Interview Problems on Arrays in DSA

Pre Placement Training Program

CAN YOU ALL SEE & HEAR ME?

Know About Your Teacher

With

Pragya Rustagi

- Expertise in DSA, OOP,Java, Python, and AWS
- Successfully mentored over 1000 students

Interview Problems on Arrays

What will you learn today?

- ❑ Find the duplicate Number
- ❑ Find the missing and repeating number in array
- ❑ Merge Intervals
- ❑ Majority Element
- ❑ Merge sorted Array
- ❑ Kth Smallest Element in an Array
- ❑ Find minimum number of merge operations to make an array palindrome

Find the Duplicate Number

Q arr[ ] = 1 3 4 2 2

Output → 2

Brute ① Sort → 1 2 2 3 4

```
for (i=1; i<n; i++) {
    if (arr[i] == arr[i-1])
        duplicate = arr[i];
        return;
    }
}
```

T = O(n log n)

S = O(1)

Optimized

1 3 4 2 2

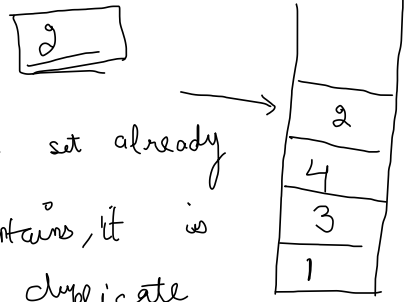
2nd

HashMap

→ number, freq

freq > 1 → duplicate

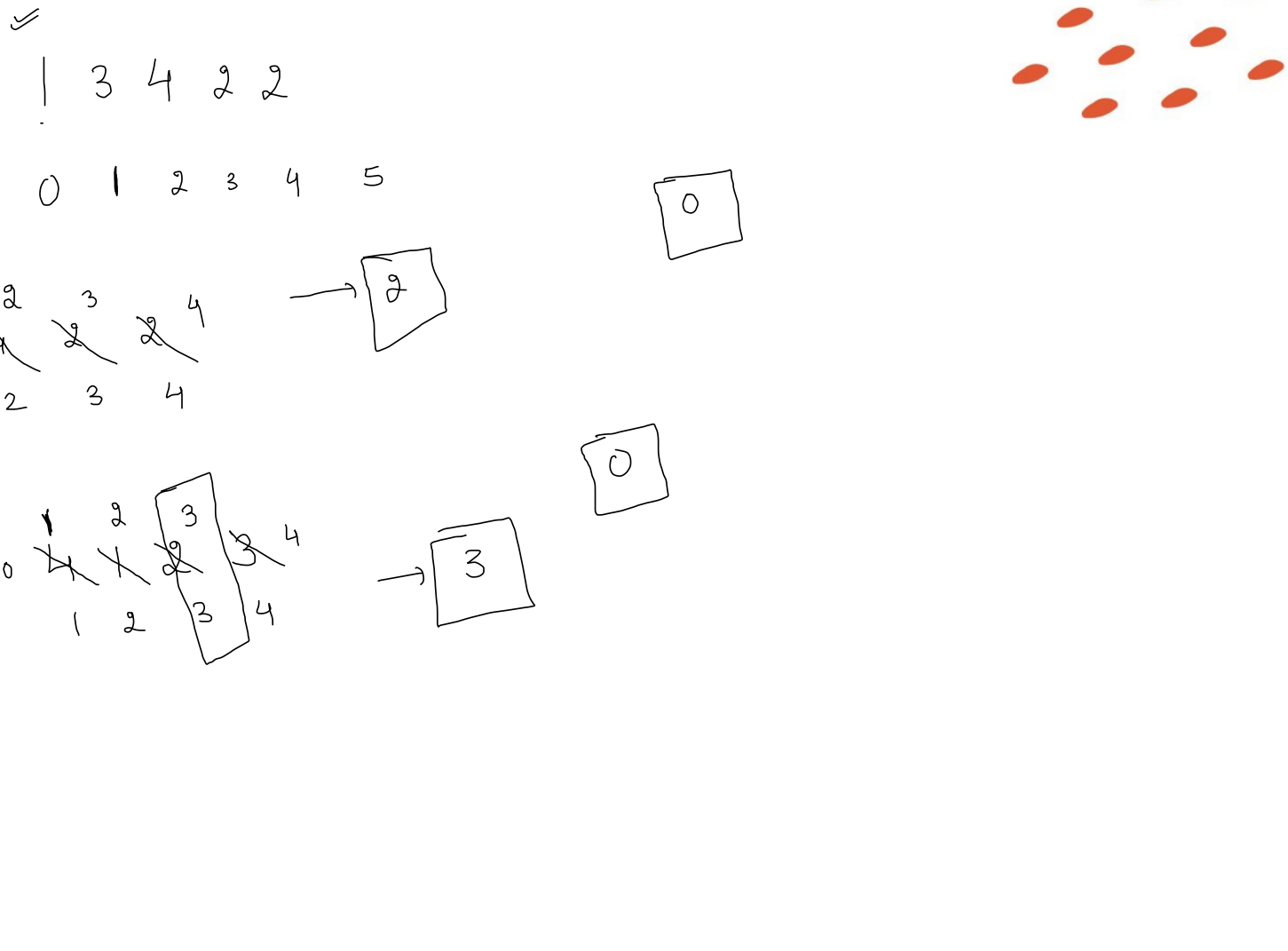
HashSet



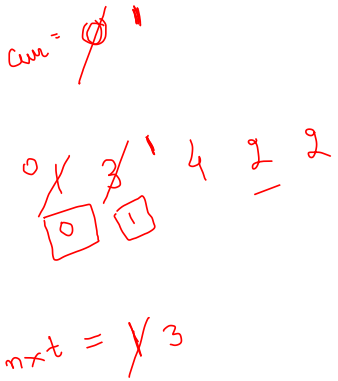
if set already contains, it is duplicate

T = O(n), S = O(n)

↳ Optimize


1 3 4 2 2
0 1 2 3 4 5
0
2
3

```java
package Arrays;

public class FindDuplicate {

