Q.1 Given an Array of size N, find the k^{th} minimum element. ($k < \log N$)

A : { 1 , 5 , -1 , 2 , 10 , 3 }

k = 2 ⇒ 1
k = 3 ⇒ 2
k = 5 ⇒ 5

Idea 1 :- Sort the array & return A[k-1]

A : { 1 , 5 , -1 , 2 , 10 , 3 }
(0 1 2 3 4 5)

↓ Sort

{ -1 , 1 , 2 , 3 , 5 , 10 }
(0 1 2 3 4 5)

→ A[k-1]

TC: O(N log N)

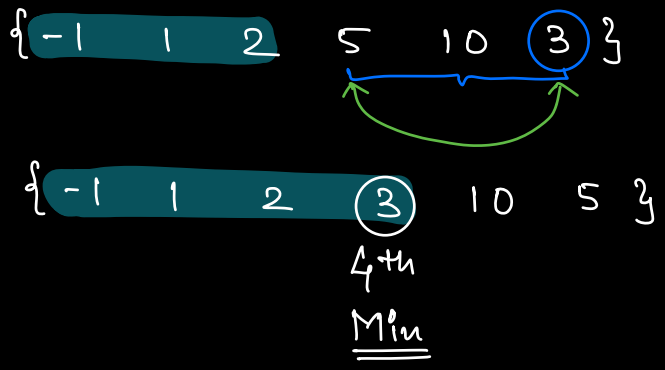
⇒ A : { -1 , 4 , 2 , 3 , 1 , 2 , 5 , 2 }

A : { 1 , 5 , -1 , 2 , 10 , 3 }
(0 1 2 3 4 5)

k = 4

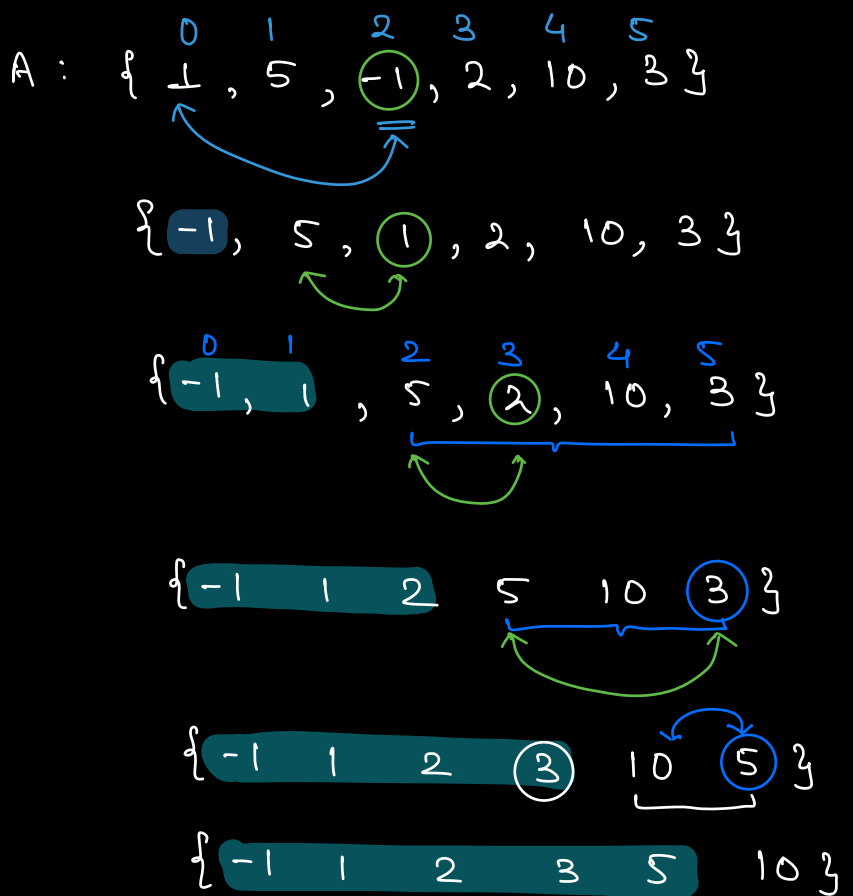
{ -1 , 5 , 1 , 2 , 10 , 3 }

{ -1 , 1 , 5 , 2 , 10 , 3 }
(0 1 2 3 4 5)



TC : $O(K \cdot N) < O(N \log N)$

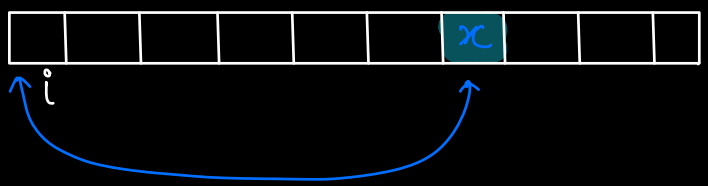
SC : $O(1)$



* If we repeat the above process N times, We'll get SORTED array.

