Today's Content

- → Insertion Sort
- → Merge 2 sorted arrays
- → Merge Sort

Q1: Given a[N], first N-1 elements are sorted. Sort entire a[].

Eg: a[6] = { 2 6 10 14 20 4 }

0	1	2	3	4	5
2	6	10	14	20	4
2	4	6	10	14	20

idea: Arrays.Sort() → TC: O(nlogn)

a[6] = { 2 6 10 14 20 4 }

0	1	2	3	4	5
2	6	10	14	20	4
	4	4	4	4	20
		6	10	14	

TC: O(n)

Idea: Iterate from back & compare adjacent elements. If not in correct order, swap!

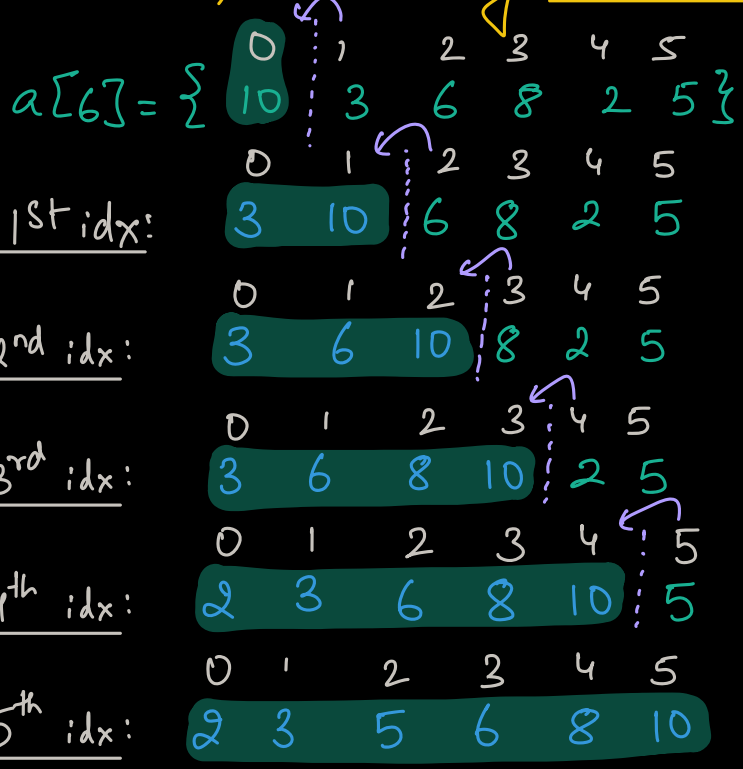
Pseudocode: Given a[N], first N-1 are sorted.

```
for(j = n-2; j >= 0; j--){
    if(a[j] > a[j+1]){
        swap(a[j], a[j+1])
    }
    else{
        break
    }
}
```

TC: O(N), SC: O(1)

Note: To insert a single element in sorted data by swapping adjacent elements to make entire data sorted ⇒ Insertion step.

Q2: Given a[N], Sort it using Insertion Step → Insertion Sort.




```
void iSort(int a[N]){
    for(i=1; i<N; i++){
        for(j=i-1; j>=0, j--){
            if(a[j] > a[j+1]){
                swap(a[j], a[j+1])
            }
            else{
                break
            }
        }
    }
}
```

TC: $O(N^2)$
SC: $O(1)$

Best - Case
TC: $O(N)$

Qn: Given 2 sorted arrays a[N], b[M]. Merge & create a new sorted array.