    private static int store(int[] nums, int cur) {
        if (cur == nums[cur])
            return cur;
        int nxt = nums[cur];
        nums[cur] = cur;
        return store(nums, nxt);
    }

    private static int findDuplicate(int[] nums) {
        return store(nums, 0);
    }

    public static void main(String[] args)
    {
        int arr[] = { 1,3,4,2,2 };

        System.out.println(findDuplicate(arr));
    }
}
```



10 Find the missing & repeating in an Array starting from → 1

arr = [3 1 5 4 2 6 7 5]

missing = 8

Repeating = 5

(1) Hashing →

0	1	1	1	1	2	1	1	0
0	1	2	3	4	5	6	7	8

Loop → 1 to 8 → number = 0 (missing)

→ number > 1 (repeating)

T = O(N)

→ S = O(N) → Optimize

8

3 1 5 4 2 6 7 5

(2) Sorting → 1 2 3 4 5 5 6 7

8

missing

repeating 5

$T = O(N \log N)$

$S = O(1)$

↑ Optimize

2 formulas

$\text{Sum}(1 \ldots N) = \frac{n(n+1)}{2}$ $\quad (1 + 2 + 3 + 4 \ldots N)$

$\text{Sum}(1^2 + 2^2 + 3^2 + 4^2 + \ldots N^2) = \frac{N(N+1)(2N+1)}{6}$

{ 3 1 5 4 2 6 7 5 }

for → 1 to 8 = $\frac{n(n+1)}{2}$ = $\frac{48 \times 9}{2}$ = 36 → SN

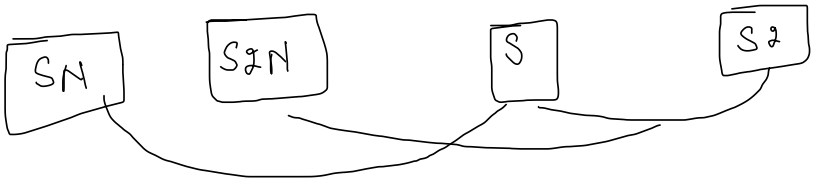
$1^2 + 2^2 \cdots 8^2 = \frac{n(n+1)(2n+1)}{6}$

