⇒ Arrangement of Data , property , Order

Asc Desc

Eg ⇒ Aditya, Anuej, Dipen, Gaurav, Pankaj, Prem
70 85 81 70 91 81

Arrange in Asc by PSP

⇒ ① Aditya, Gaurav, Prem, Dipen, Anur. Pank.
70 70 81 81 85 90

② G Ad. Pr Dip An. Pan

③ G Ad Dip Pre An Pank.

④ Ad G Dip Pren, An, Pank

Stable Sort

Q Given an array of N distinct elemats. & K

find the Kth largest elemat.

⇒ $M_1 = 15$

⇒ $M_2 = 8$

Eg ⇒ A ⇒ 5, 8, 1, 3, 15, 9, 2 (N-1)

K = 1 ⇒ 15

sort ⇒ N log N

Iterate & find O(N)

K = 2 ⇒ Keep 2 variables

1st layer = max_1 = max (A[0], A[1])
2nd layer = max_2 = min (A[0], A[1])

```
for (i = 2; i < N; i++) {
    if (A[i] < max2) {
        // do nothing
        continue;
    }
    if (A[i] > max1) {
        max2 = max1
        max1 = A[i];
    }
    else {
        max2 = A[i];
    }
}

return max2;
```

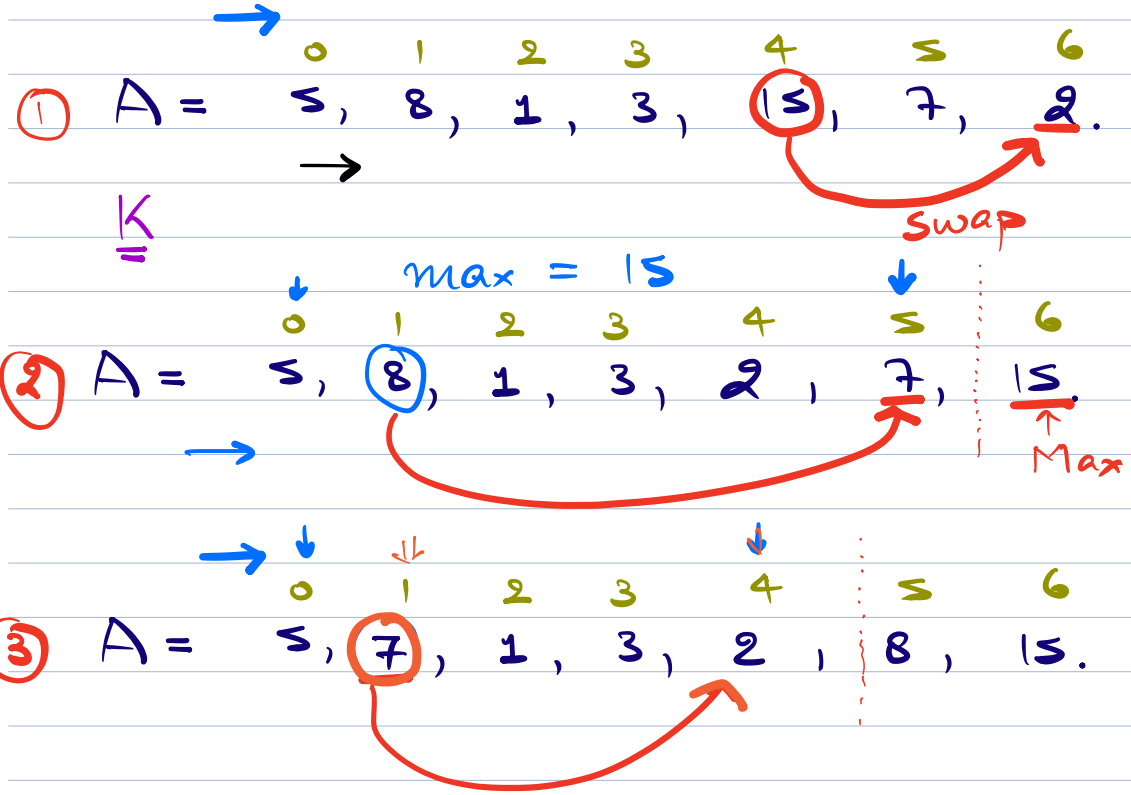
T.C. = O(N)
S.C. = O(1)

K = 3 ⇒ 3 variables

⋮

K variables

Const No extra space is allowed



Steps

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

1) Iterate & find max. (O(N))
2) Replace max with last index of active array region. O(1)
3) Repeat

⇒ Repeat the same process K times & get the K+th largest element.

