Agenda : Learn all sorting algorithms.

Arrangement of data → Order → Asc / DESC

Property

Eg :

	A	B	C	D	E	F
PSP:	70	85	81	70	91	81

Asc Order ⇒ 1) A D C F B E ✓

by PSP ✓ × 2) A D F C B E

✓ × 3) D A C F B E

✓ × 4) D A F C B E

Stable Sort

Q Given an array of N distinct elements & K, find the Kth largest element.

K = 3

Eg A : 5, 8, 1, 3, 15, 9, 2

1) K = 1 (max)

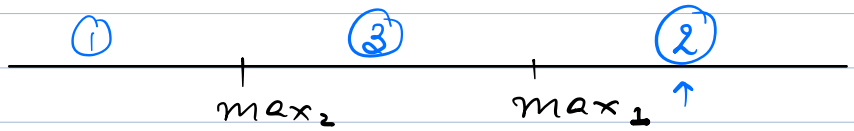
1 variable max = A[0]

Iterate over A & find max.

2) K = 2 (2 variables, max1, max2)

1st max 2nd max

$max_1 = max(A[0], A[1])$
$max_2 = min(A[0], A[1])$

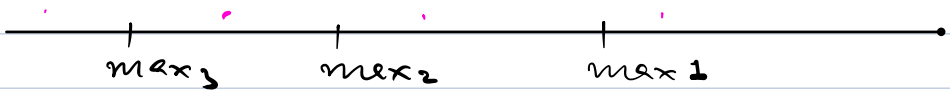


Iterate from i = 2 to N-1 {

```
if (A[i] < max2) {
        continue;
}
else if (A[i] > max1) {
        max2 = max1
        max1 = A[i];
}
else {
        max2 = A[i];
}
```

}

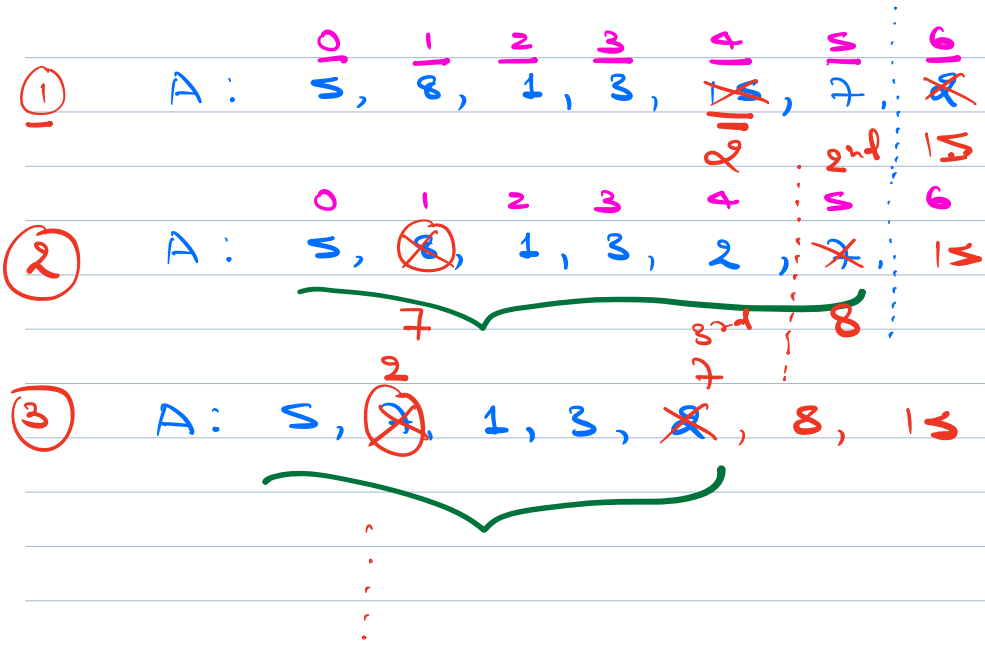
K = 3 ⇒ 3 variables



K ⇒ K variables

Const : No extra space

max



1	2	3	.....	K
N-1	N-2	N-3		N-K

Ans

$T.C. = O(N \times K)$

Q What happens if I repeat this process N-1 times.