⇒ Selection Sort

Select the minimum element & replace it with it's correct position element.




```
for( i = 0; i < N; i++ ) {
    min_ele = A[i]
    min_indem = i;
    for( j = i+1; j < N; j++ ){
        if ( A[j] < min_ele ) {
            min_ele = A[j];
            min_indem = j ;
        }
    }
    Swap ( A[i], A[min_indem] );
}
```

A : { (-1), 5, (-1), 2, 10, 3 }

{ -1, 5, 1, 2, 10, 3 }

i = 0 → 1

m = ~~1~~ → ~~5~~ 1

ind = 0 (2) ~~1~~ 2

{ -1, 1, 5, 2, 10, 3 }

TC : $O(N^2)$

SC : $O(1)$ ⇒ Inplace.

1) Inplace : SC : $O(1)$
   ↳ NO Extra space.

2) Stable :

{ 1 4 2 3 2 6 5 }

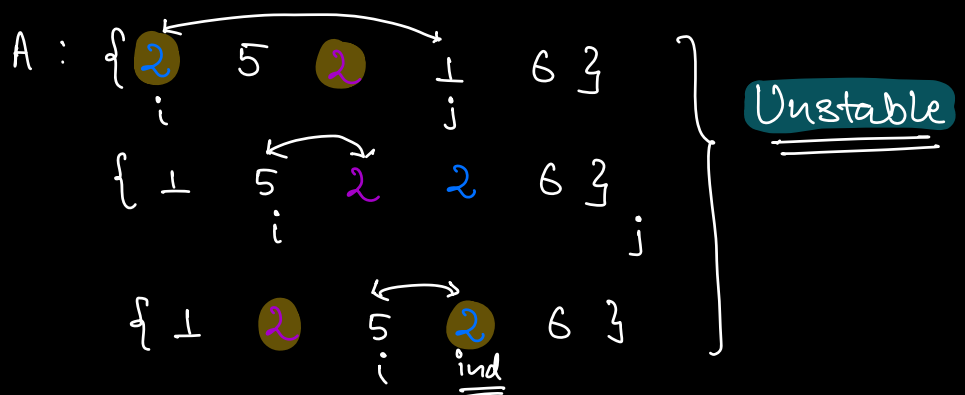
↓ sort ⇒ Stable.

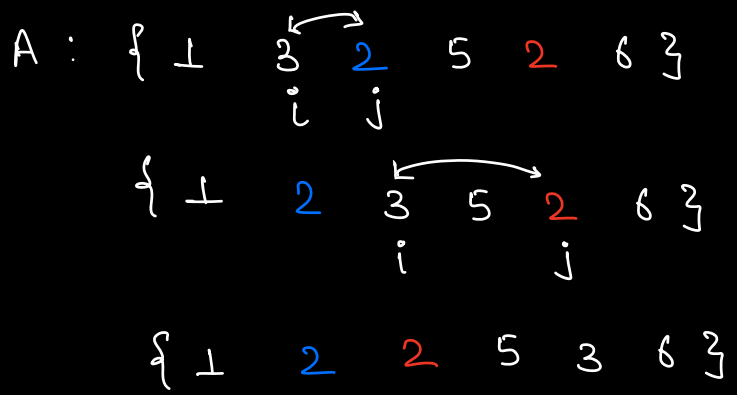
{ 1 2 2 3 4 5 6 }

duplicate

→ If the relative order of the elements isn't changing after sorting them that Algo is called Stable Sorting algo.

Quiz Maximum no. of swaps in Selection Sort.