$= \frac{8 \times 9 \times 17}{6} = 204$ → S2N

1 12
× 17
84
12×
204

3 + 1 + 5 + 4 + 2 + 6 + 7 + 5 = 33 → S

$3^2 + 1^2 + 5^2 + 4^2 - - \cdots 5^2 = 165$ → S2



val1 $S - SN = x - y$ —

val2 $S2 - S2N = x^2 - y^2$ —

$x^2 - y^2 = (x+y)(x-y)$

$x+y = \dfrac{x^2 - y^2}{x-y}$ } val3

$x - y = \text{val3}$

$x + y = \text{val1}$

$+$

$x = \text{val3} + \text{val1}$

$y = \cancel{\text{val1}}\ x - \text{val1}$

$S = 33$

$SN = 36$

$S_2 = 165$

$S_2N = 204$

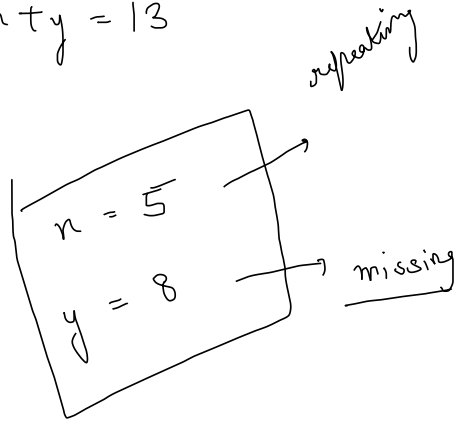
$$x + y = \frac{165 - 204}{33 - 36} = \frac{-39}{-3} = 13$$

$O(1) \longrightarrow$ S & SN $\{ O(N)$

$T = O(N)$
$S = O(1)$

$x - y = -3$

$x + y = 13$



```java
package Arrays;
import java.util.*;

public class MissingAndRepeatingNumber {

    public static int[] findMissingRepeatingNumbers(int[] a) {
        long n = a.length; // size of the array
        // Find Sn and S2n:
        long SN = (n * (n + 1)) / 2;
        long S2N = (n * (n + 1) * (2 * n + 1)) / 6;

        // Calculate S and S2:
        long S = 0, S2 = 0;
        for (int i = 0; i < n; i++) {
            S += a[i];
            S2 += (long)a[i] * (long)a[i];
            System.out.println(S2);
        }

        //S-Sn = X-Y:
        long val1 = S - SN;

        // S2-S2n = X^2-Y^2:
        long val2 = S2 - S2N;

        //Find X+Y = (X^2-Y^2)/(X-Y):
        val2 = val2 / val1;

        //Find X and Y: X = ((X+Y)+(X-Y))/2 and Y = X-(X-Y),
        // Here, X-Y = val1 and X+Y = val2:
        long x = (val1 + val2) / 2;
        long y = x - val1;

        int[] ans = {(int)x, (int)y};
        return ans;
    }

    public static void main(String[] args) {
        int[] a = {3, 1, 2, 5, 4, 6, 7, 5};
        int[] ans = findMissingRepeatingNumbers(a);
        System.out.println("The repeating and missing numbers are: {"
                + ans[0] + ", " + ans[1] + "}");
    }
}
```

Q Majority Element

arr = [2 4 2 4 4 4] , size = 6

Element that occurs more than $N/2$ times

$\frac{6}{2} = 3$ times

o/p = 4

Brute → 2 loops → i = 2 4 2 4 4 4

Count++ j = i+1 ..... arr[i] = arr[j]

Count > N/2

T = O(N²)
S = O(1)

majority → 4

Optimize

2 4 2 4 4 4

(2) mg

Hashmap → Key, value

↓ number ↓ frequency

> N/2

Output

T = O(N)

S = O(N)

