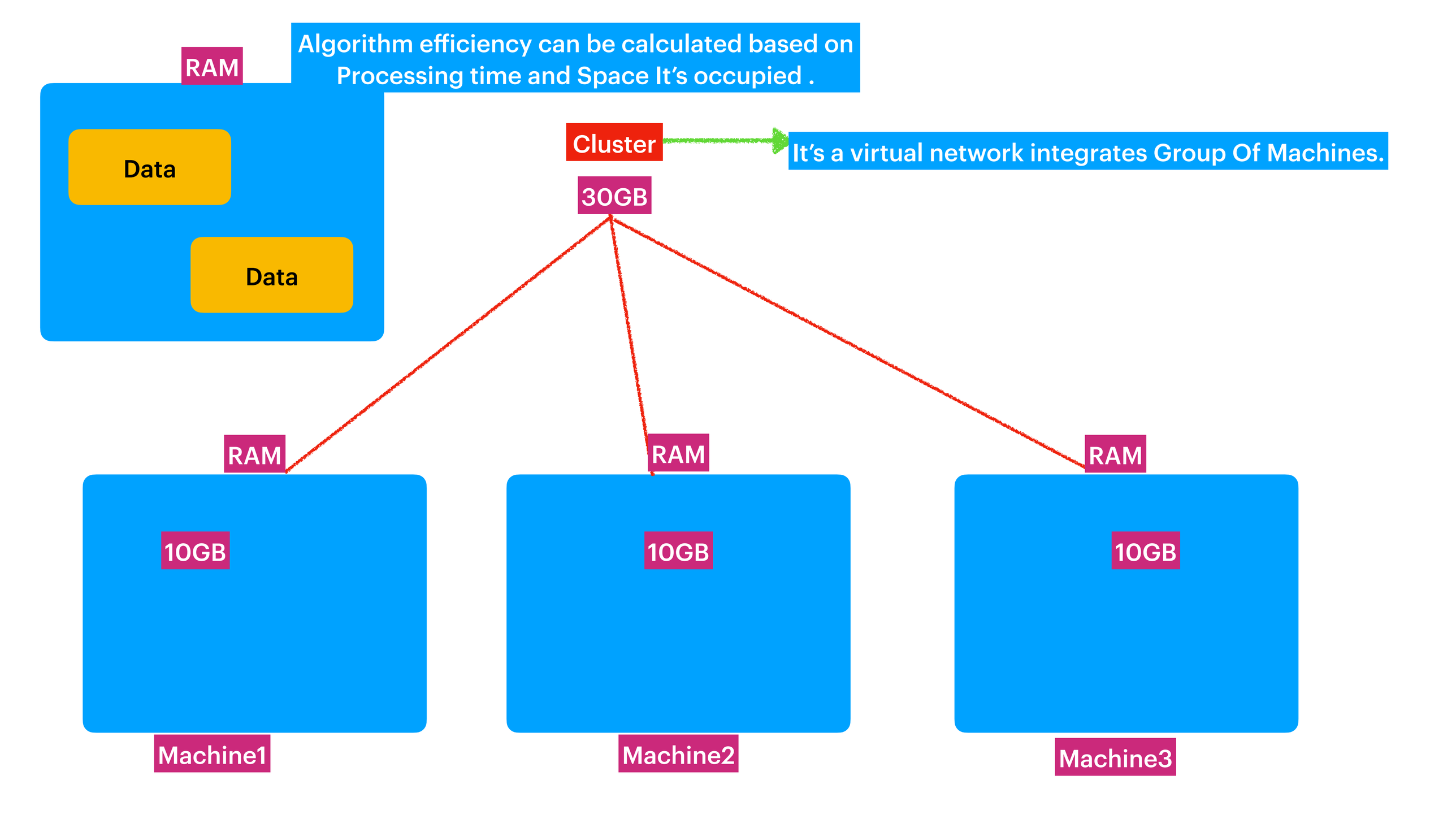
