Sorted array has binary search tree property. If you take a “mid" element, elements less than "mid” would be on left side & elements greater than “mid" would be on right side. This key we can use to divide array to half in every iteration/recursive call.

{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.

If size of the array is 8 => you can find the element in 3 steps

If we use recursion , then we call method recursively logn times. So that there are logn stack frames would be active.
So In case of reclusion the SpaceComplexity would be O(logn)

n = 8

n/2

n/4

n/8

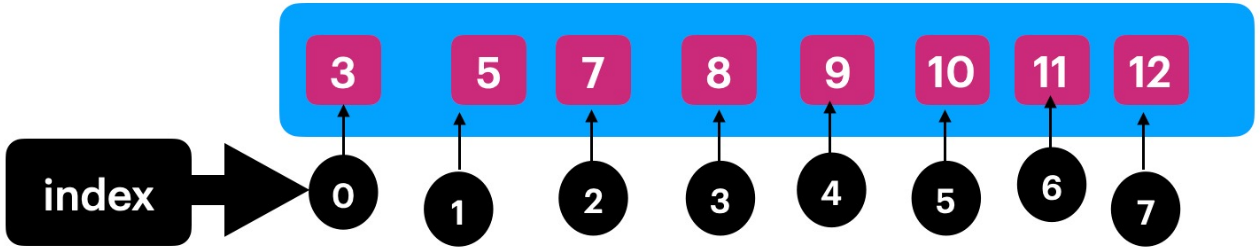
In a binary search for every iteration/ recursion we divide array half.
For n = 8 , n/8 is the possible sub problem so this can be reached in 3 steps .
log2^3 = 8
So the TimeComplexity is O(log(n))

Using iterative
Time Complexity : O(n)
Space Complexity : O(1)

Using Recursion with Binary Search
Time Complexity : O(logn)
Space Complexity : O(logn)
As logn stack frames were active

Using Iterative with Binary Search
Time Complexity : O(logn)
Space Complexity : O(1)