T.C. = $O(n \times k)$

Q What happens if I repeat the above app. N-1 times.

⇒ Array becomes sorted.

⇒ Selection Sort.

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

Const2 Only allowed to swap adjacent elements.

	0	1	2	3	4	5	6
A ⇒	5,	~~8~~ 1,	~~8~~ 3,	~~8~~,	~~15~~ 7,	~~7~~ 8 2,	~~2~~ 15.

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

Repeat k times ⇒ A[N-k] ⇒ k^{th} largest ele.

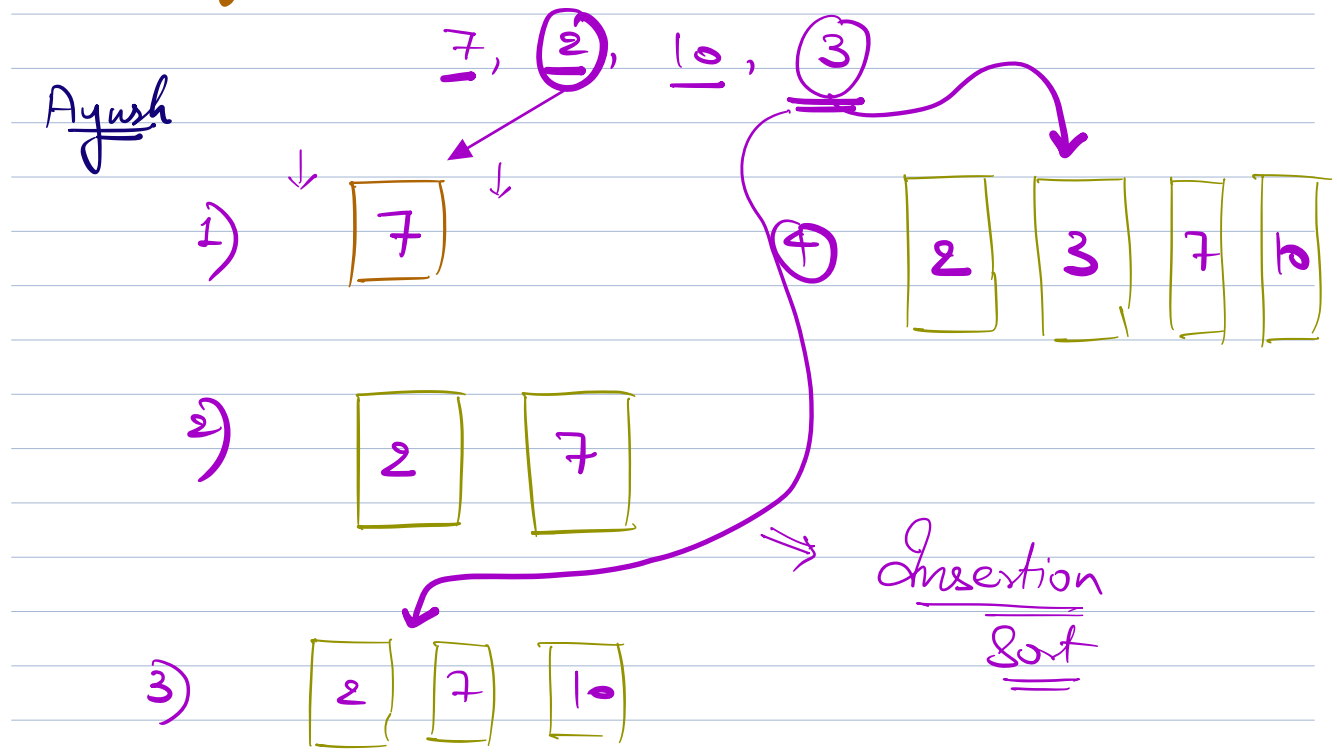
Repeat (N-1) times ⇒ Sorted Array

⇓

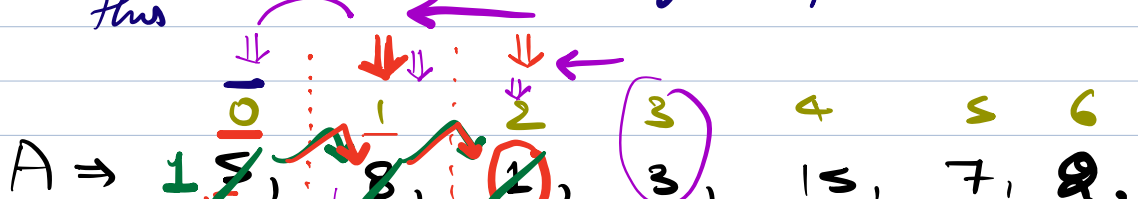
Bubble Sort.

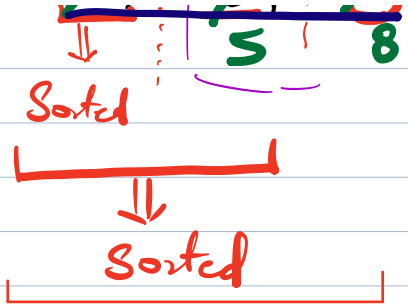
T.C. = $O(N^2)$

Playing cards ⇒ Shubham is dealing.



⇒ let's take an array input to understand this





1) Find the correct position to insert the new ele. ⇒ O(N) ⇒ O(logN) → BS

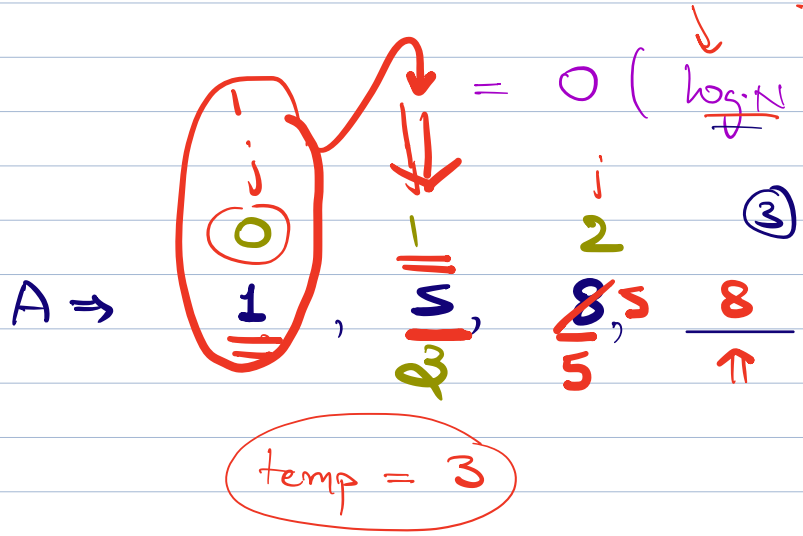
2) Shift the elemats to make space ⇒ O(N)

3) Insert at correct position. ⇒ O(1)

T.C. = O(N + N)

Search Shift

= O(logN + N) =



Code

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

```
⇐ for ( i=1;  i<n;  i++) {
        temp =  arr[i];
        int j
→   for ( j= i-1;  j >= 0;  j--) {
            if (arr[j] > temp) {
                arr[j+1] = arr[j];
            }
            else {
                break;
            }
        }
        arr[j+1] = temp;
    }
```

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

{1, 2, 3, 4, 5}

A

A ⇒ 5, 4, 3, 2, 1

Asc

1 + 2 + 3 + 4 + 5

Q) Given an array of size N.

Find the smallest subarray s.t. if we sort that subarray, the complete array is sorted.

Eg A ⇒ 1, 3, 4, 2, 5, 6

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

2, 3, 4

s =

e =

Soln

1) Create a copy & sort it

2) Compare the copy & original array

	0	1	2	3	4	5
A =	1,	3,	4,	2,	5,	6
CSA ⇒	1,	2,	3,	4,	5,	6.

→

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

S.C = $O(N)$

H.W Try to solve in T.C. $(O(N))$!!

next levels

Q Given an array of size N.

⇒ Sort the given array.