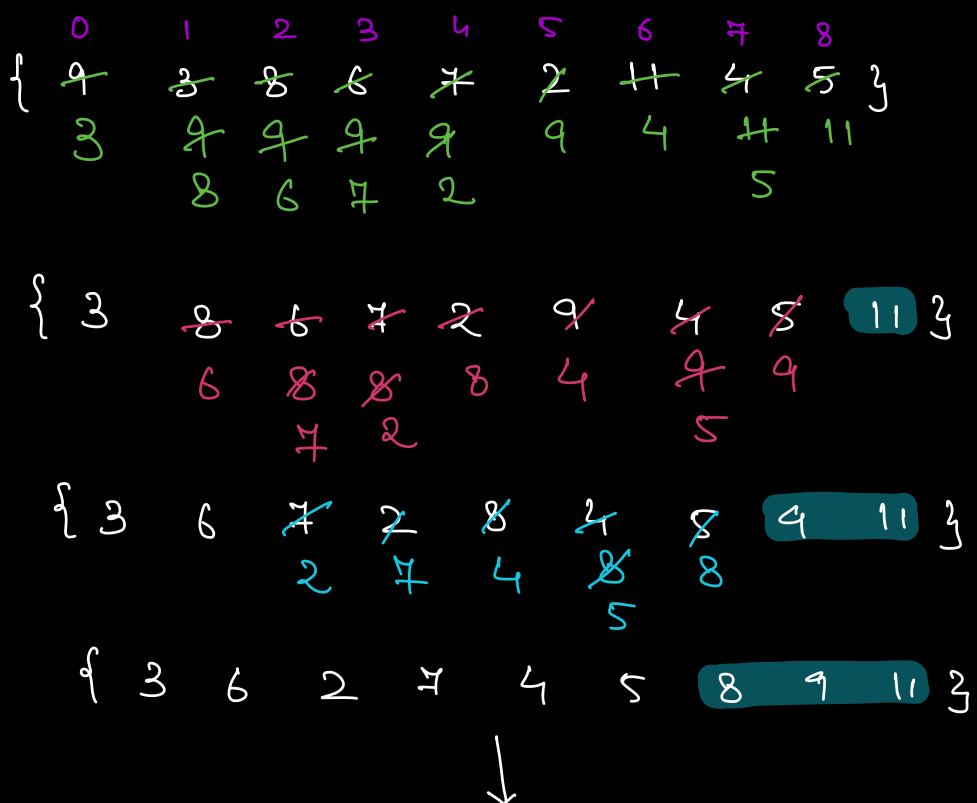




HW Try to implement Stable version of Selection Sort.

#

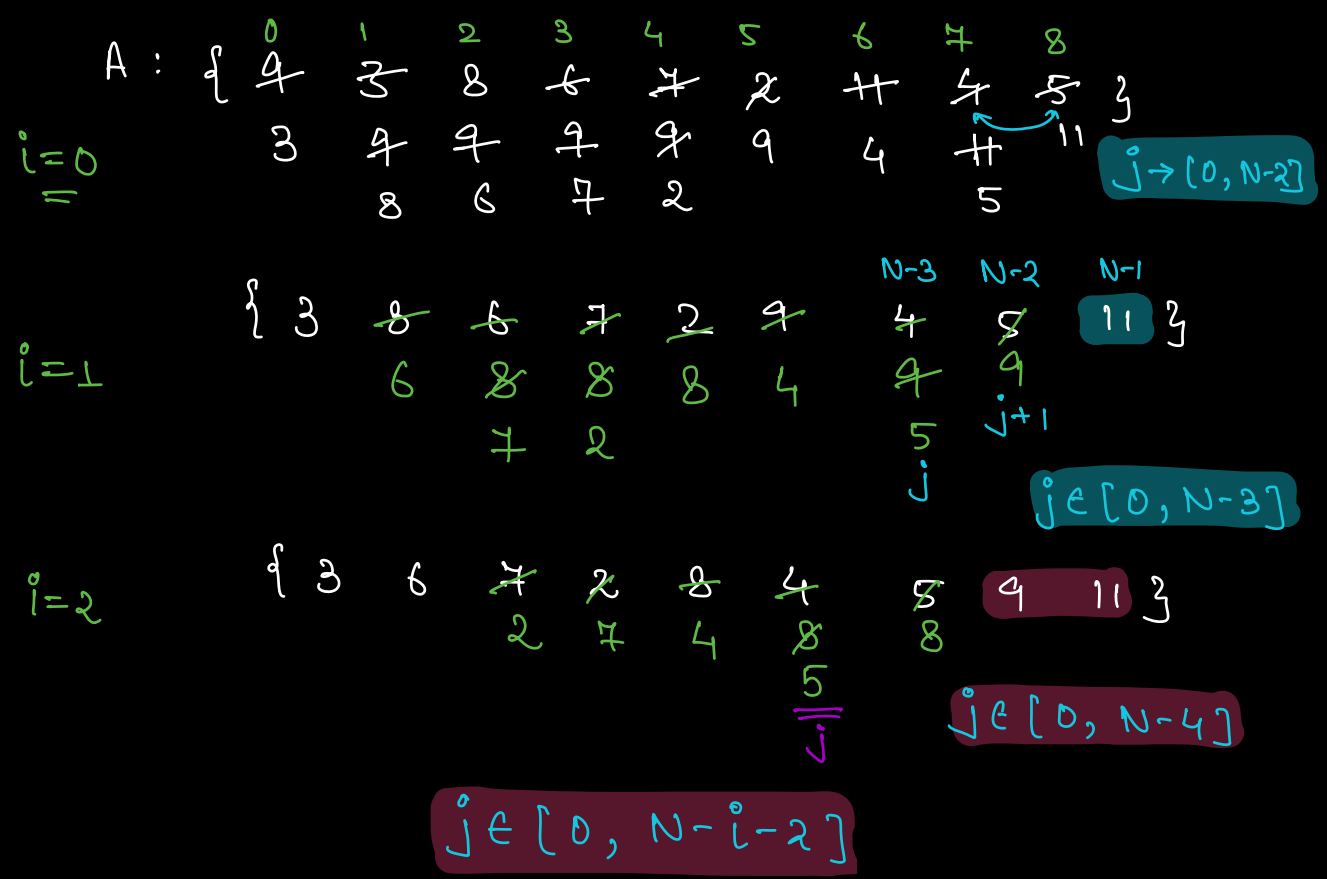
Swapping the pos^n with non consecutive elements isn't allowed.



