Array Questions

Q). Find if array is sorted or not

Soln:- arr = [1, 2, 4, 3, 8, 9].

Generic method:-

```
Traverse i;
if arr[i+1] > arr[i].
    i = i+1.
else
    return false.
```

Recursion Logic:-

```
boolean fun (int[] arr, int index)
{
    if (index < arr.length){
        if arr[index+1] < arr[index]
            return false;
```

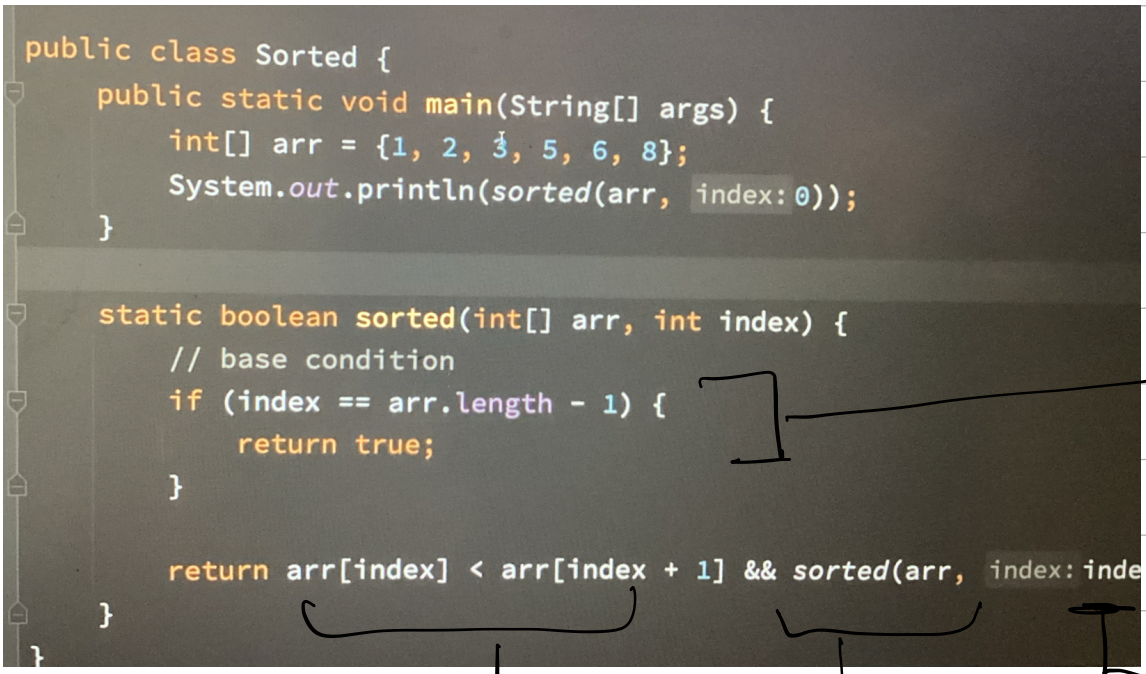
terminating condition.

```
        else
            return fun(arr, index+1)
    }
    else
        return true;
}
```

→ index comes to end but false is not returned so return true.

Alternate method BEST!

CODE :=



```java
public class Sorted {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 5, 6, 8};
        System.out.println(sorted(arr, index: 0));
    }

    static boolean sorted(int[] arr, int index) {
        // base condition
        if (index == arr.length - 1) {
            return true;
        }

        return arr[index] < arr[index + 1] && sorted(arr, index: inde
    }
}
```

Q). Find an item in array using Linear search with recursion.

Solution

Base condition :-

index reaches end without finding target.

```
if( index == arr.length )
{ return false;
}.
```

Logic:- if (arr[index] == target)
return true.

else.
return fun(arr, target, index+1)

Recursion Good Logic:-

fun(arr, target, index) {

if(index == arr.length)
return false;

else
return arr[index]==target || fun(arr, target, index+1);

→ returns true if found.

returns.
true. if index is within arr.length.

Code:-

```
public class Find {
    public static void main(String[] args) {

    }

    static boolean find(int[] arr, int target, int index) {
        if (index == arr.length) {
            return false;
        }
        return arr[index] == target || find(arr, target, index: index + 1);
    }
}
```

Q). How to find the indices where target is present multiple times.

Eg:- arr = [1, 2, 4, 5, 4, 6]

target = 4.

indices = 2, 4th.

Logic:- if (arr[index] == target)
list.add(index);

Here instead of returning true add the index to an arraylist.

Code:-

```
static ArrayList<Integer> list = new ArrayList<>();
static void findAllIndex(int[] arr, int target, int index) {
    if (index == arr.length) {
        return;
    }
    if (arr[index] == target) {
        list.add(index);
    }
    findAllIndex(arr, target, index: index + 1);
}
```

How to return an Arraylist

Q). How to return multiple indices where target is present
(Same as previous question)

Solu:- arr = [1, 2, 3, 4, 4, 8]

target = 4.
ans = [3, 4] → ArrayList.

```
static ArrayList fun(arr, target, index, list)
{
    if(index == arr.length)
        return list;
    if(index == arr[index])
        list.add(index);
    else
        return fun(arr, target, index+1, list);
}
```

return type

Here returning function is done because at last function call list itself is returned

NOTE:- If you create a list within function body, with each function call new list object will be created (more space). So, create it else where and pass it as

parameter to recursive function. So that same object is modified again and again.