⇒ All odd valued elements are present in sorted order

⇒ " Even " " " " "

" " "

Eg ⇒ 3, 9, 2, 4, 12, 11, 18, 15

O O E E E O E O

Sol^n ⇒ Use qsort / Msort ⇒ O(NlogN)

2) i) Split the array into odd & even

(i) Odd ⇒ 3, 9, 11, 15 — S.C. O(N)

(j) Even ⇒ 2, 4, 12, 18 — T.C. O(N)

$S_1 + S_2 = N$

2) 2 3 4 9 11 12 15

0 1 2 3 4 5 6 7

index index index index

⇒ Same can be extended to merge any 2 sorted array.

Code

A ⇒ Input array //

N ⇒ size of A

odd[N] Even[N] ⇒ declare

countOdd = 0 countEven = 0;

// Split.

```
for( i=0; i<N; i++) {
        if( (A[i] & 1) == 1) {
                odd[countOdd] = A[i];
                countOdd++;
        }
        else {
                even[countEven] = A[i];
                countEven++;
        }
}
```

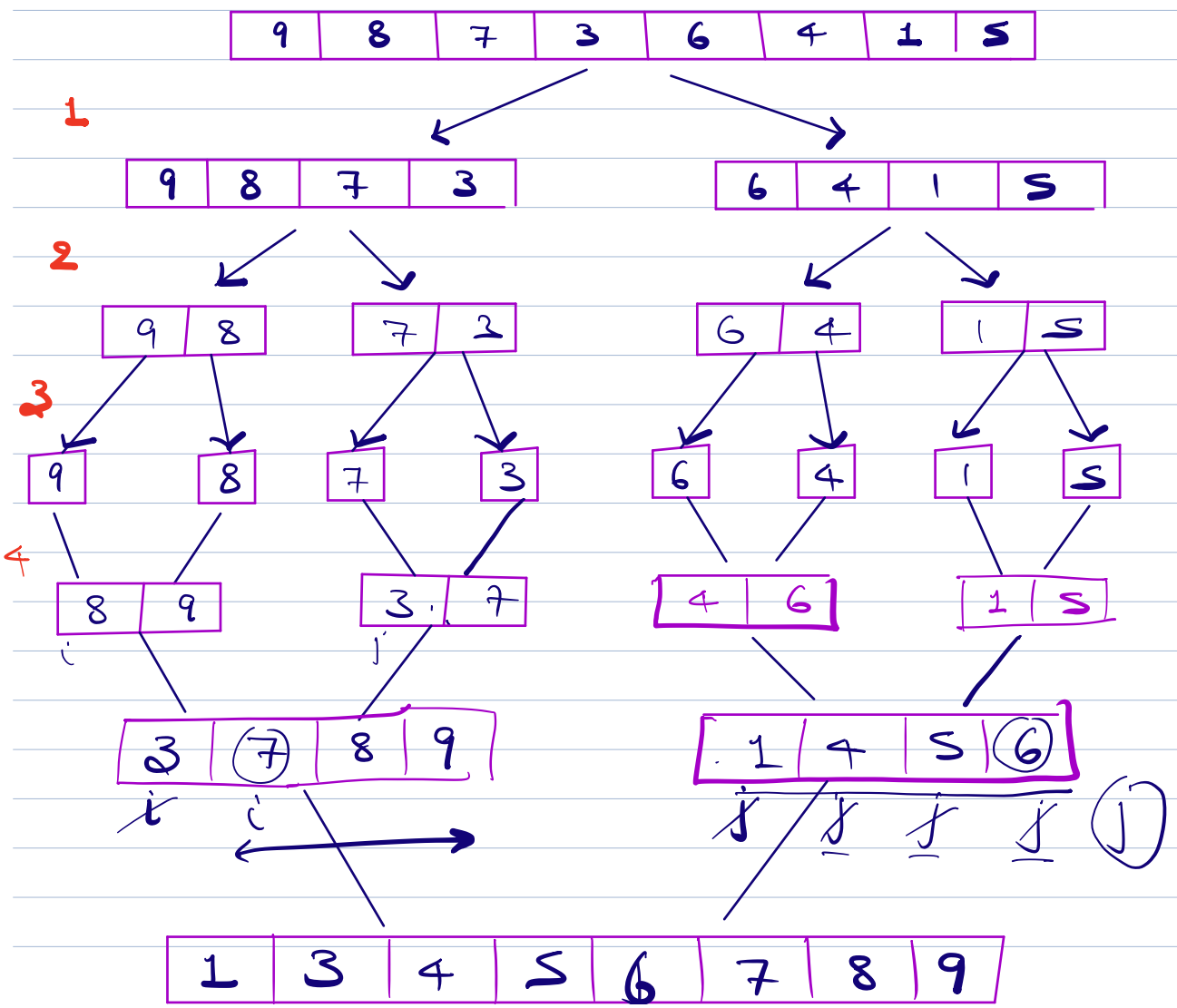
i = 0 , i = 0 , index = 0;