⇒ If we repeat the above process N times, we'll get sorted array.

⇒ BUBBLE SORT

```
for(i=0; i< N; i++) {
    for(j= 0; j<= N-i-2; j++ ){
        if( A[j] > A[j+1] )
            swap( A[j], A[j+1] );
    }
}
```



TC : $O(N^2)$

i	j
0	[0, N-2] ⇒ N-1
1	[0, N-3] ⇒ N-2
⋮	⋮
N-1	⇒ 0

$$\underbrace{(N-1) + (N-2) + \cdots + 1}_{\frac{N(N-1)}{2}}$$

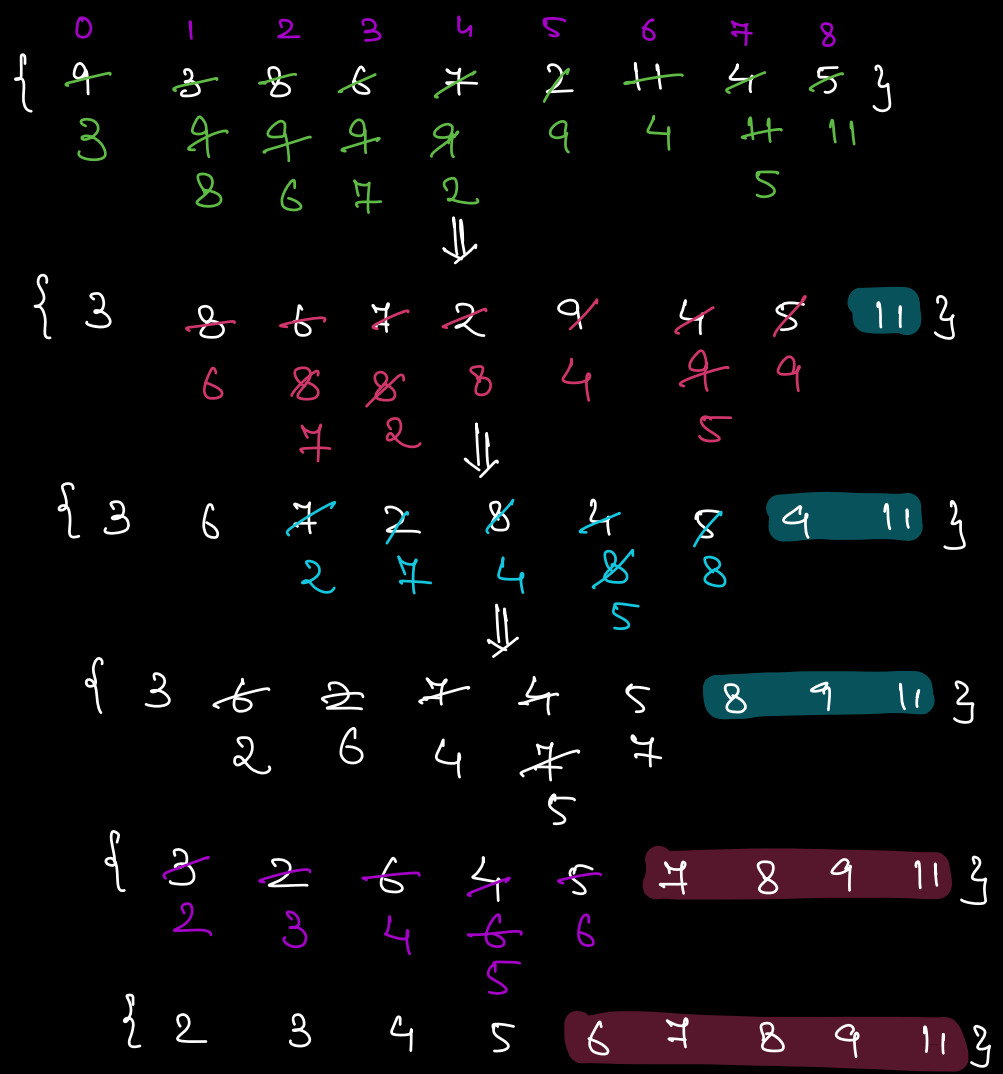
SC : $O(1)$ ⇒ Inplace.