Optimize ✗

(3) Greedy

4 2 4 4 4

majE = 2 majC = 1

```
for (i = 1 ; i<N; i++) {
    (arr[i] == majE)  majC ++
else  majC --;
if (majC == 0) {
        majE = arr[i]
        majC = 1 } }
```

X 2 4 2 4 4 4

maj E = 2 maj C = 1 0

(_ _ _ _) — N/2+1

maj C ≥ 0

4 (maj C == 0)

maj E = 4

maj C = 1 0

4 == 2 X

maj E = 2 4

maj C = 1 0 N+1 = 2

+1

```java
package Arrays;

import java.util.*;

public class MajorityElement {

    static int printMajority(int arr[]) {

        int majElement = arr[0], majCount = 1;

        for(int i=1;i<arr.length;i++) {

            if(arr[i] == majElement) {
                majCount++;
            }
            else {
                majCount--;
                if(majCount == 0) {
                    majElement = arr[i];
                    majCount = 1;
                }
            }
        }
        return majElement;
    }

    public static void main(String[] args)
    {
        int arr[] = { 2,2,4,4,4,4,4 };

        System.out.println(printMajority(arr));
    }
}
```

majE = 2

majC = 1

// O(N)

} Assumption wrong

Update majE & majC

T = O(N)

S = O(1)

10:25

① Minimum number of merge operations to make an array palindromic

Palindrome → N I T I N

1 2 3 2 1

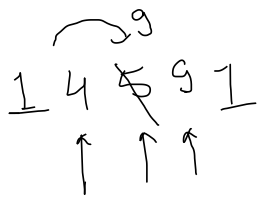
arr = 1 4 5 9 1

1 9 9 1 } Yes

start i = 0

end j = n-1

```
if (arr[i] == arr[j])
{
    i++;
    j--;
}
```



merge = 0

X

```
if ( arr[i] < arr[j] ) {
}
    i++;
    arr[i] += arr[i-1]; // merge
    merge++;
```

}

i > j X

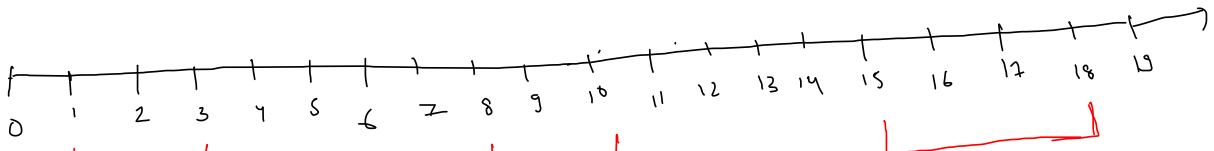
```
if ( arr[i] > arr[j] ) {
}   j--;
    arr[j] += arr[j+1]
    merge++;
}
```

```java
package Arrays;

public class PalindromicArrays {

    // Returns minimum number of count operations
    // required to make arr[] palindrome
    static int findMinOps(int[] arr,int n)
    {
        int ans = 0; // Initialize result

        // Start from two corners
        for (int i=0,j=n-1; i<=j;)
        {
            // If corner elements are same,
            // problem reduces arr[i+1..j-1]
            if (arr[i] == arr[j])
            {
                i++;
                j--;
            }

            // If left element is greater,then
            // we merge right two elements
            else if (arr[i] > arr[j])
            {
                // need to merge from tail.
                j--;
                arr[j] += arr[j+1] ;
                ans++;
            }

            // Else we merge left two elements
            else
            {
                i++;
                arr[i] += arr[i-1];
                ans++;
            }
        }

        return ans;
    }

    // Driver method to test the above function
    public static void main(String[] args)
    {
        int arr[] = new int[]{1,4,5,9,1};

        System.out.println("Count of minimum operations is "+ findMinOps(arr,arr.length));

    }
}
```