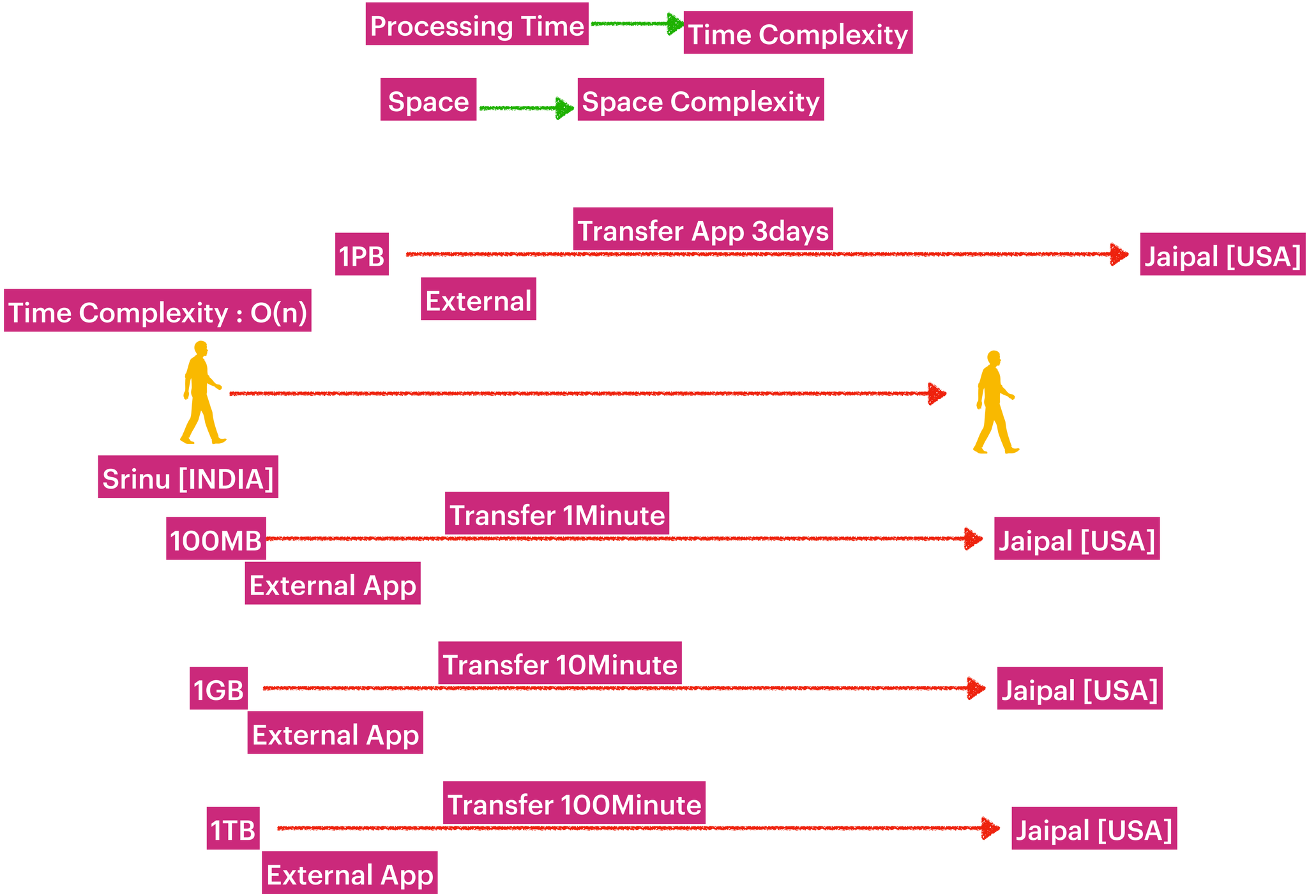