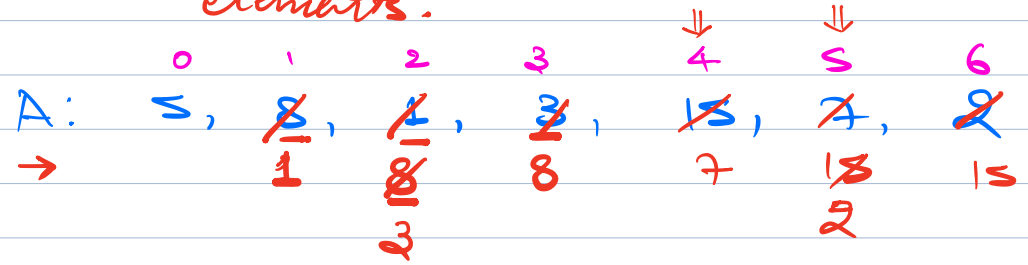
⇒ Array becomes sorted.

⇒ Selection Sort

$T.C. = O(N^2)$

Const² Only allowed to swap adjacent

elements.




```
for (i = 0; i < size-1; i++) {
    if (A[i] > A[i+1]) {
        temp = A[i];
        A[i] = A[i+1];
        A[i+1] = temp;
    }
}
```

Repeat (N-1) times ⇒ Sorted.

Bubble Sort

T.C. = $O(N^2)$

Playing cards.

7, 2, 10, 3, A, J, K, 4.

1) [7]

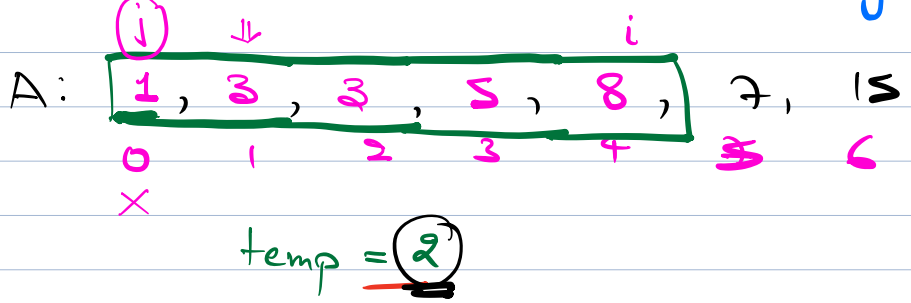
2) [2] [7]

Goal: Cards in hand should always be sorted.

3) [2] [7] [10]

4) [2] [2] [7] [10]

⇒ Let's extend this to array.



1) Find correct position to insert ⇒ O(N)

2) Shift the elements to the right ⇒ O(N)

3) Insert the element at correct position. ⇒ O(1

T.C. = O(N + N) = O(N)

Code

```
void InsertionSort(int arr[], int n){

    for(i=1; i<N; i++){
        temp = arr[i];
        int j;
        for(i = i-1; i>=0; i--){
```

```
        if (arr[j] > temp) {
            arr[j+1] = arr[j];
        }
        else {
            break;
        }
    }
    arr[j+1] = temp;
}
```

T.C. = $O(N^2)$

Q) Given an array of size N. Find the smallest subarray s.t. if we sort that subarray, the complete array becomes sorted. (ASC)

Eg A: 1, 3, 4, 2, 5, 6.

(indices: 0, -1, 2, 3, 4, 5)