Q Merge Intervals

{1, 3} {2, 6} {8, 10} {15, 18}

Overlap



colliding

→ O/P

{1, 6}

{8, 10}

{15, 18}

{1,3} {2,6} {8,10} {15, 18}

Brute

1,3 2,6 = {1,6}

3 > 2 → merge

1,5 5,10 = {1,10}

5 == 5 → merge

{1,3} − {2,6} = {1,6}

8,10 15,18

10 >= 15 ✗

for(i=0 ; i<n-1
{

for(j=i+1 ; j<N
}

=

}

T = O(N²)

S = O(N)

Optimize

Approach {1,3} {2,6} {8,10} {15,18}

① → Sort (1,3) (2,6) (8,10) (15,18) =

② (1,3) → (2,6) × ⌊, ⌊, } O(N)

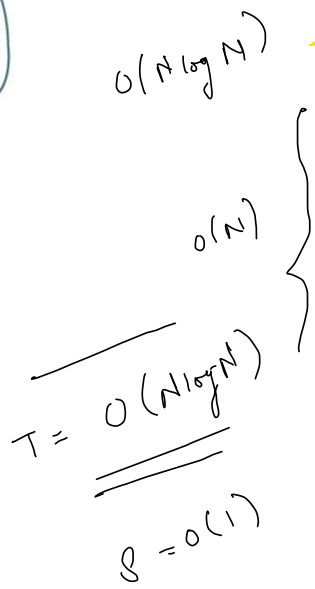
(1,6) → (8,10) ×

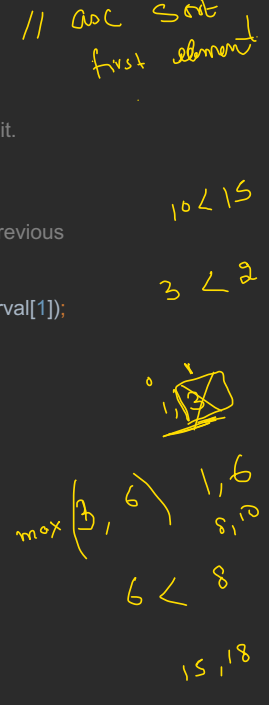
List of List ans

(1,6)
(8,10)
(15,18)

T = O(n log n)

```java
package Arrays;
import java.util.*;

public class MergeIntervals {

    private static int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
        LinkedList<int[]> merged = new LinkedList<>();
        for (int[] interval : intervals) {
            // if the list of merged intervals is empty or if the current
            // interval does not overlap with the previous, simply append it.
            if (merged.isEmpty() || merged.getLast()[1] < interval[0]) {
                merged.add(interval);
            }
            // otherwise, there is overlap, so we merge the current and previous
            // intervals.
            else {
                merged.getLast()[1] = Math.max(merged.getLast()[1], interval[1]);
            }
        }
        return merged.toArray(new int[merged.size()][]);
    }

    public static void main(String[] args) {
        int[][] intervals = {{1,3},{2,6},{8,10},{15,18}};

        int[][] results = merge(intervals);

        for(int[] interval : results) {
            System.out.println(interval[0] + " " + interval[1]);
        }

    }
}
```





Q Merge 2 Sorted Arrays

(m) arr1 = {1, 2, 3, 0, 0, 0} (6) → Sorted

(n) arr2 = {2, 5, 6}

Array → arr1 = 1 2 2 3 5 6

size = 6

Brute

1 2 3 2 5 6 } → m+n

→ (m+n) log (m+n)] optimize

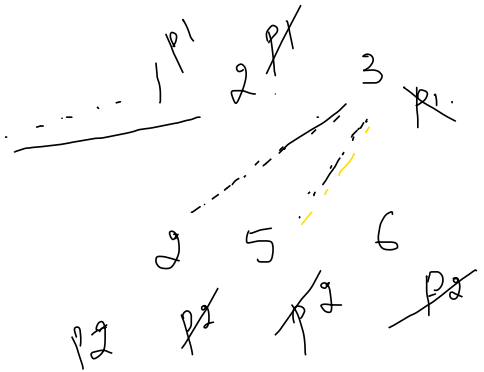
Sort

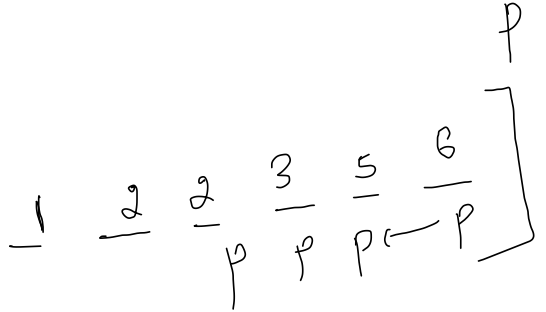
Space = O(m+n)

