SINGLE ELEMENT IN A SORTED ARRAY

☆ In this problem, we are given an array of sorted elements where each element appears twice except one. We are supposed to return that element.

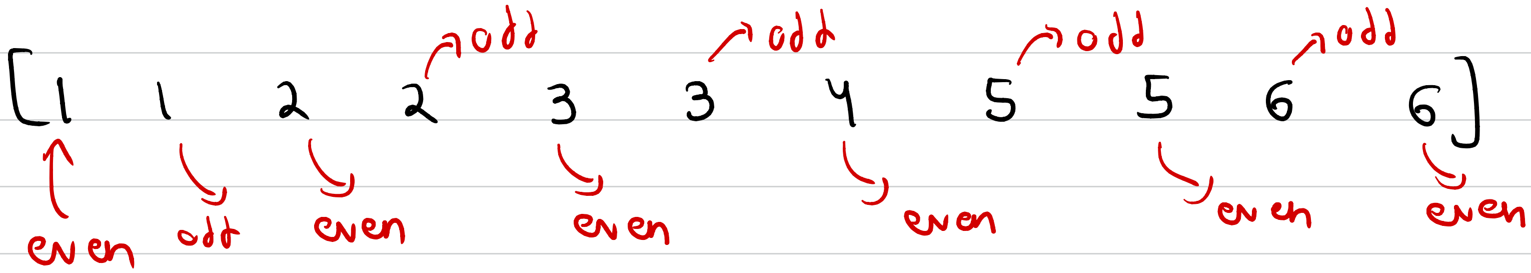
Brute force solution is to obviously take one pass through the entire array, check for the left and right of each element. Optimal solution involves the use of binary search, where we check if our element is on left half or right half by even, odd indexing. To eliminate edge cases we lower our search space to 1 → N − 2.

Pseudocode :

```
singleElementSorted(arr, N) {
    if (N == 1) return arr[0]
    if (arr[0] != arr[1]) return arr[0]
    if (arr[N - i] != arr[N - 2]) return arr[N-1]
    low = 1
    high = N - 2
    while (low <= high) {
        mid = (low + high) / 2
        if (arr[mid] != arr[mid + 1] &&
                arr[mid] != arr[mid - 1]) {
```

```
            return arr[mid]
        } else if (mid % 2 == 0) {
            high = mid - 1
        } else if (mid % 2 != 0) {
            low = mid + 1
        }
    }
    return -1
}
```

This comes from the observation that



If I am standing at an even index, my pair is to my right and hence is the element but if I am standing at an odd index, my pair is to my left and hence is the element.