A : {2 5 2 1 6}

↓

A : {1 2 2 5 6}

Stable.



⇒ In any iteration, if there's NO swap happening then Array has already become sorted.

Q: Merge 2 **Sorted Arrays**.

⇒ Given 2 sorted arrays of size M & N, Merge them into a single sorted array.

A: { 2 5 7 12 20 24 29 } → M

B: { 6 9 10 14 18 19 } → N

C: { 2 5 6 7 9 10 12 14 18 19 20 24 29 }

↳ M+N

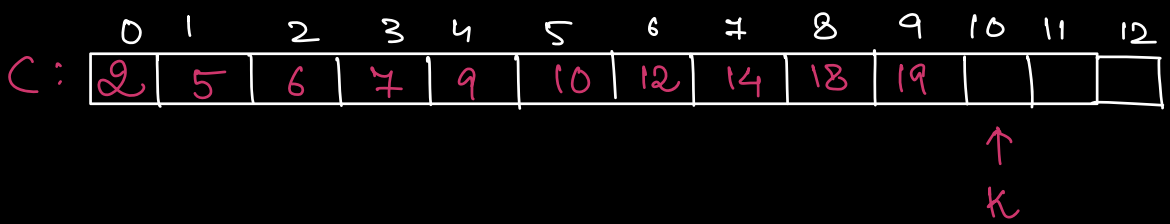
`int C[N+M]`

$$O(M) + O(N) + O((M+N)\log(M+N))$$

A: { 2 5 7 12 20 24 29 } M = 7
i

B: { 6 9 10 14 18 19 } N = 6
j

C[k] = min(A[i], B[j])



```
int[] merge(A[], M, B[], N){
    C[N+M];
    i=0, j=0, k=0
    while(i<M && j<N){
        if(A[i]<B[j]){
            C[k] = A[i];
            i++, k++
        }
        else{
            C[k] = B[j]
            j++, k++
        }
    }
    while(i<M){
        C[k] = A[i];
        i++, k++
    }
    while(j<N){
        C[k] = B[j];
        j++, k++
    }
    return C;
}
```

C[k++] = A[i++]

A = [1,2,3] B = [4,5,6] so
TC : O(A+B)