## Find Element 7



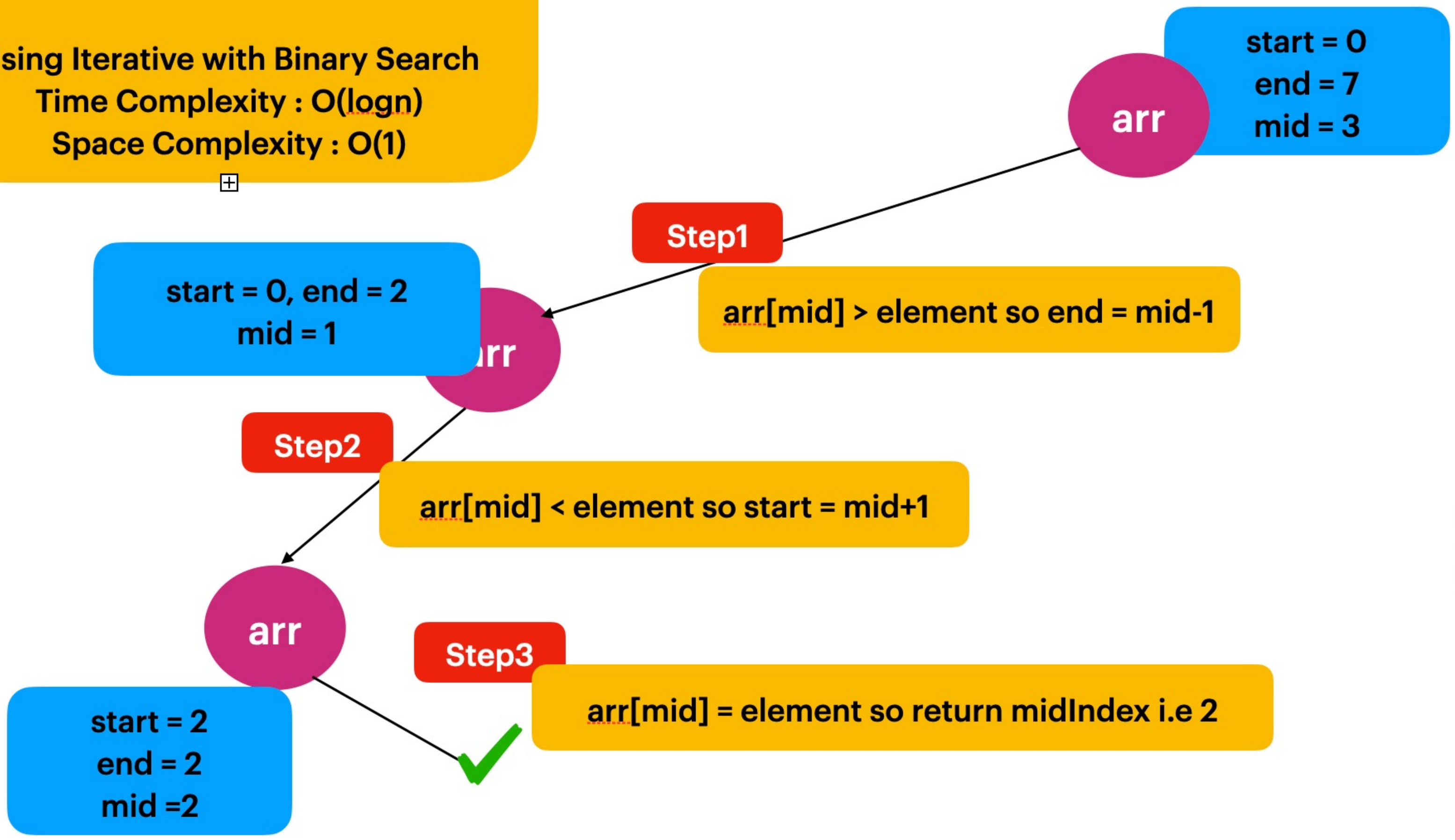

Algorithm to Find an element in SortedArray:
Its Binary Search

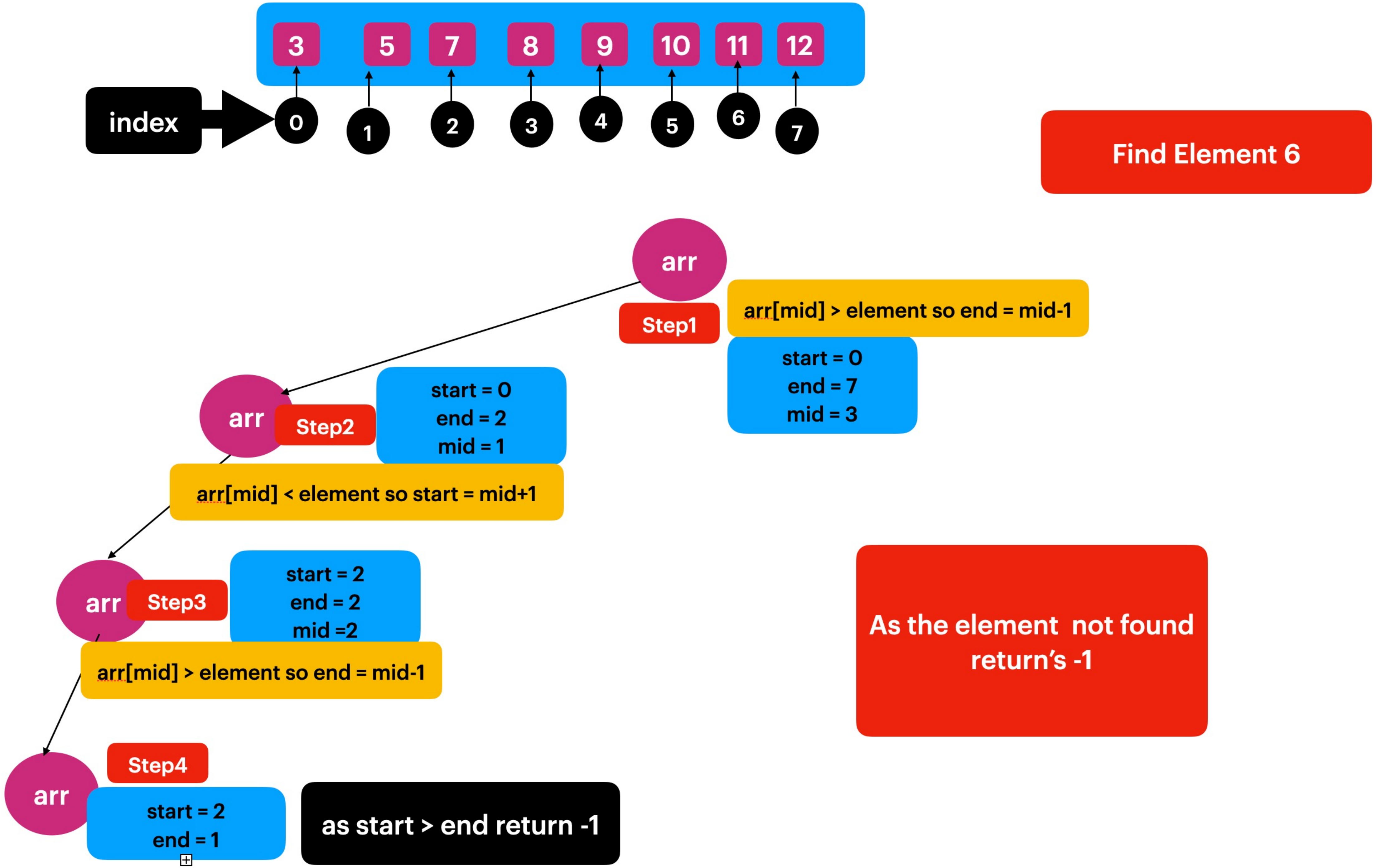
initialise start = 0
end = arr.length-1

1. Find the mid , if the arr[mid] == element then return mid index.