Algorithm efficiency can be calculated based on Processing time and Space It's occupied .
RAM
Data
Data
Cluster
It's a virtual network integrates Group Of Machines.
30GB
RAM
RAM
RAM
10GB
10GB
10GB
Machine1
Machine2
Machine3


Processing Time
Time Complexity
Space
Space Complexity
1PB
Transfer App 3days
Jaipal [USA]
External
Time Complexity : O(n)
Srinu [INDIA]
100MB
Transfer 1Minute
Jaipal [USA]
External App
1GB
Transfer 10Minute
Jaipal [USA]
External App
1TB
Transfer 100Minute
Jaipal [USA]
External App

Processing Time → Time Complexity

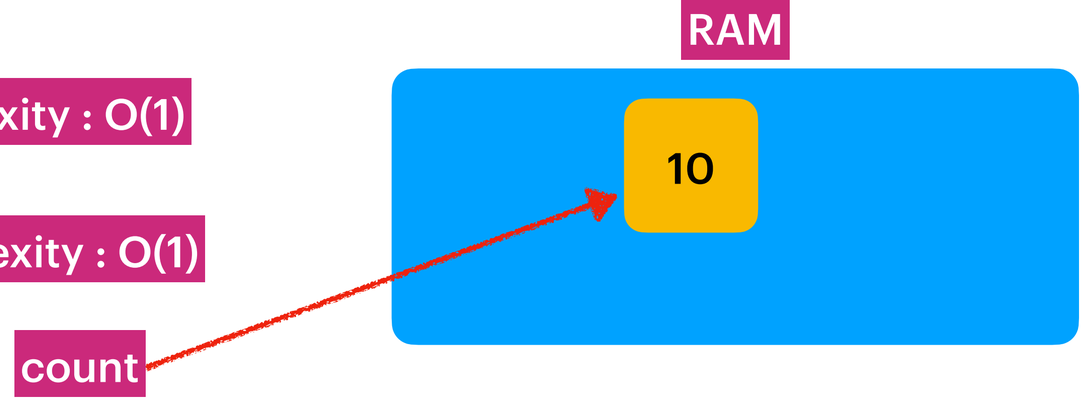
Space → Space Complexity

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

Time Complexity : O(1)

```
int count = 10;
```

Space Complexity : O(1)

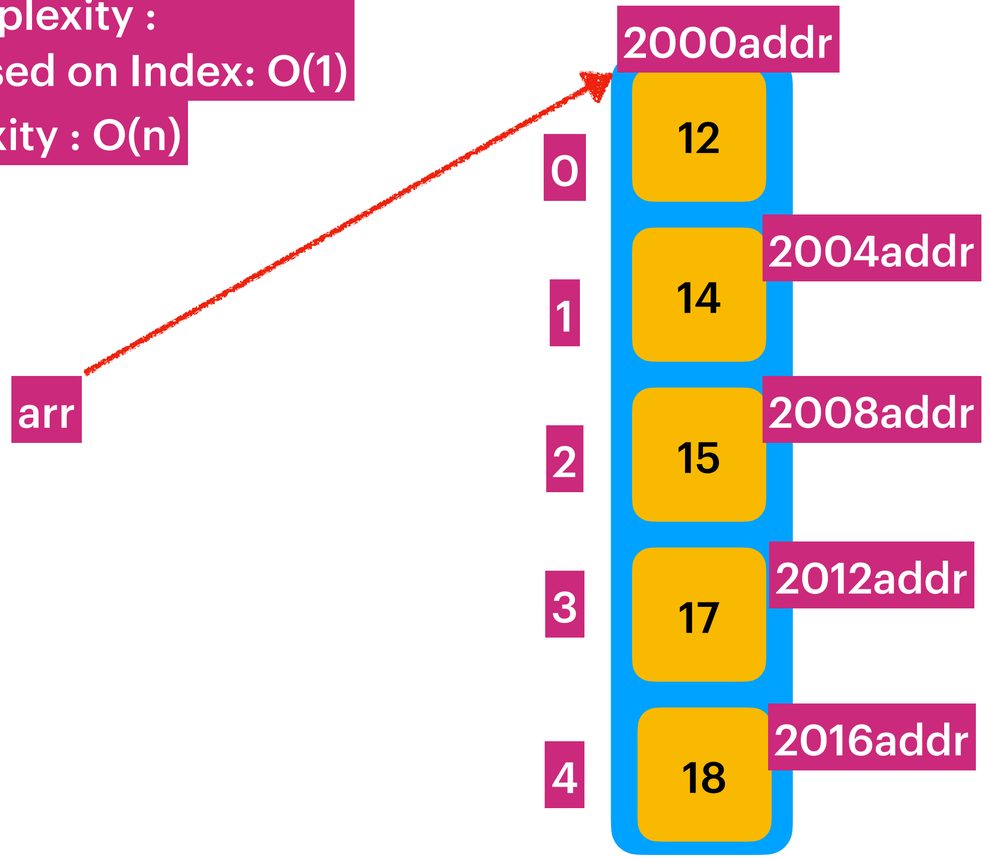


Array

Time Complexity :
Access Element Based on Index: O(1)