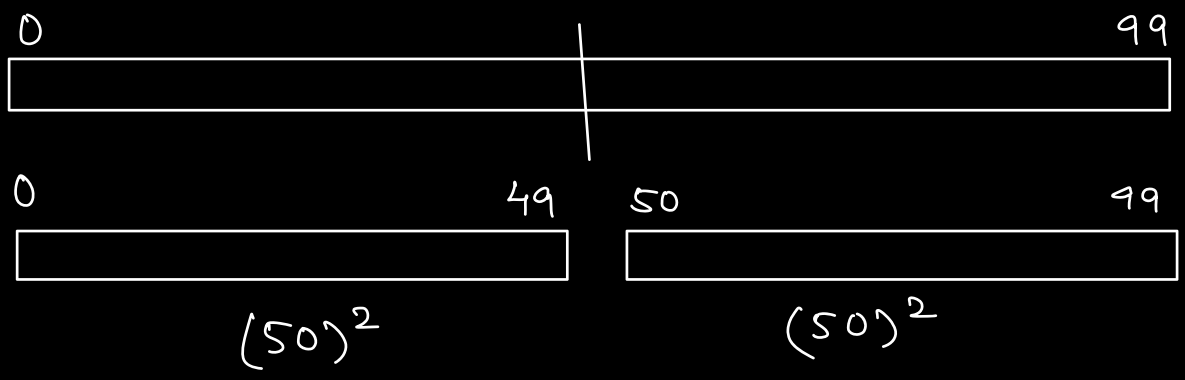
TC : O(N+M)

SC : O(N+M) { Storing the ans. array }

→ Selection } ⇒ $O(N^2)$
→ Bubble

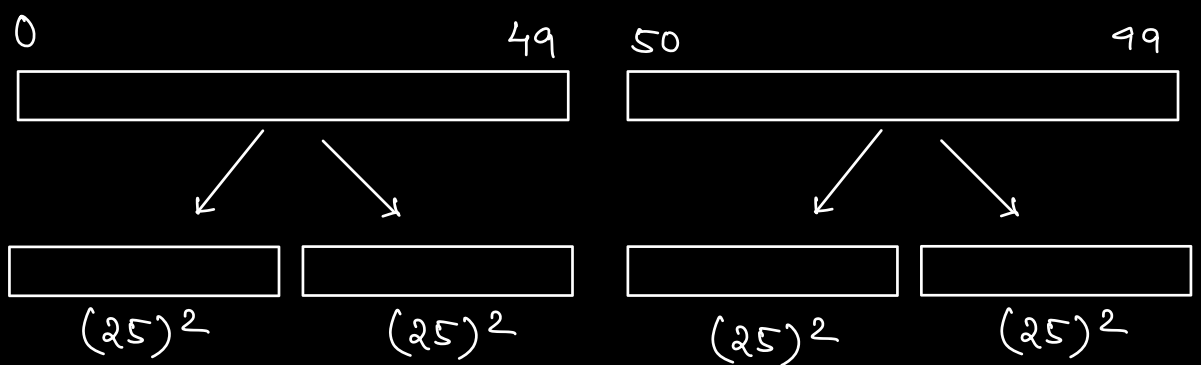
int a[100]