2, 3, 4

Sorted: 1, 2, 3, 4, 5, 6

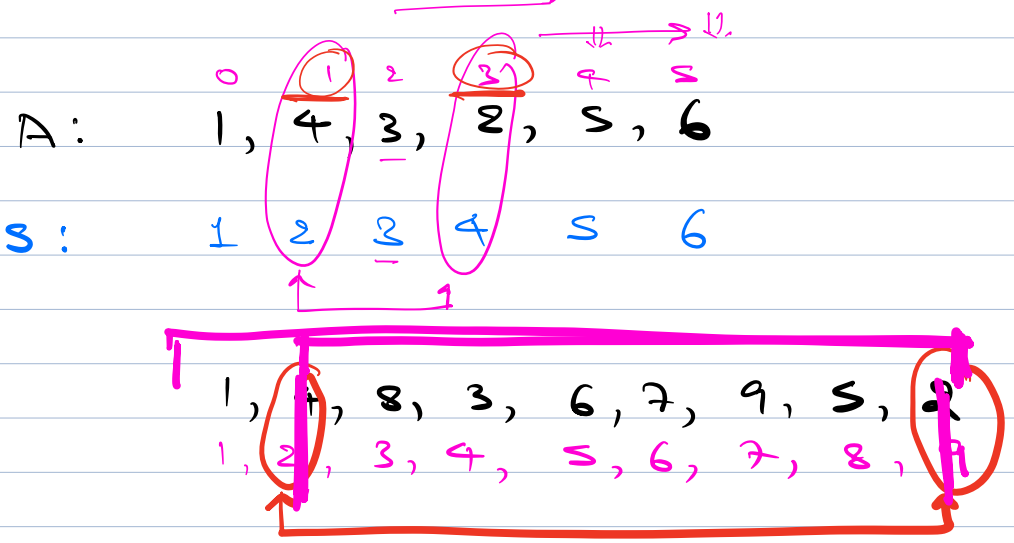
Soln 1) Create a copy & sort it

2) Match index by index.

T.C. = $O(N + N\log N + N)$

$= O(N \log N)$

S.C. = $O(N)$



$\log_2 4 = 2$, for $N > 4$,

$N \log N > 2N$

H.W. Try to solve this in O(1) S.C. & O(N) T.C.

Break till 10:44

Given an array of size N.
Sort the array.

⇒ All odd valued elements are present in sorted order

⇒ " Even " " " " "

" " " "

	0	1	2	3	4	5	6	7
A:	3	9	2	4	12	11	18	15
	O	O	E	E	E	O	E	O

Soln

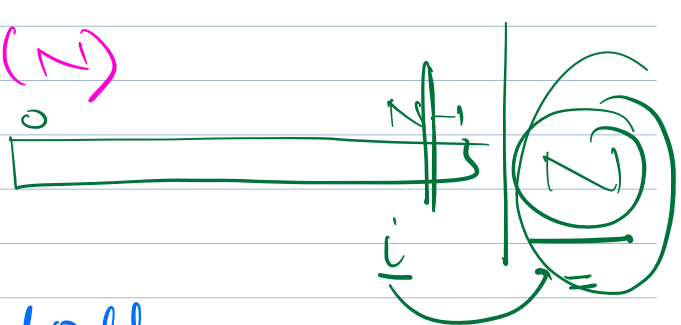
1) Use any sorting algo ⇒ TC. O(N log N)

2)

	0	1	2	3
Odd :	3	9	11	15
	i	i	i	i → i
Even :	2	4	12	18
	0	1	2	3 →
	j	j	j	j

Sorted : 2, 3, 4, 9, 11, 12, 15, 18

Orig array T.C. = O(N)



Code

Odd [N] countOdd = 0

Even [N] countEven = 0

Split

for (i = 0; i < N; i++) {

D

```
if ((A[i] & 1) == 1) {
        odd[countOdd] = A[i];
        countOdd++;
    }
    else {
        even[countEven] = A[i];
        countEven++;
    }
}
```