Space Complexity : O(n)

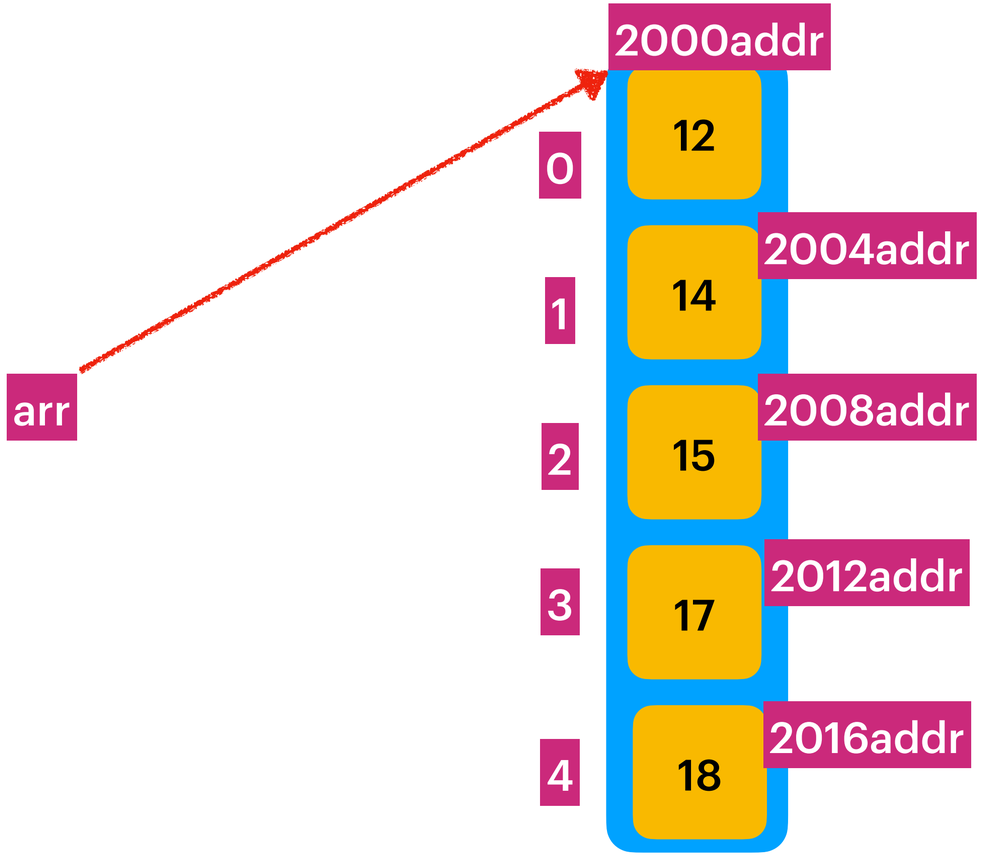
```
int[] arr = {12,14,15,17,18};
```



arr[0] = 2000addr + 0 * 4 = 2000addr = 12

arr[3] = 2000addr + 3 * 4 = 2012addr = 17

arr[4] = 2000addr + 4 * 4 = 2016addr = 18

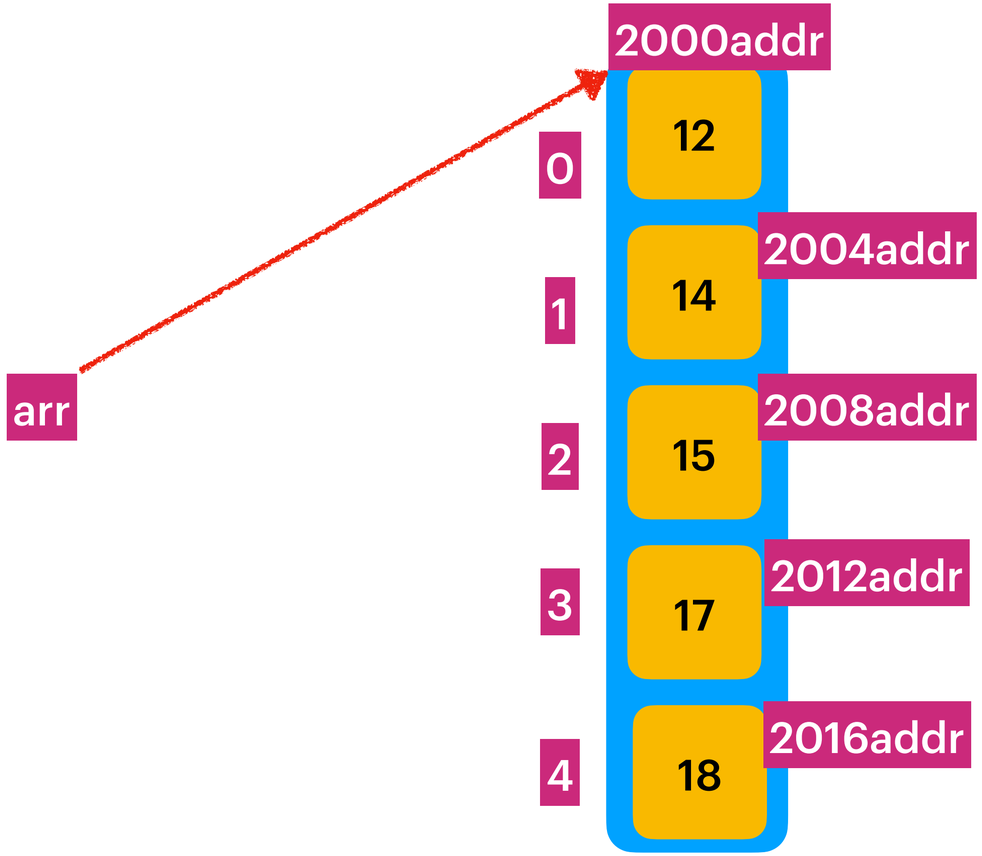


Search an Element in an Array :
Time Complexity : O(n)
Space Complexity : O(1)

