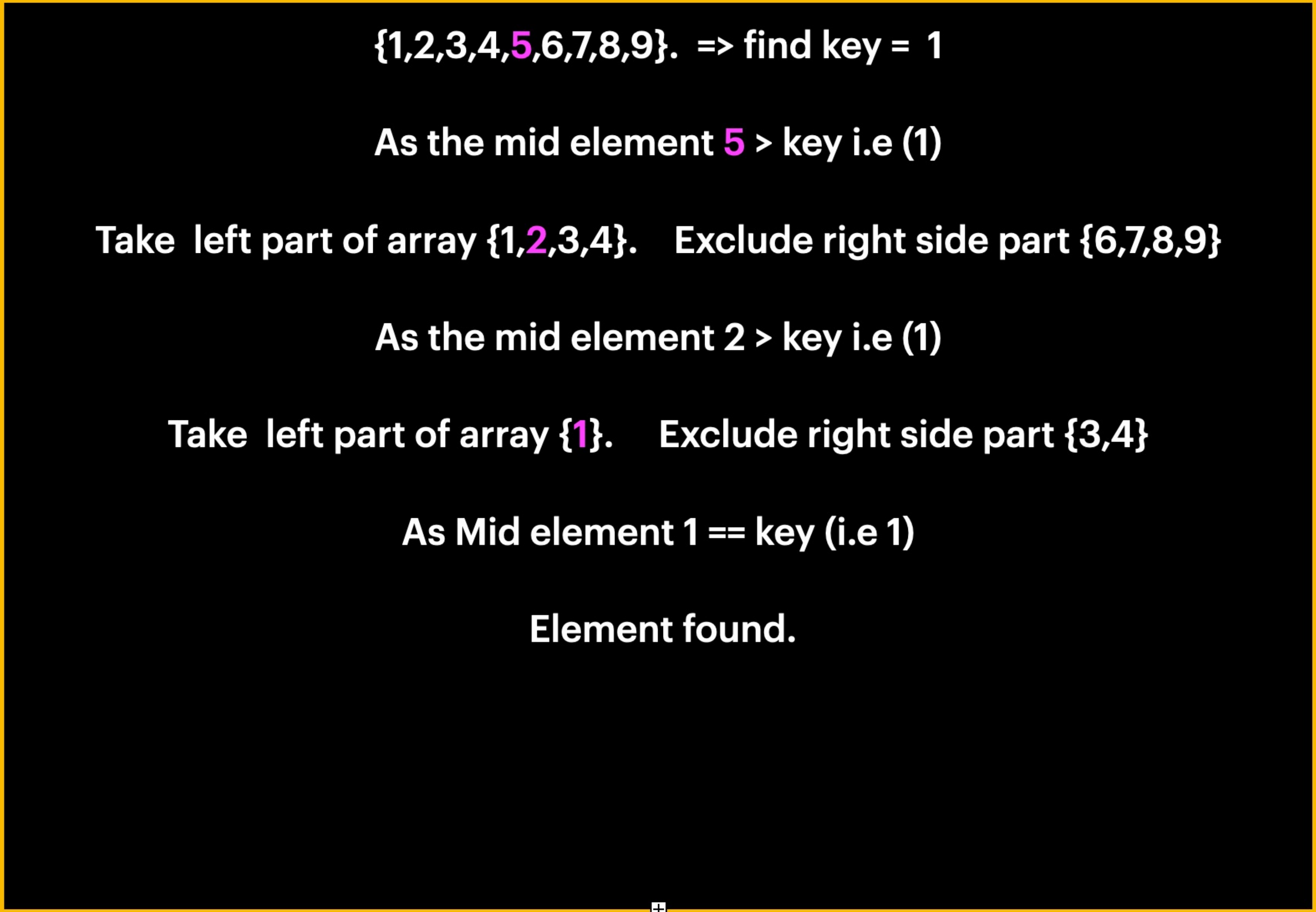
Why Sorting ?

Usually for searching an element in unsorted array takes linear time i.e O(n).

If we sort the array, we can apply the binary search on elements so that searching would be done within O(logn).

That’s a great improvement when we consider larger datasets.



{1,2,3,4,5,6,7,8,9}. => find key = 1

As the mid element 5 > key i.e (1)