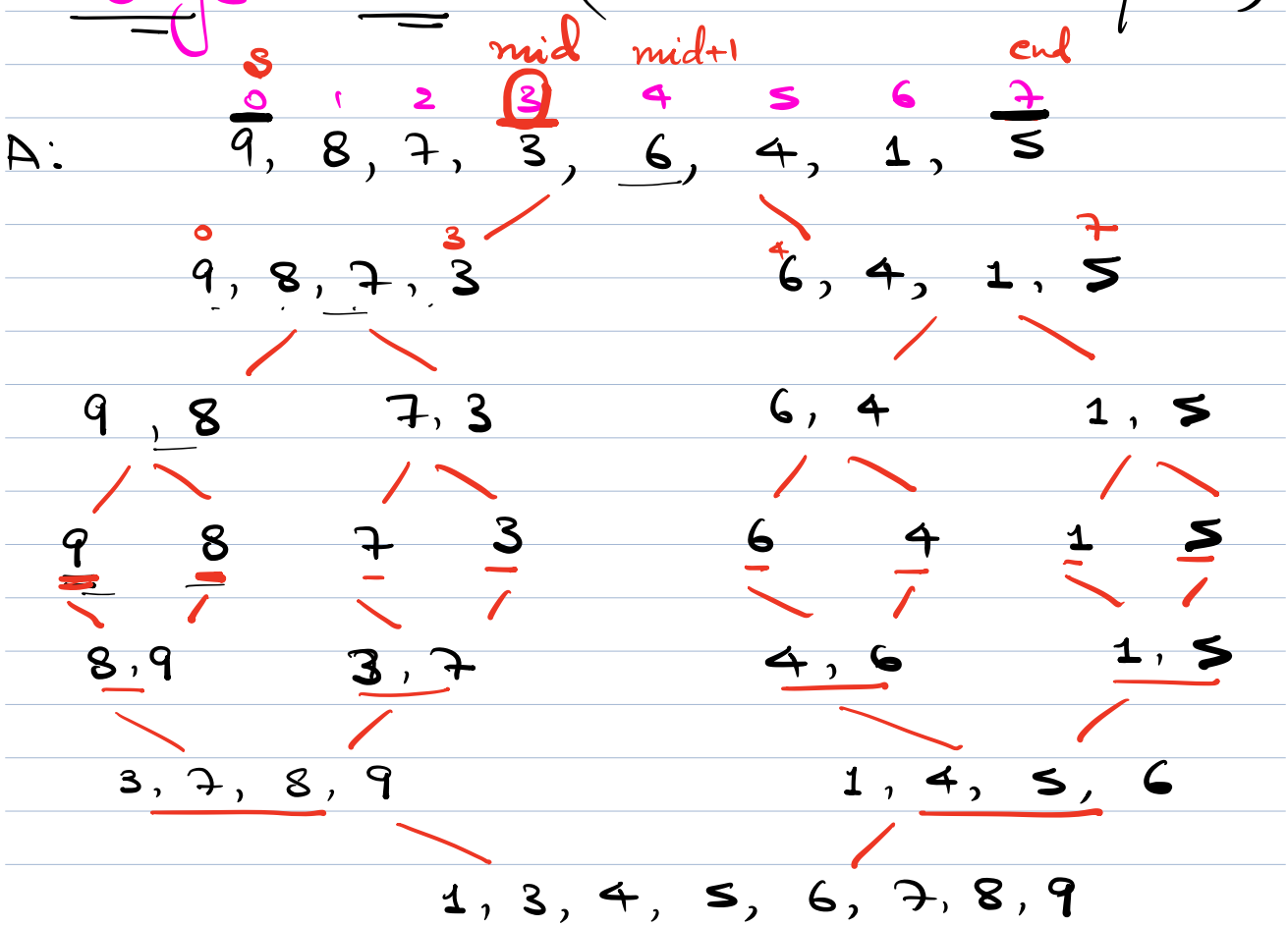
Merge

```
i = 0, j = 0;  , index = 0
while (i < countOdd && j < countEven) {
    if (odd[i] <= even[j]) {
        A[index] = odd[i];
        i++;
        index++;
    }
    else {
        A[index] = even[j];
        j++;
        index++;
    }
}

if (i < countOdd) {
    while (i < countOdd) {
```

```
            A[index] = odd[i];
            i++;
            index++;
        }
    }
    else {
        while (j < countEven) {
            A[index] = even[j];
            j++;
            index++;
        }
    }
}
```

Merge Sort (Divide & Conquer)