⇒ $(100)^2$ → 10000



iterations = $(50)^2 + (50)^2 + (50+50)$

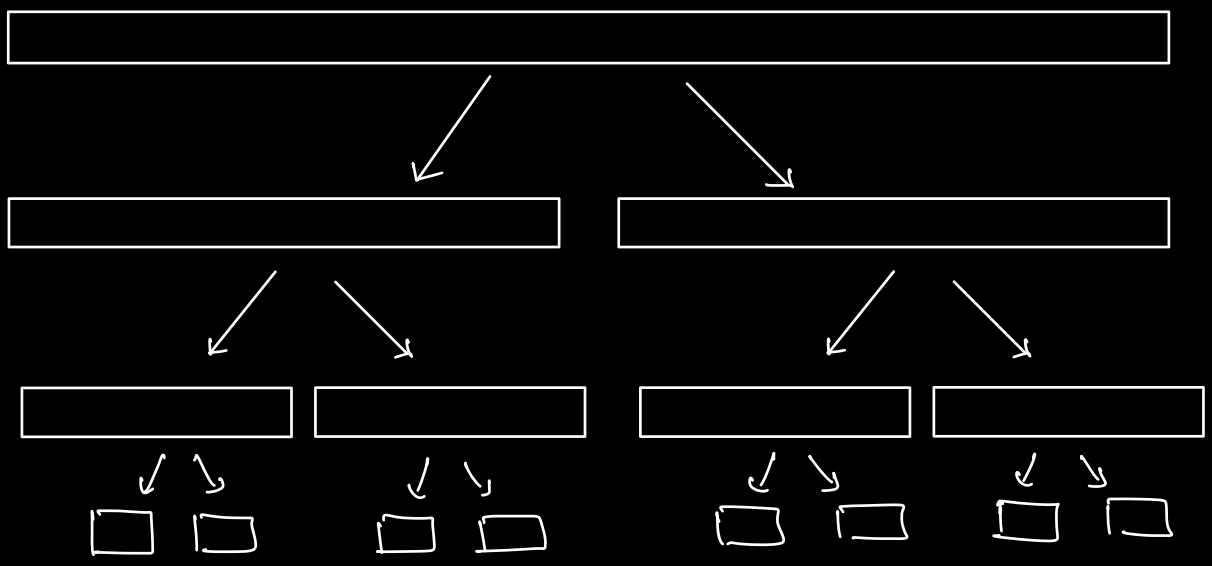
= 5100



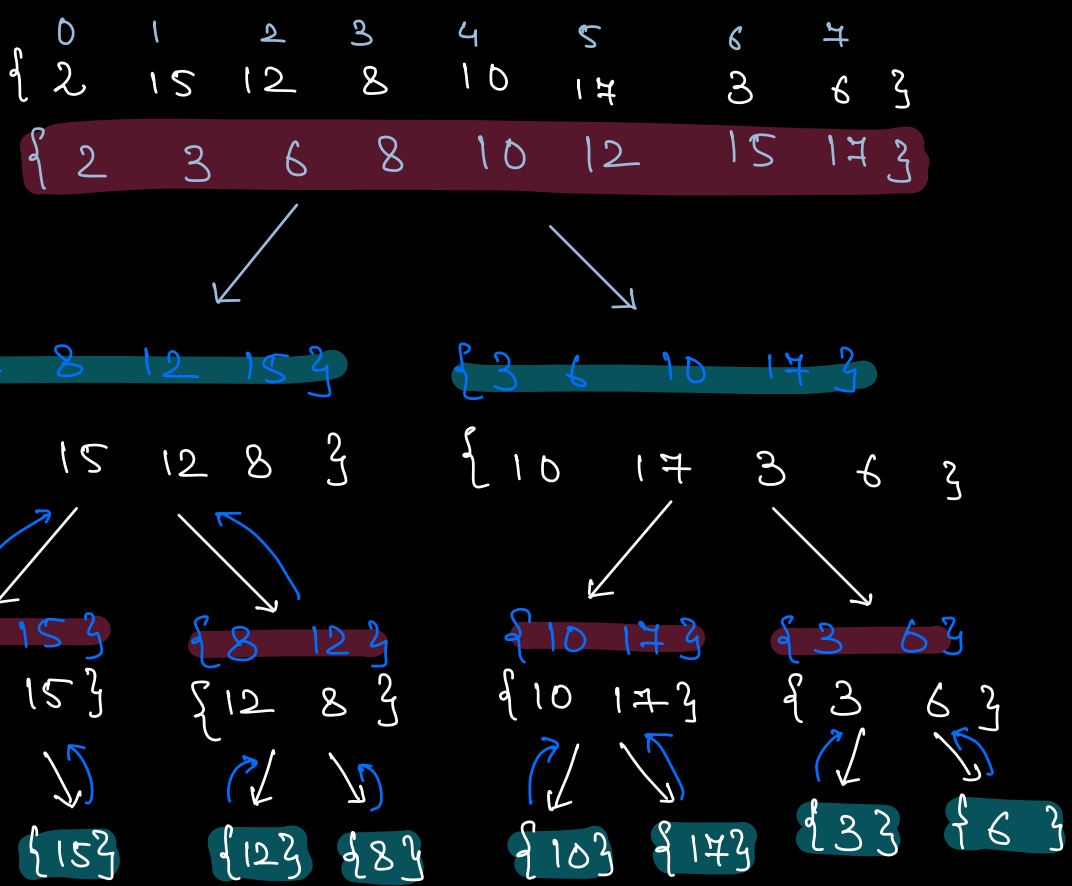
iterations = $4*(25)^2 + (25+25) + (25+25) + (50+50)$