arr1 = {1, 2, 3, 0, 0, 0} → 3 (m)

arr2 = {2, 5, 6} → 3 (n)

p1 = arr1[m-1] p2 = arr2[n-1]





p2 = 0

```java
package Arrays;

public class MergeSortedArrays {

    private static void merge(Integer[] nums1, int m, Integer[] nums2, int n) {
        // Set p1 and p2 to point to the end of their respective arrays.
        int p1 = m - 1;
        int p2 = n - 1;

        // And move p backwards through the array, each time writing
        // the smallest value pointed at by p1 or p2.
        for (int p = m + n - 1; p >= 0; p--) {
            if (p2 < 0) {
                break;
            }
            if (p1 >= 0 && nums1[p1] > nums2[p2]) {
                nums1[p] = nums1[p1--];
            } else {
                nums1[p] = nums2[p2--];
            }
        }
    }

    public static void main(String[] args)
    {
        Integer nums1[] = new Integer[] {1,2,3,0,0,0};
        Integer nums2[] = new Integer[] {2,5,6};
        int m = 3;
        int n = 3;
        merge(nums1, m, nums2, n);


        for(int i=0;i<m+n;i++) {
            System.out.print(nums1[i] + " ");
        }

    }
}
```

3 + 3 - 1 = 5

// p = m+n-1 (index in arr)

// post decrement operator

T = O(m+n)

S = O(1)

Q Kth Smallest Element in an Array

arr = [12 3 5 7 4 19 26]

k = 3

3th smallest Element ⟶ 5

(1) Brute ⟶ Sort ⟶ 3 4 5 7 12 19 26
0 1 2

K-1 ⟶ 2nd

T = O(n log n)

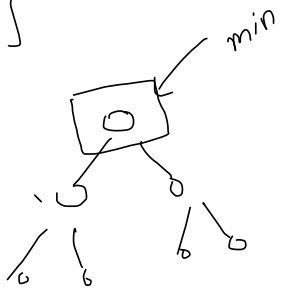
S = O(1)

Optimized

Unsorted 12 3 5 7 4 19 26

HashMap / Set → $T = O(N \log N)$, $S = O(N)$ } ✗

→ Heap (Priority Q) → min Heap



k = 3

3 min { min = 3 ✓, min = 4, min = 5 → $T = (n \log n)$

Heapify

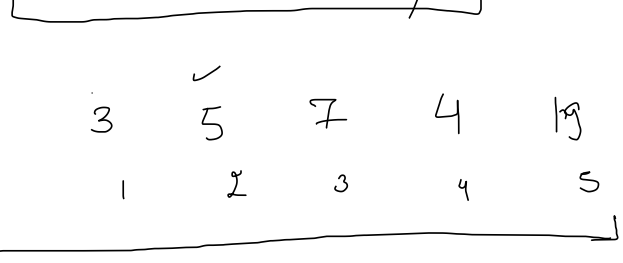
optimize → ($K \log N$)

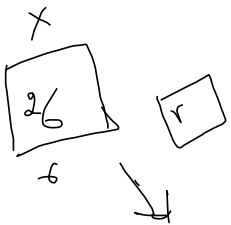
Quick Select - - - - - - -

Quick Sort

Pivot (Partitioning)