CODE:-

written in main function.

```
    ArrayList<Integer> ans = findAllIndex(arr, target: 4, index: 0, new ArrayList<>());
    System.out.println(ans);
}
```

```
static ArrayList<Integer> findAllIndex(int[] arr, int target, int index, ArrayList<Integer> list) {
    if (index == arr.length) {
        return list;
    }
    if (arr[index] == target) {
        list.add(index);
    }
    return findAllIndex(arr, target, index: index + 1, list);
}
}
```

-X- Suppose we are creating a list within function call, without passing it as parameter.

→ With each function call, new list will be created,

So, the element (for example the index) we add to the list in a function call, will not be updated to a common list. As, a new list is created in each function call.

3rd 5th

Diagram: arr = 1, 2, 3, 4, 5, 4, 6

fun(arr, target, idx).

↓

fun(arr, 4, 0)

list[ ]

↓

fun(arr, 4, 1) ↑ [5,3]

list[ ]

↓

fun(arr, 4, 2) ↑ [5,3]

list:[ ]

↓

fun(arr, 4, 3) ↑ [5,3]

list = [3]

↓

fun(arr, 4, 4) ↑ [5]

list = [ ]

↓

fun(arr, 4, 5) ↑ [5]

list = [5]

↓

fun(arr, 4, 6) ↑ [ ]

list = [ ]

- How content is sent to above calls.

With each function call, new list is there. Find a way to pass answers from below

calls to above calls.

NOTE:- We use an arraylist variable to assign the returned list from each function call.

Later, we add that variable to the list variable created with each function call.

At the end, return the list which now contains list elements from below calls.

Code:-

```
static ArrayList<Integer> findAllIndex2(int[] arr, int target, int index) {

    ArrayList<Integer> list = new ArrayList<>();

    if (index == arr.length) {
        return list;
    }

    // this will contain answer for that function call only
    if (arr[index] == target) {
        list.add(index);
    }
    ArrayList<Integer> ansFromBelowCalls = findAllIndex2(arr, target, index: index + 1

    list.addAll(ansFromBelowCalls);

    return list;
}
```

Local list.

variable list that stores answers from calls.

Rotated Binary Search Using Recursion

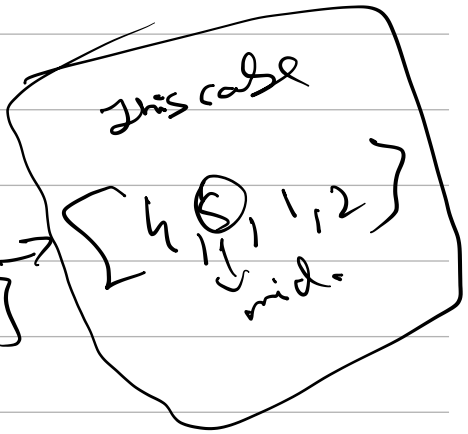
An array is called rotated, if

For example :- arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]

is written as arr = [4, 5, 6, 7, 8, 9, 1, 2, 3]

So, here array is rotated 6 times to get '4' in first index.

Logic :-



① . If arr[s] ≤ arr[mid]

{

if key ≥ arr[s] && key ≤ arr[mid]

end = mid - 1 ;

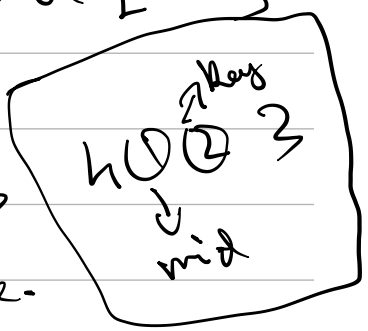
else .

start = mid + 1 ;

}

② . If key ≥ arr[mid] && ≤ arr[end]

start = mid + 1 ;



this case.

③ . else

end = mid - 1 ;

Here

① -> Represents searching in sorted parts. Here both start and mid are in sorted part.

4 5 6 7 | 1 2 3 4

sorted 1 | sorted 2

② -> Here mid and end both are in sorted part.

③ -> Here start is in left sorted part and mid is in right sorted part.

CODE:-

```
static int search(int[] arr, int target, int s, int e) {
    if (s > e) {
        return -1;
    }

    int m = s + (e-s) / 2;
    if (arr[m] == target) {
        return m;
    }
```

```
    if (arr[s] <= arr[m]) {
        if (target >= arr[s] && target <= arr[m])
            return search(arr, target, s, e: m-1);
        } else {
            return search(arr, target, s: m+1, e);
        }
    }

    if (target >= arr[m] && target <= arr[e]) {
        return search(arr, target, s: m+1, e);
    }
    return search(arr, target, s, e: m-1);
}
```