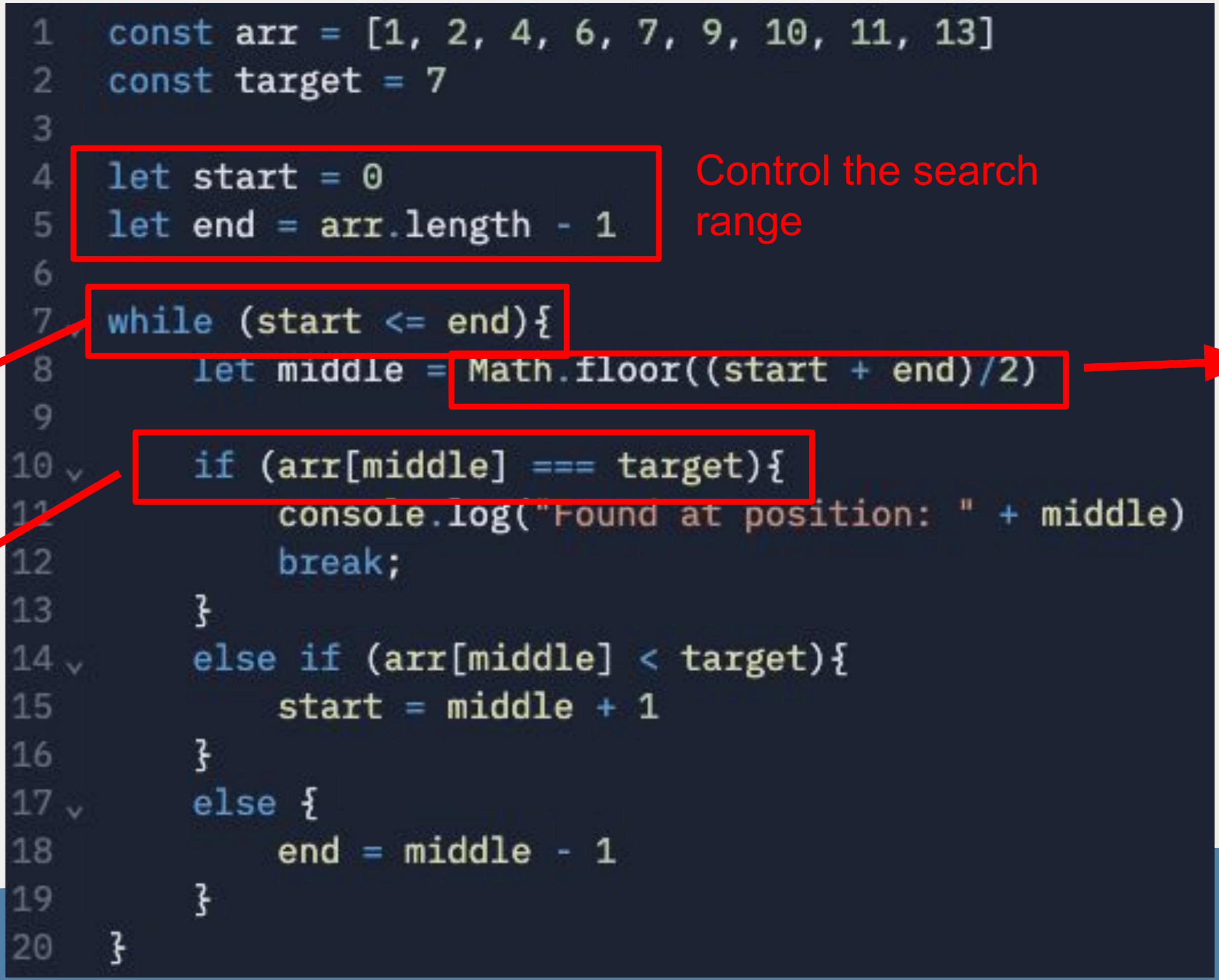
1. Start with the element at the middle position
    a. if the element at the middle equals to the target, then return true
    b. if the element at the middle is larger than the target, shrink the search range to smaller half
    c. if the element at the middle is smaller than the target, shrink the search range to the larger half
2. Repeat step 1 until target is found or the range contains only 1 element

Input Array: [1, 2, 4, 6, 7, 9, 10, 11, 13]
Target: 3

Start: 0
End: 8

</talentlabs>

Converting into Code



```
const arr = [1, 2, 4, 6, 7, 9, 10, 11, 13]
const target = 7

let start = 0
let end = arr.length - 1

while (start <= end){
    let middle = Math.floor((start + end)/2)

    if (arr[middle] === target){
        console.log("Found at position: " + middle)
        break;
    }
    else if (arr[middle] < target){
        start = middle + 1
    }
    else {
        end = middle - 1
    }
}
```

Control the search range

Handle the odd length scenario

Exit Condition: start is larger than the end

Found!