Eg: a[3] = {-1, 4, 8}
b[2] = {2, 9}
c[5] = {-1, 2, 4, 8, 9}

idea 1: a[N], b[M]. Create c[N+M]
1) Copy a[] to c[] → O(n)
2) Copy b[] to c[] → O(m)
3) Sort c[] → O[(N+m) log (N+m)]

idea 2:

	0	1	2	3	4	5	6
a[7] = {	-5,	-1,	3,	7,	10,	12,	15 }

p1

	0	1	2	3	4
b[5] = {	-4,	0,	2,	8,	9 }

p2

	0	1	2	3	4	5	6	7	8	9	10	11
c[12] = {	-5	-4	-1	0	2	3	7	8	9	10	12	15 }

p3

// Pseudo code:

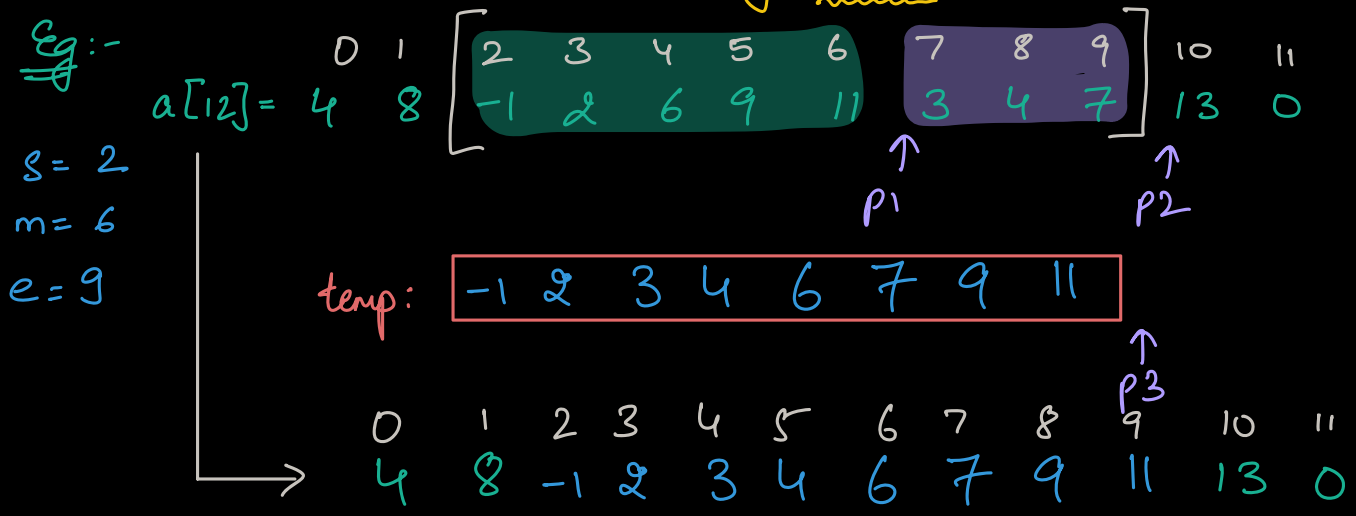
```
int [] merge (int a[], int b[], int n, int m) {
    p1 = 0, p2 = 0 , p3 = 0
    int c [n+m]
    while ( p1 < n && p2 < m ) {
        if (a [p1] <= b[p2]) {
            c[p3] = a [p1]
            p1++, p3++
        }
        else {
            c[p3] = b[p2]
            p2++, p3++
        }
    }
    // Copy rest of the elements.
    while( p1 < N){
        c[p3] = a[p1]
        p1++, p3++
    }
    while( p2 < m) {
        c [p3] = b[p2]
        p2++, p3++
    }
    return c
}
```

TC : O(n+m)
SC : O(1)

Qn: Given a[N] & 3 idx s, m, e.

Given: subarray [s m] is sorted.
subarray [m+1 e] is sorted.
Sort the subarray [s e].