= 2700

10000 → 5100 → 2700

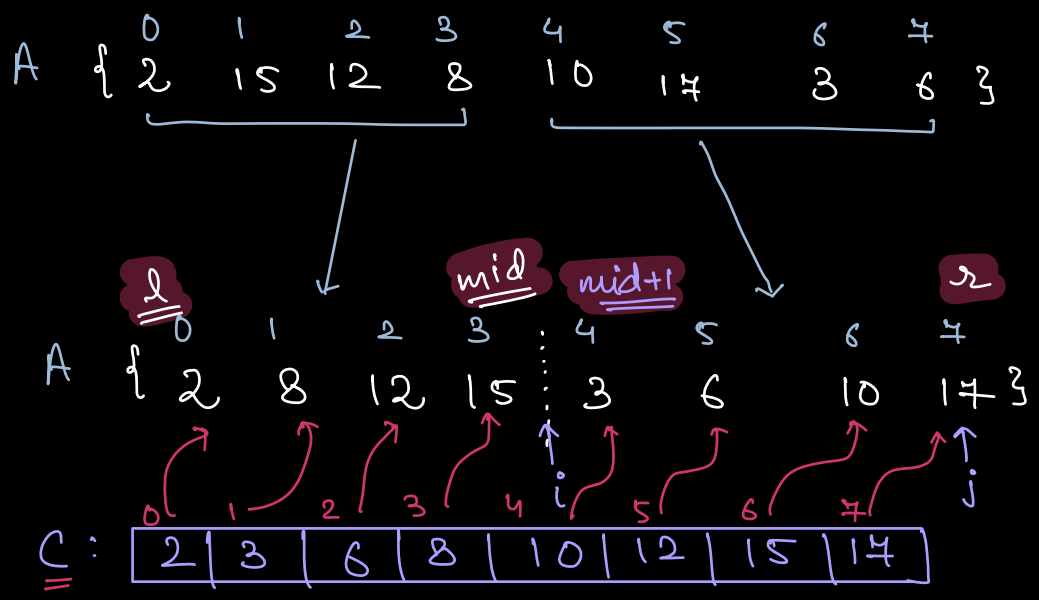


MERGE SORT (Divide & Conquer)



→ Recursion

```
                                    0   N-1
                                    ↑   ↑
Void  mergeSort ( A[] , l , r) {
   // Assumption : mergeSort(A, x, y) fun^
   // sorts the Array from x to y
   if ( l == r ) return ;
   mid = (l+r)/2
   mergeSort( A, l, mid );
   mergeSort( A, mid+1, r );
   merge( A, l, mid, r );
}
```



ms(A, 0, 7)

↳ m = 3

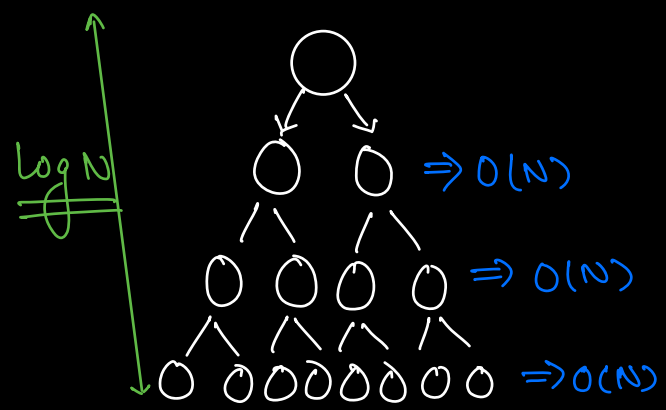
→ ms(A, 0, 3)

→ ms(A, 4, 7)

→ merge(A, 0, 3, 7) → 1st part : 0 to 3, 2nd part : 4 to 7

$T(N) = 2T(N/2) + O(N)$

TC : $O(N \log N)$



SC: $O(\log N) + O(N) \rightarrow O(N)$

Recursive stack

Merge fun

$O(N + \log N)$

Inplace ✗

HW Stability

———— * ————

#Merge two sorted sub arrays from l to y & y+1 to r.

```
merge ( A[], int l, int y, int r ) {
    i = l
    j = y+1
    k = 0
    int C[r-l+1]
    while( i<=y && j<=r ) {
        if( A[i] < A[j] ) {
            C[k] = A[i];
            i++, k++;
        }
        else {
            C[k] = A[j];
            j++, k++;
        }
    }
    while( i<=y ) {
        C[k] = A[i]
        i++, k++;
    }
    while( j<=r ) {
        C[k] = A[j]
        j++, k++;
    }
```

// copy the array (C) into Array (A)

```
for(x = l; x <= r ; x++) {
        A[x] = C[x-l];
}
}
```

Doubts

A: 1 2 3 (i)

B: 3 4 5 (j)

C[6]: {1 2 3 3 4 5}