Sheryians Coding School

Live Cohort

Day 85

DSA Notes – Cyclic Sort & Binary Search

Cyclic Sort

Definition:

Cyclic sort is an efficient in-place sorting algorithm specifically used when dealing with consecutive numbers or numbers in a specific range (like 1 to N or 0 to N). It places each element at its correct index based on value (usually x → index x-1 or x → index x).

Code(Examples):

```
while (i < nums.length) {
    let correct = index where nums[i] should be;

    if (nums[i] is valid and not at correct index) {
        swap(nums[i], nums[correct]);
    } else {
        i++;
    }
}
```

Use Case:

- Sorting numbers in range without extra space.
- Finding missing or duplicate numbers in limited range.
- Efficient for in-place manipulation of arrays.

Interview Q&A:

Q 94 – Missing Number

Definition:

Find the missing number in an array containing numbers from 0 to n, with one number missing.

```
var missingNumber = function(nums) {
    let i = 0;
    while (i < nums.length) {
        let correct = nums[i];
        if (nums[i] < nums.length && nums[i] !== nums[correct]) {
            [nums[i], nums[correct]] = [nums[correct], nums[i]];
        } else {
            i++;
        }
    }
    for (let i = 0; i < nums.length; i++) {
        if (nums[i] !== i) return i;
    }
    return nums.length;
};
```

Q 95 – Find All Numbers Disappeared in an Array

Definition:

Find all the numbers from 1 to n that are missing in an array.

```
var findDisappearedNumbers = function(nums) {
    let i = 0;
    while (i < nums.length) {
        let correct = nums[i] - 1;
        if (nums[i] !== nums[correct]) {
            [nums[i], nums[correct]] = [nums[correct], nums[i]];
        } else {
            i++;
        }
    }
    let result = [];
    for (let i = 0; i < nums.length; i++) {
        if (nums[i] !== i + 1) result.push(i + 1);
    }
    return result;
};
```

Use Case:

- Efficient for identifying multiple missing numbers in an unsorted array of values from 1 to n.

Q 96 – First Missing Positive

Definition:

Find the smallest missing positive integer in an unsorted integer array.

```
var firstMissingPositive = function(nums) {
    let i = 0;
    while (i < nums.length) {
        let correct = nums[i] - 1;
        if (nums[i] > 0 && nums[i] <= nums.length && nums[i] !== nums[correct]) {
            [nums[i], nums[correct]] = [nums[correct], nums[i]];
        } else {
            i++;
        }
    }
    for (let i = 0; i < nums.length; i++) {
        if (nums[i] !== i + 1) return i + 1;
    }
    return nums.length + 1;
};
```

Binary Search

Definition:

Binary search is an efficient algorithm to search a target element in a sorted array by repeatedly dividing the search range in half. Works in O(log n) time.

```
let start = 0, end = nums.length - 1;
while (start <= end) {
    let mid = Math.floor((start + end) / 2);
    if (nums[mid] === target) return mid;
    else if (nums[mid] < target) start = mid + 1;
    else end = mid - 1;
}
return -1; // or return start, based on use case
```

Use Case:

- Find element position in sorted array.
- Find insert position.
- Count occurrences.
- Locate first/last positions.

Interview Questions Based on Binary Search:

Q 98 – Search Insert Position (Leetcode 35)

Definition:

Return the index if the target is found. If not, return the index where it would be inserted.

```
var searchInsert = function(nums, target) {
    let start = 0, end = nums.length - 1;
    while (start <= end) {
        let mid = Math.floor((start + end) / 2);
        if (nums[mid] === target) return mid;
        else if (nums[mid] < target) start = mid + 1;
        else end = mid - 1;
    }
    return start;
};
```

Use Case:

Useful in autocomplete, ranking, and indexing systems where exact match or insert position is needed.

Q 99 – First and Last Position of Element in Sorted Array

Definition:

Find the starting and ending position of a target value.

```
function searchRange(nums, target) {
    return [findIndex(nums, target, true), findIndex(nums, target, false)];
}
function findIndex(nums, target, findFirst) {
    let start = 0, end = nums.length - 1, index = -1;
    while (start <= end) {
        let mid = Math.floor((start + end) / 2);
        if (nums[mid] === target) {
            index = mid;
            if (findFirst) end = mid - 1;
            else start = mid + 1;
        } else if (nums[mid] < target) start = mid + 1;
        else end = mid - 1;
    }
    return index;
}
```

Use Case:

Used in log analysis, search engines, and range queries to find boundaries.

Q 100 – Count of an Element in a Sorted Array

Definition:

Find how many times a given element occurs in a sorted array.

```javascript
function countOccurrences(nums, target) {
    const first = findIndex(nums, target, true);
    if (first === -1) return 0;
    const last = findIndex(nums, target, false);
    return last - first + 1;
}
function findIndex(nums, target, findFirst) {
    let start = 0, end = nums.length - 1, index = -1;
    while (start <= end) {
        let mid = Math.floor((start + end) / 2);
        if (nums[mid] === target) {
            index = mid;
            if (findFirst) end = mid - 1;
            else start = mid + 1;
        } else if (nums[mid] < target) start = mid + 1;
        else end = mid - 1;
    }
    return index;
}
```