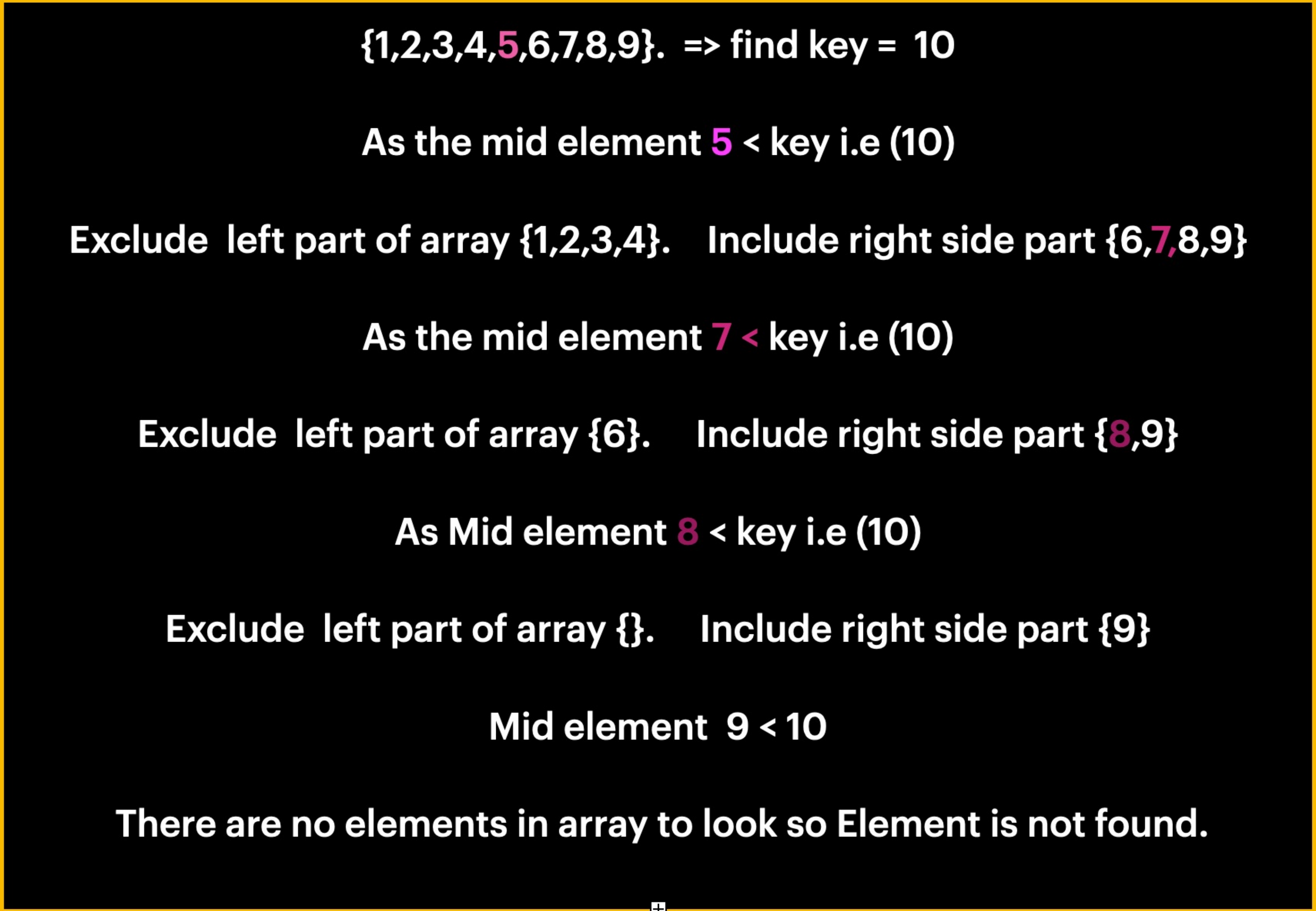
Take left part of array {1,2,3,4}. Exclude right side part {6,7,8,9}

As the mid element 2 > key i.e (1)

Take left part of array {1}. Exclude right side part {3,4}

As Mid element 1 == key (i.e 1)

Element found.



{1,2,3,4,5,6,7,8,9}. => find key = 10

As the mid element 5 < key i.e (10)

Exclude left part of array {1,2,3,4}. Include right side part {6,7,8,9}

As the mid element 7 < key i.e (10)

Exclude left part of array {6}. Include right side part {8,9}

As Mid element 8 < key i.e (10)

Exclude left part of array {}. Include right side part {9}

Mid element 9 < 10

There are no elements in array to look so Element is not found.

2 power 1 = 2

2 power 2= 4

2 power 3= 8

.....

2 power 15= 32,768

....

2 power 31 = 2,147,483,648

Time Complexity => log n with base 2
=> O(log n)

When we working on Sorting the following techniques would be considered:

1. Time Complexity
2. Space Complexity
3. Stability
4. Internal Sort / External Sort
5. Recursive / Non-Recursive
6. Compare / Swap

Time Complexity :

The easiest way to classify an algorithm is by *time complexity*, or by how much relative time it takes to run the algorithm

Stability :

Stability talks about , there can be duplicate elements in an array. After sorting would that order be maintained.

Ex : arr : { 1,7,2,1} Here we have element 1 as duplicate with sort property as colour. So after sort Red colour 1 should come first then the Green 1 should appear.

After sort : arr : { 1,1,7,2}

Space Complexity :

There are two types of classifications for the space complexity of an algorithm:

in-place or *out-of-place* given a different input size.
in-place algorithm talks about swapping the elements in an original array, which results in constant space complexity. Its more space efficient but mis use can leads to data mess.
Out-Place algorithm always takes at the extra copy of data which leads O(n) space complexity.

Internal (RAM) / External (Disk) Sort:

If sorting happens on RAM its internal sort , If the sort happened as hard disk level then its External Sort.
We usually go for External Sort if the input data size is greater than RAM size.

Recursive / Non-Recursive :

Talks about Is the sort login is on iterative or recursive.

Compare/Swap :

Talks about does the sort logic takes how may comparisons & swaps

Selection Sort :

i = 0 {5, 11, 3, 2, 10, 1}

i = 1 {1, 11, 3, 2, 10, 5}

i = 2 {1, 2, 3, 11, 10, 5}

i = 3 {1, 2, 3, 11, 10, 5}

i = 4 {1, 2, 3, 5, 10, 11}

i = 5 {1, 2, 3, 5, 10, 11}

Time Complexity => n^2
Space Complexity => O(1)
Stability => There is No Stability
Internal Sort
Non - Recursive
Swap => swap(n)

Selection Sort Algorithm :

For current element iteration =>
Find out the smallest element if the element found then swap the current element with smallest element.

Repeat the iteration process from index 0 to n-1 .

In the example pink colour represents current element , green colour represents smallest element left in array so that both will be swapped.