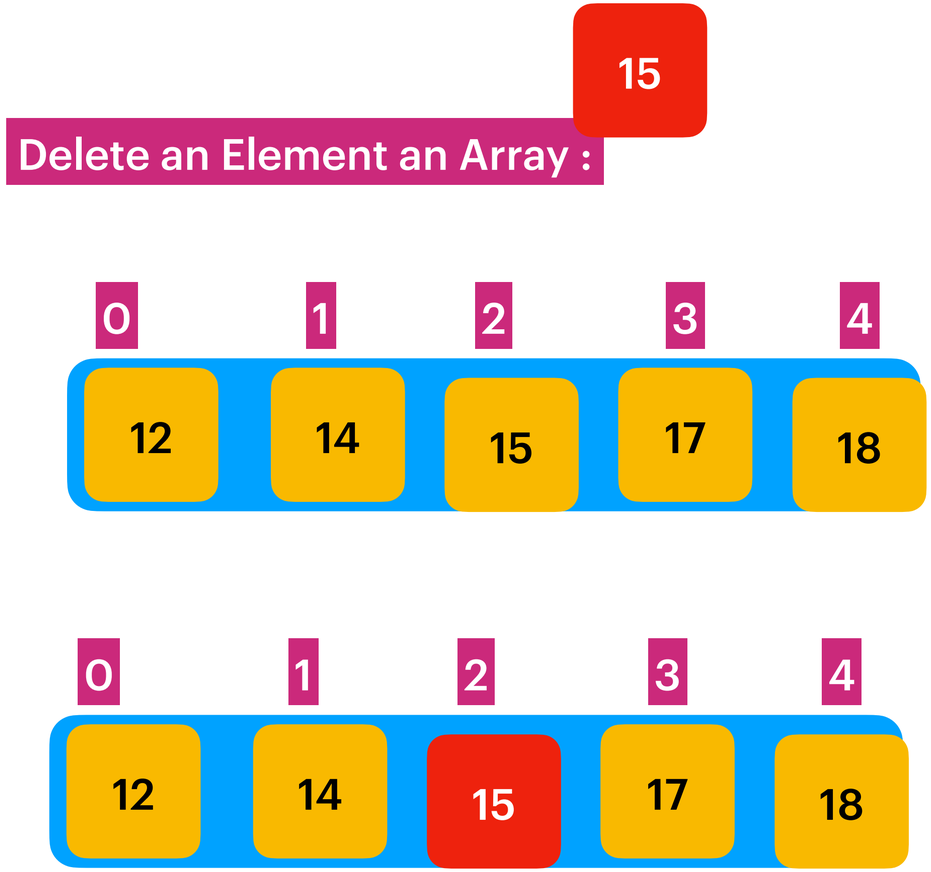
SearchElement 10

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

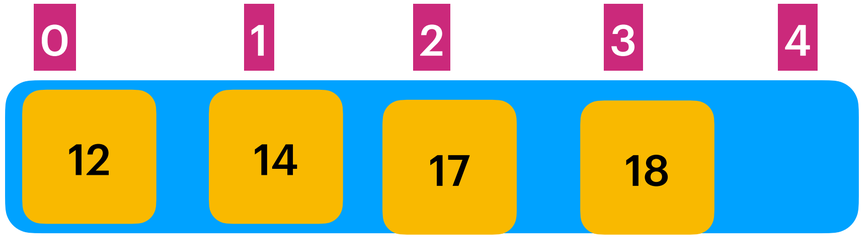
```
for(int i = 0; i < n ; i++)
{
if(arr[i] == 10)
{
return true;
}
}

return false;
```



Delete an Element an Array :
Search & Delete : O(n)
Shifting operation : O(n)
Space Complexity : O(1)