Pivot (Partitioning)





sort

if (arr[j] <= x) { swap i++ }

arr
0
6
k=3

26 3 5 7 4 19 12

3 26 5 7 4 19 12

5 26 7 4 26 19 26

k = 3 / o/p = 7

Pivot

10 1 7 2 25 11 14

k = 3

↓ Partitioning

(1 7 2) 10 (25 11 14)

4

Partitioning

↓

Put pivot element at its correct pos

k = 3

Pivot 1 7 2

↓ Partitin

pivot (2+1) 3

3 == k

pivot (2+1) < k → right

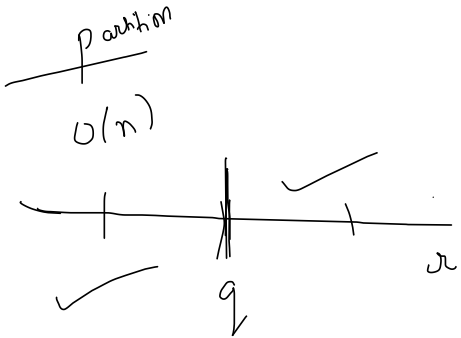
1 (7 2)

0 ↓ 2 Partitin

2 7

```java
package Arrays;

import java.util.Arrays;
import java.util.Collections;

class Kthsmallest {
    // Standard partition process of QuickSort.
    // It considers the last element as pivot
    // and moves all smaller element to left of
    // it and greater elements to right
    public static int partition(Integer[] arr, int l, int r)
    {
        int x = arr[r], i = l;
        for (int j = l; j <= r - 1; j++) {
            if (arr[j] <= x) {

                // Swapping arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;

                i++;
            }
        }

        // Swapping arr[i] and arr[r]
        int temp = arr[i];
        arr[i] = arr[r];
        arr[r] = temp;

        return i;
    }

    // This function returns k'th smallest element
    // in arr[l..r] using QuickSort based method.
    // ASSUMPTION: ALL ELEMENTS IN ARR[] ARE DISTINCT
    public static int kthSmallest(Integer[] arr, int l,
                                  int r, int K)
    {
        // If k is smaller than number of elements
        // in array
        if (K > 0 && K <= r - l + 1) {

            // Partition the array around last
            // element and get position of pivot
            // element in sorted array
            int pos = partition(arr, l, r);

            // If position is same as k
            if (pos - l == K - 1)
                return arr[pos];

            // If position is more, recur for
            // left subarray
            if (pos - l > K - 1)
                return kthSmallest(arr, l, pos - 1, K);

            // Else recur for right subarray
            return kthSmallest(arr, pos + 1, r,
                K - pos + l - 1);
        }

        // If k is more than number of elements
        // in array
        return Integer.MAX_VALUE;
    }

    // Driver's code
    public static void main(String[] args)
    {
        Integer arr[]
            = new Integer[] { 12, 3, 5, 7, 4, 19, 26 };
        int K = 3;

        // Function call
        System.out.print("K'th smallest element is " + kthSmallest(arr, 0, arr.length - 1, K));
    }
}
```

$$T(n) = \begin{cases} O(1) & n = 1 \\ T(n) + T(n-q) & n > 1 \\ \text{or} \\ T(n) + (q) \end{cases}$$



Best case $T(n) = O(n) + O\left(\frac{n}{2}\right)$

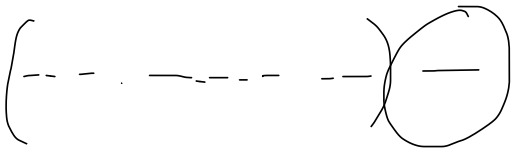
$= \boxed{O(n)}$

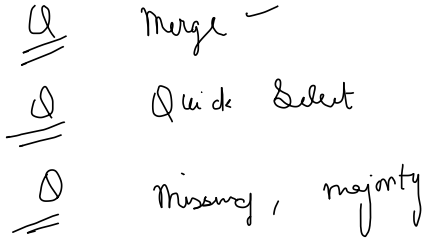
Space $= O(\log n)$

Worst Case $= T(n) = O(n) + O(n-1)$

$= O(n^2)$

N-1





Let's Summarize


THANK
YOU