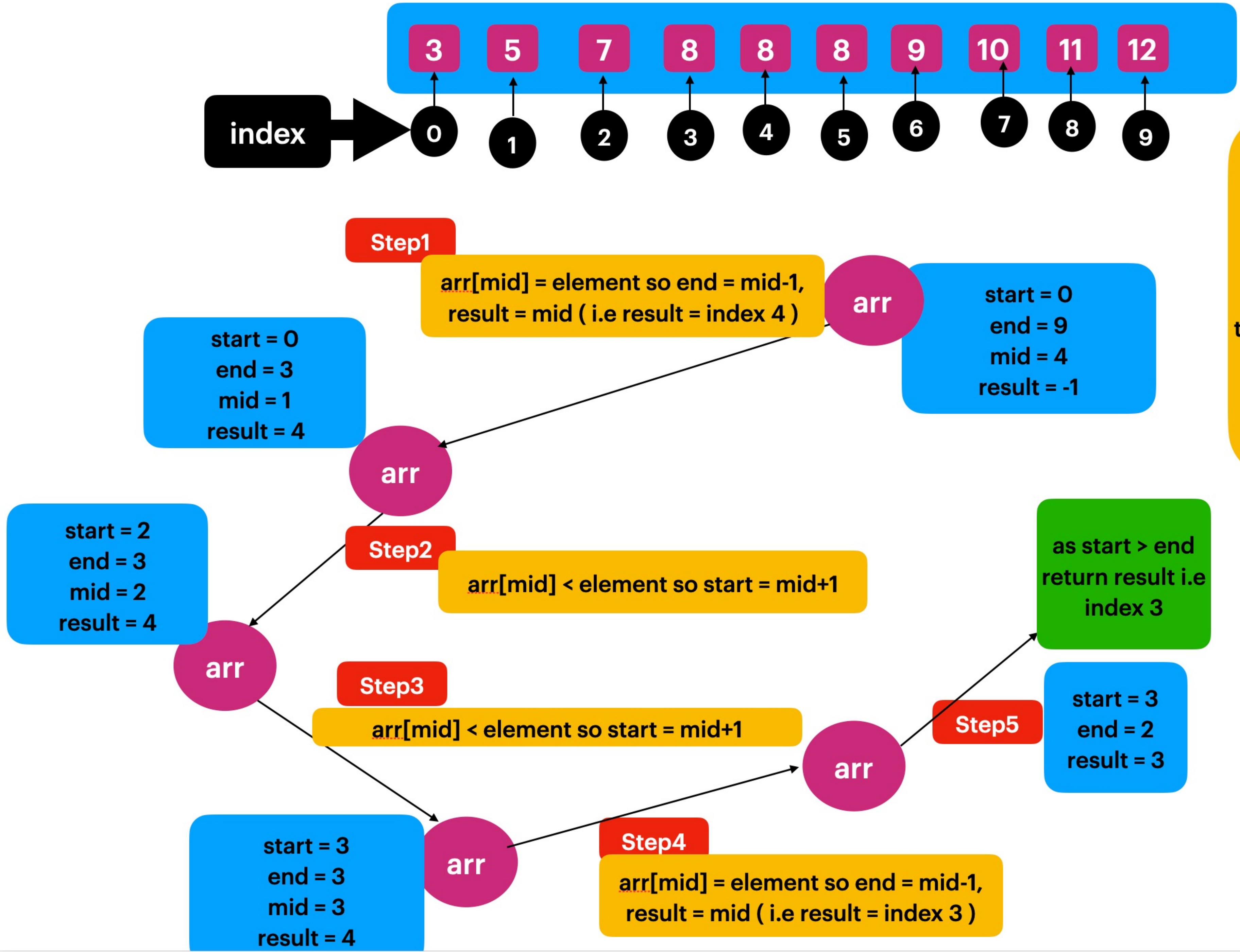
2. If the arr[mid] > element then move left side
end = mid - 1

3. If the arr[mid] < element then move right .
i.e start = mid+1
Base Condition start <= end




3
5
7
8
9
10
11
12
index
0
1
2
3
4
5
6
7
Find Element 6
arr
Step1
arr[mid] > element so end = mid-1
start = 0
end = 7
mid = 3
arr
Step2
start = 0
end = 2
mid = 1
arr[mid] < element so start = mid+1
arr
Step3
start = 2
end = 2
mid =2
arr[mid] > element so end = mid-1
As the element not found return’s -1
arr
Step4
start = 2
end = 1
as start > end return -1