Code

```
void mergeSort( A[], 0, N-1 ) {
    if (s == e) { return; }
    mid = (s+e)/2        ( s + (e-s)/2 )
    mergeSort ( A[], s, mid);
    mergeSort ( A[], mid+1, end);
    merge ( A[], s, mid, end) ;
}

merge ( A[], s, mid, end) {
    // Merge A[s, mid] with A[mid+1, end]
    n1 = mid - s + 1
    n2 = end - (mid+1) + 1

    A1 [n1]
    A2 [n2]
    index = 0
    for (i = s; i <= mid; i++) {
        A1 [index] = A[i];
        index++;
    }
    index = 0;
```

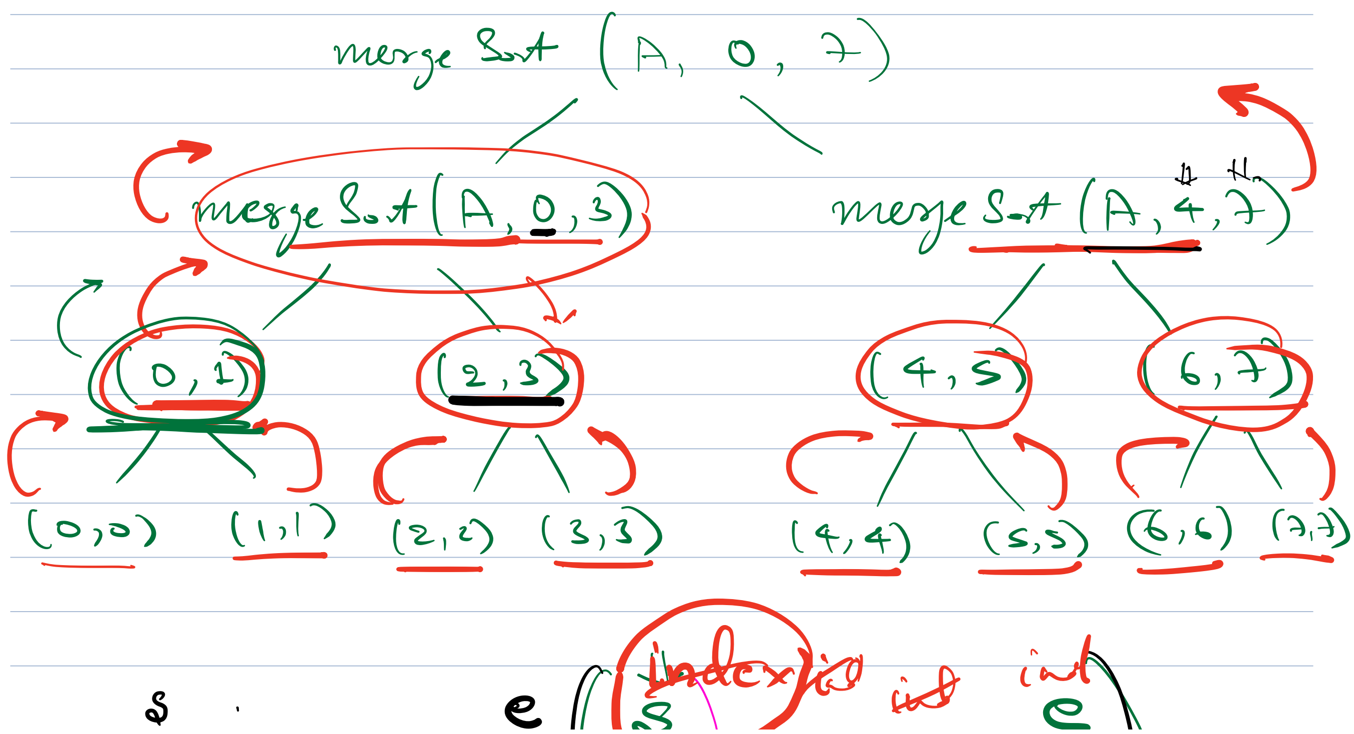
```
for (i = mid+1; i <= end; i++){
        A2[index] = A[i];
        index++;
}

i = 0;   j = 0;   index = s

    Remaining same as last problem.

}
```

H.W = T.C. of merge sort. (N log N)



3, 7, 8, 9, e, (4, 6, 1, 5), (1, 4, 5, 6)

A1 = 4, 6

A2 = 1, 5