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

T.C. = O(N)

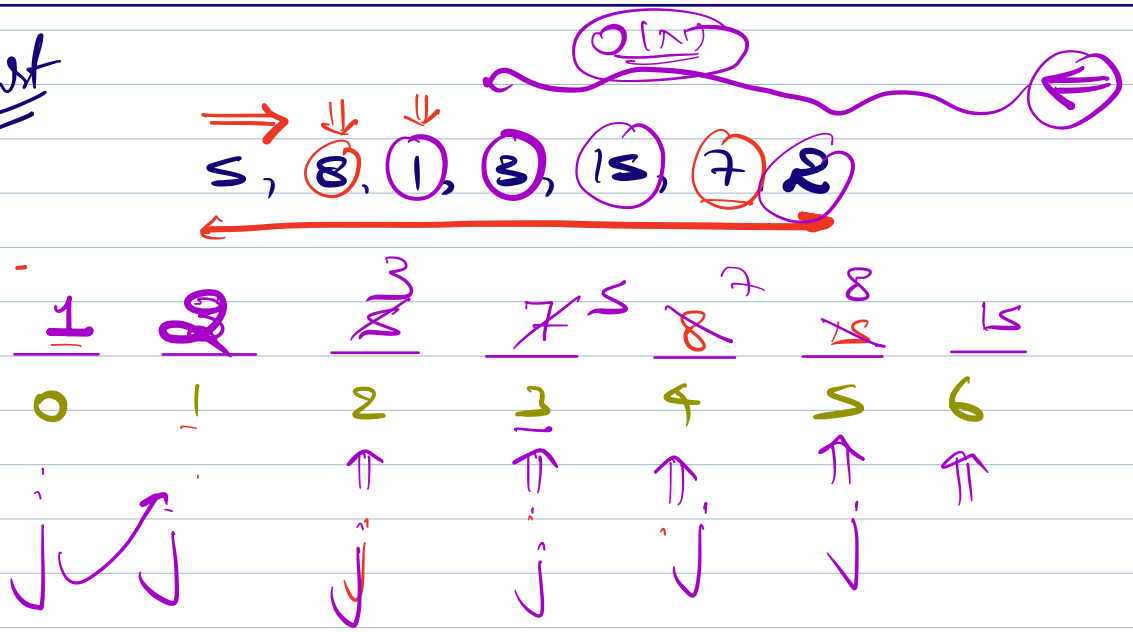
S.C. = O(N)

Merge Sort (Divide & Conquer)