# Find First Occurrence of 8

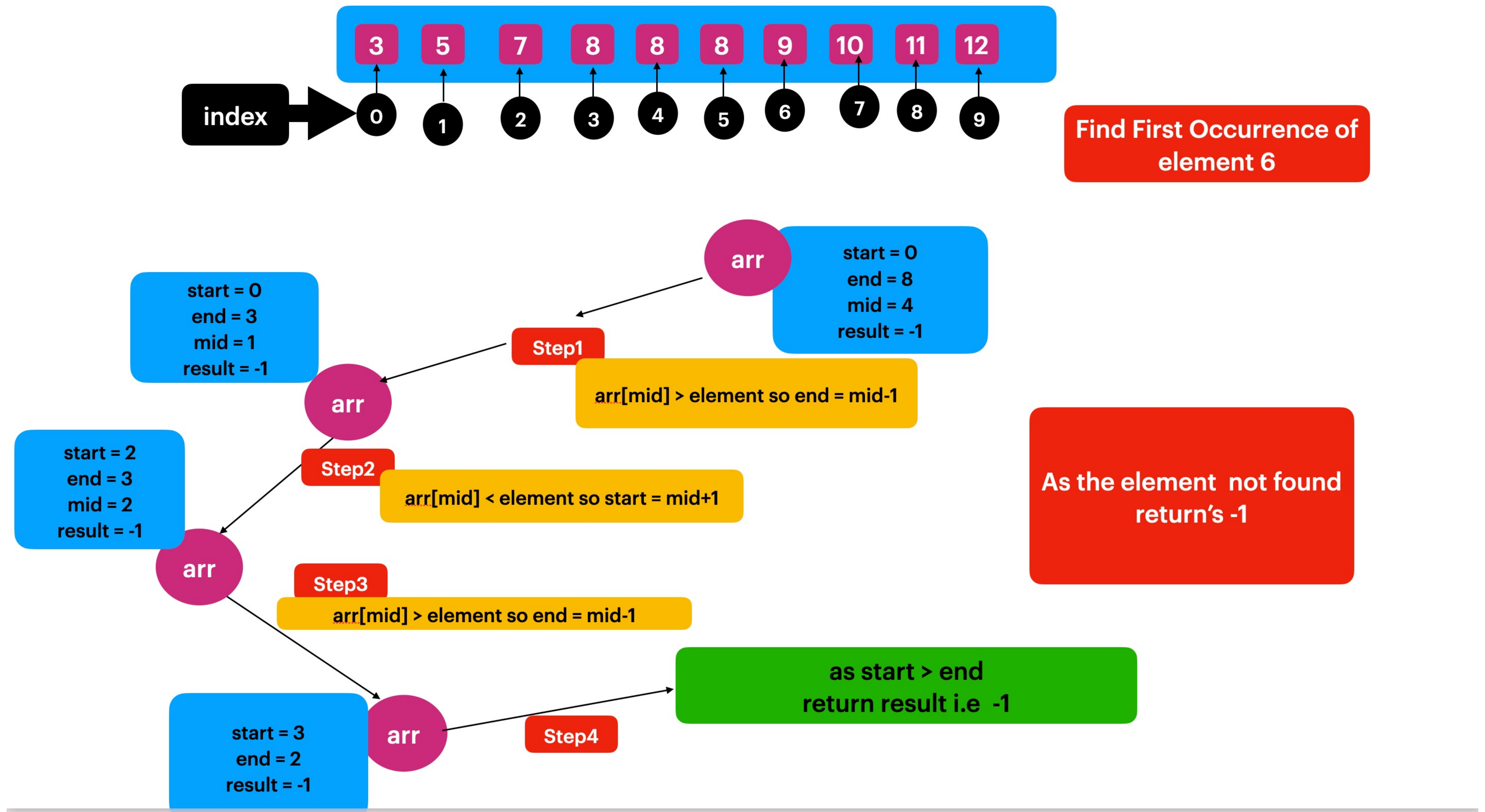



Algorithm :
Same as Binary Search Algorithm (earlier one)
The only difference is , take a helper variable(i.e result), when we find the element, first update the helper variable then move to left side of the array to reach first occurrence.
when arr[mid] == element
result = mid
end = mid -1

Using iterative
Time Complexity : O(n)
Space Complexity : O(1)

Using Recursion with Binary Search
Time Complexity : O(logn)
Space Complexity : O(logn)
As logn stack frames were active

Using Iterative with Binary Search
Time Complexity : O(logn)
Space Complexity : O(1)


3 5 7 8 8 8 9 10 11 12
index
0 1 2 3 4 5 6 7 8 9
Find First Occurrence of element 6
arr
start = 0
end = 8
mid = 4
result = -1
Step1
arr[mid] > element so end = mid-1
start = 0
end = 3
mid = 1
result = -1
arr
Step2
arr[mid] < element so start = mid+1
start = 2
end = 3
mid = 2
result = -1
arr
Step3
arr[mid] > element so end = mid-1
arr
start = 3
end = 2
result = -1
Step4
as start > end
return result i.e -1
As the element not found return’s -1