When we delete an element in an Array, we would need to Shift all the right index values before.



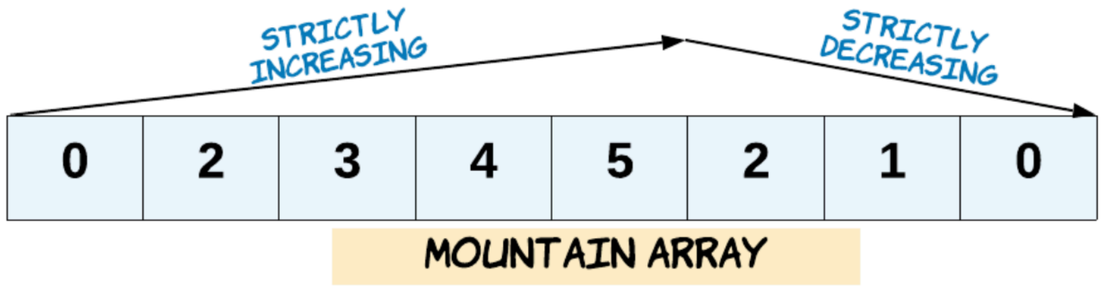
941. Valid Mountain Array

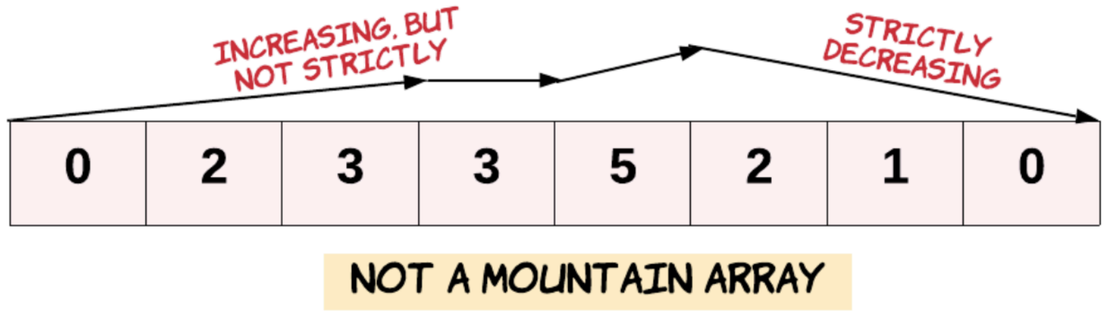
Easy 2073 145 Add to List Share

Given an array of integers `arr`, return `true` *if and only if it is a valid mountain array*.