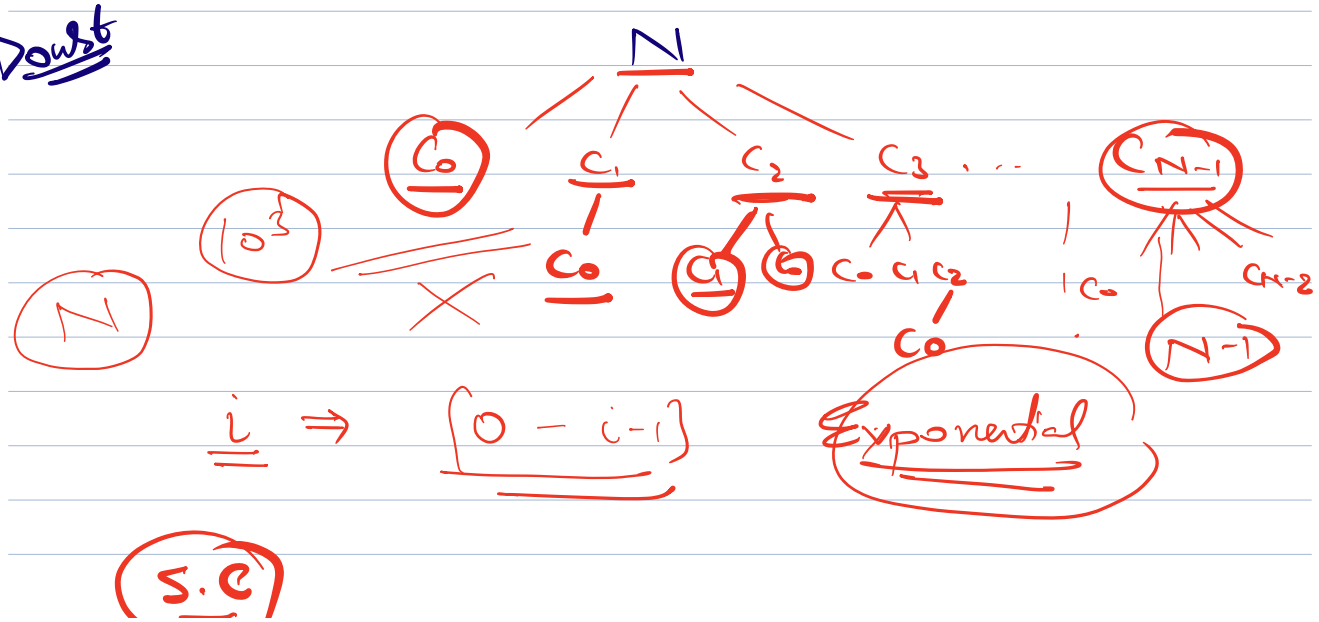


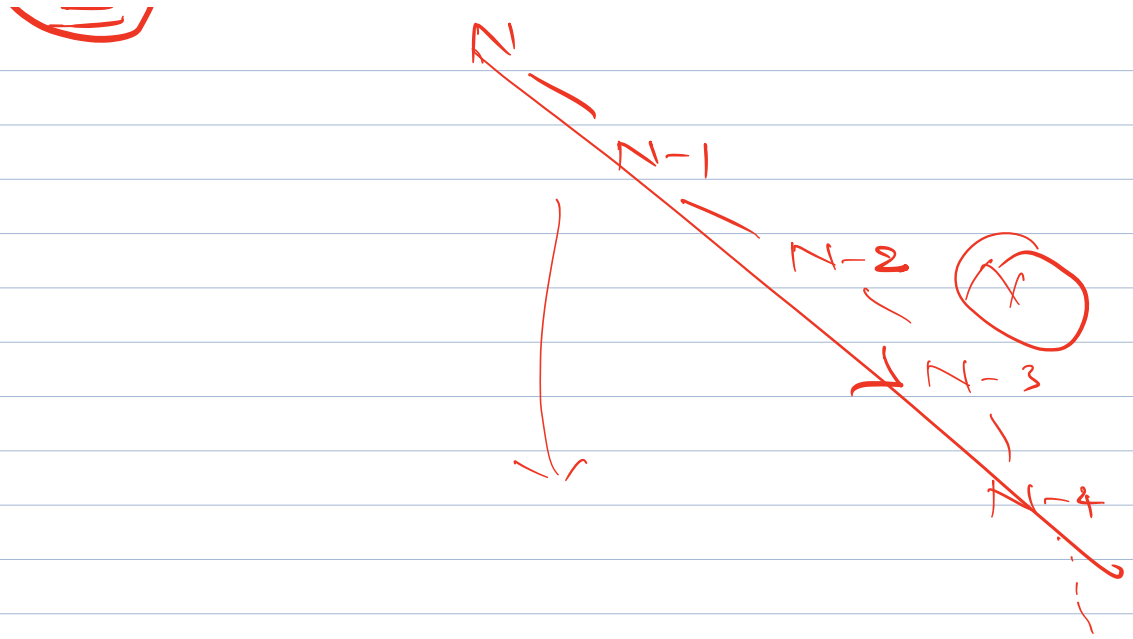
H.W. Try to write recursive code of merge sort

Doubt



Doubt




N
N-1
N-2
N-3
N-4