Recall that arr is a mountain array if and only if:

- `arr.length >= 3`
- There exists some `i` with `0 < i < arr.length - 1` such that:
  - `arr[0] < arr[1] < ... < arr[i - 1] < arr[i]`
  - `arr[i] > arr[i + 1] > ... > arr[arr.length - 1]`





Example 1:

```
Input: arr = [2,1]
Output: false
```

Example 2:

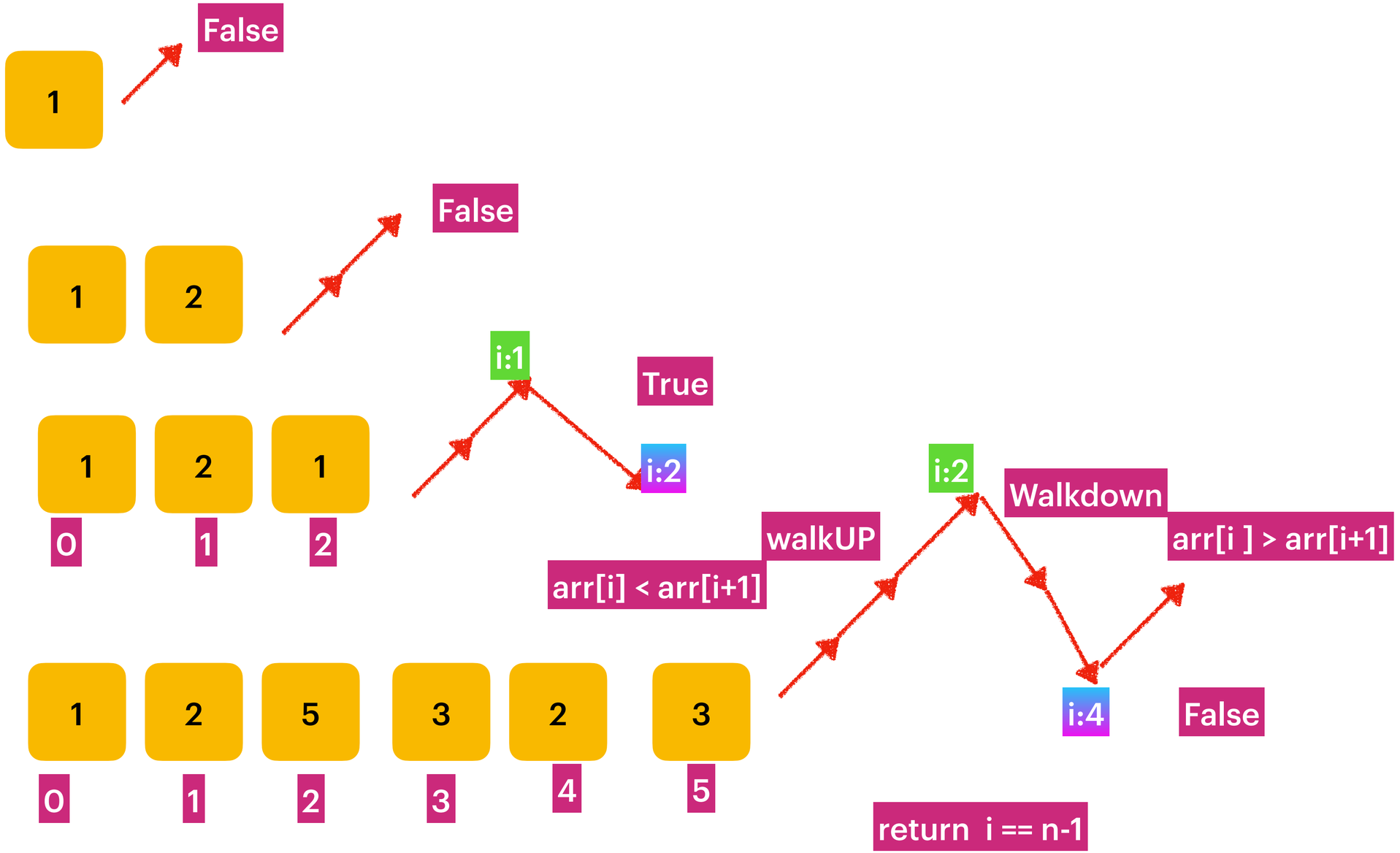
```
Input: arr = [3,5,5]
Output: false
```

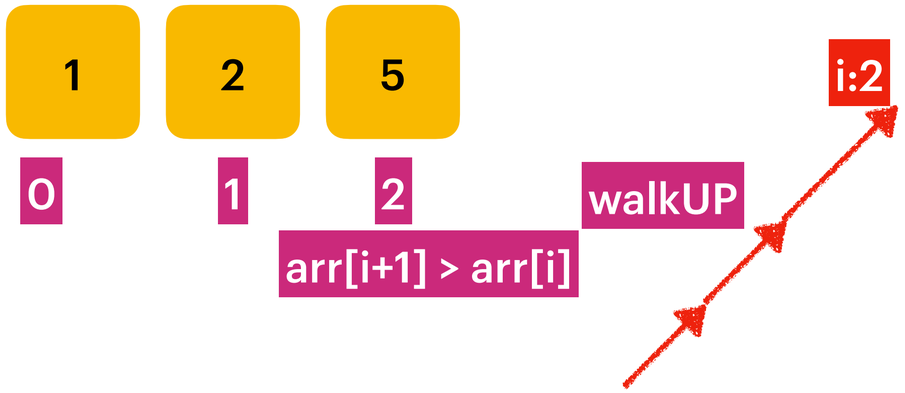
Example 3:

```
Input: arr = [0,3,2,1]
Output: true
```

Constraints:

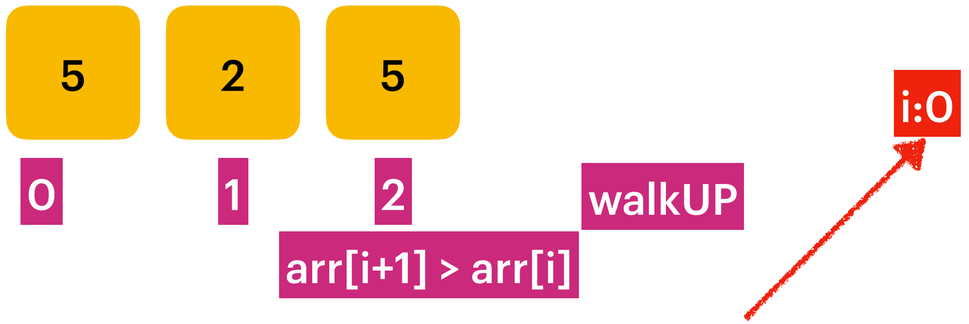
- `1 <= arr.length <= 10^4`
- `0 <= arr[i] <= 10^4`


1
False
1
2
False
1
2
1
0
1
2
i:1
True
i:2
1
2
5
3
2
3
0
1
2
3
4
5
walkUP
arr[i] < arr[i+1]
i:2
Walkdown
arr[i] > arr[i+1]
i:4
False
return i == n-1



Base Checks :

After Walkup if your index either ‘0’ Or ‘n-1’ return false



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

Algorithm

1. WalkUP till arr[i] < arr[i +1]
2. Have Base check

either index ‘0’ or ‘n-1’ return false.
If the elements are ascending order then walkup results to index value as ‘n-1’

If the second element is less than the first element then index value ‘0’

3. WalkDown till arr[i] > arr[i +1]

4. Return true if the index reached to last element otherwise return false:
return i == n-1