Eg:-



// Pseudo code:

```
int [] merge (int a[], int s, int m, int e) {
    p1 = s, p2 = m+1, p3 = 0
    temp [e-s+1]  // no. of elements in subarray [s e]
    while (p1<=m && p2<=e) {
        if (a[p1] <= a[p2]) {
            temp[p3] = a[p1]
            p1++, p3++
        }
        else {
            temp[p3] = a[p2]
            p2++, p3++
        }
    }
    // Copy rest of the elements.
    while(p1<=m){
        temp[p3] = a[p1]
        p1++, p3++
    }
    while(p2<=e) {
        temp[p3] = a[p2]
        p2++, p3++
    }
    // Copy temp array to a[]
    K=0
    for(i=s; i<=e; i++) {
        a[i] = temp[K]
        K++
    }
}
```

TC: $O(e-s+1) \Rightarrow O(n)$

SC: $O(n)$

a[i] = temp[i-s]

a[]	temp[]
i	i-s
s	0
s+1	1
s+2	2

HW: Can you do this in O(1) SC?

↳ If yes, how?

↳ If no, why?

Break: 8:40 am

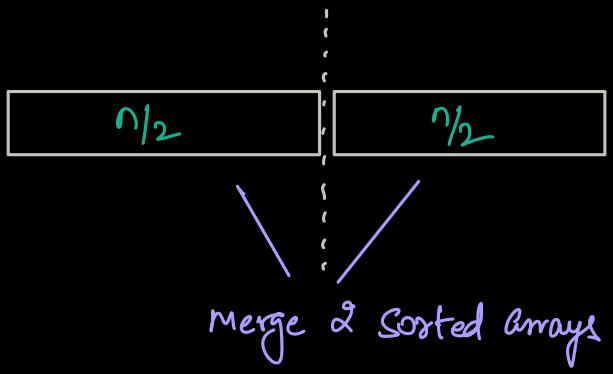
Bubble Sort
Selection Sort
Insertion Sort
} TC: $O(n^2)$

Qn: Given 100 elements. Sort them.

Case 1: Use n^2 algo (any of above 3)

$= n^2$ (10,000)

Case 2:

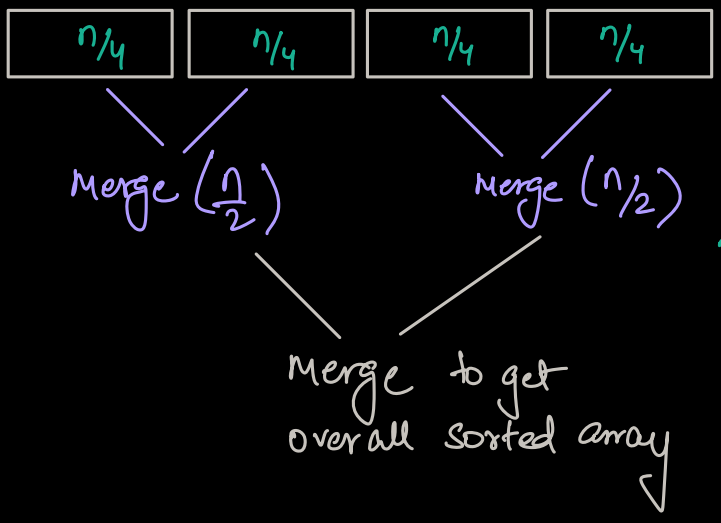


$= \left(\frac{n}{2}\right)^2 \times 2 + n$

$= 5100$

~ 50% improvement

Case 3:

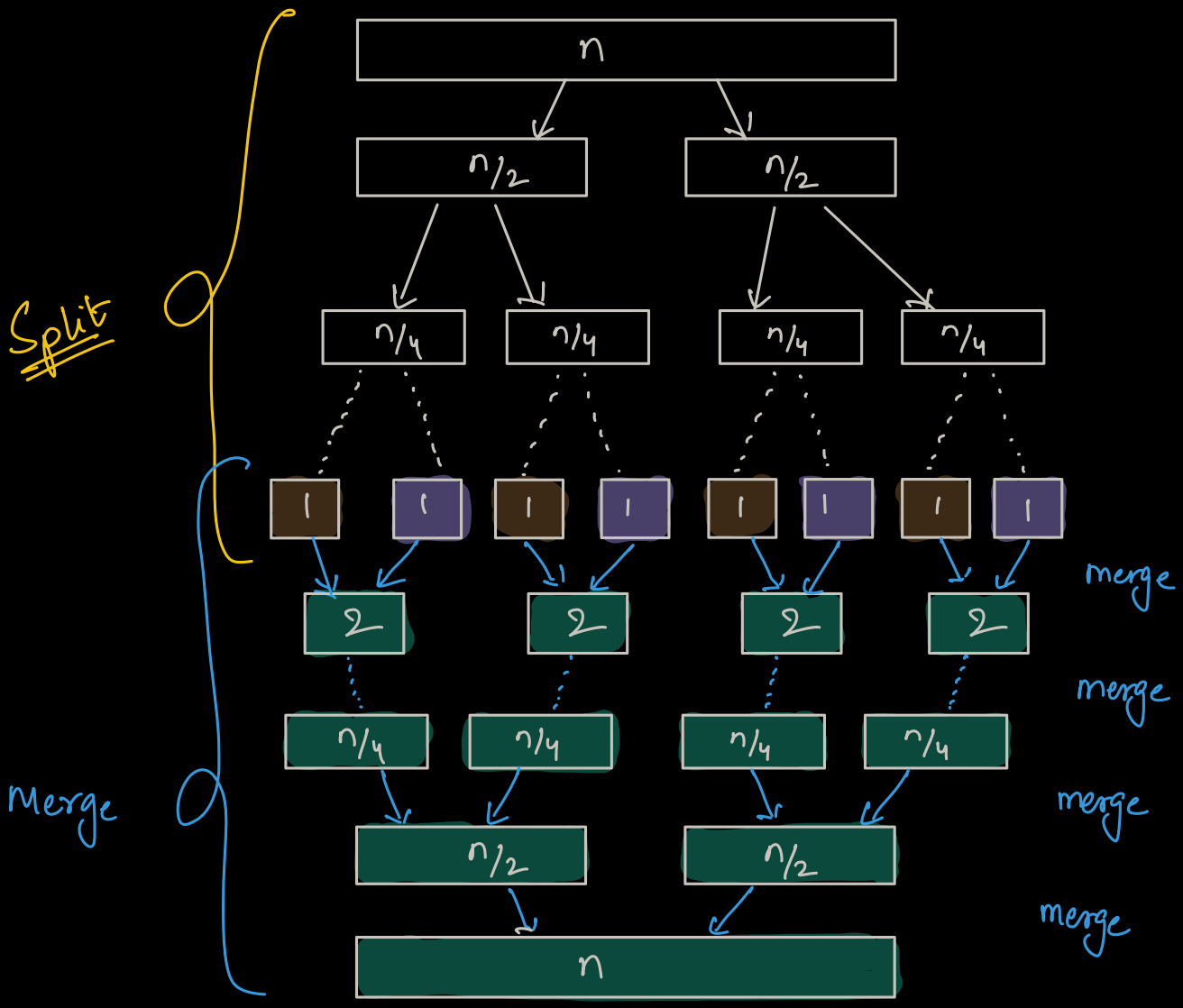


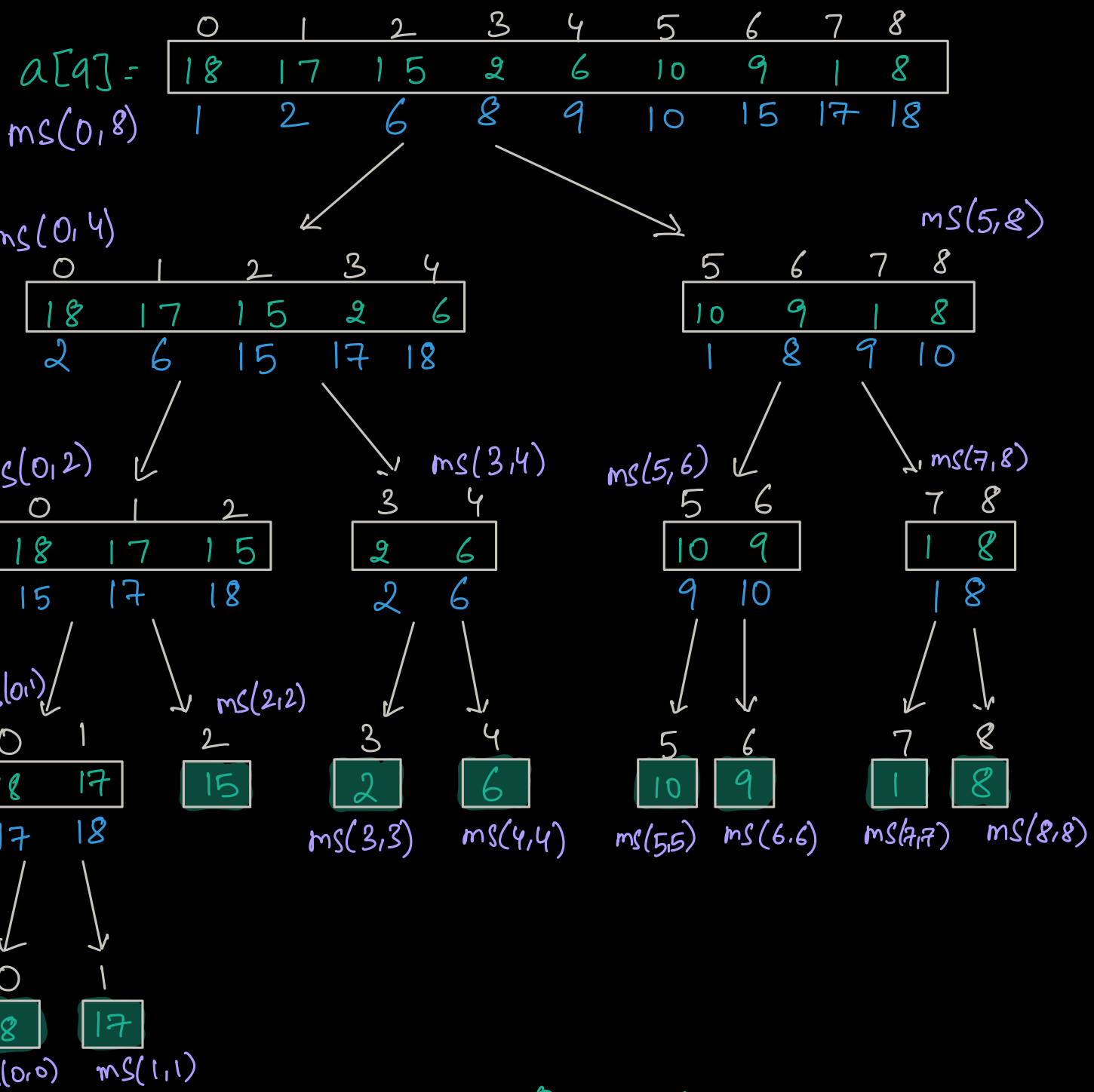
$= \left(\frac{n}{4}\right)^2 \times 4 + \left(\frac{n}{2}\right) \times 2 + n$

$= 2700$

~ 50% improvement over Case 2.

Case 4: Keep on dividing until 1 element is remaining (General Case)





// Pseudocode

```
void mergeSort(int a[], int s, int e) {
        if (s == e) { return }
        mid = (s+e)/2
        mergeSort(a, s, mid)
        mergeSort(a, mid+1, e)
        merge(a, s, mid, e) ⇒ Prev On
}

main() {
        mergeSort(a, 0, n-1) // Sort entire array.
}
```

Recurrence Relation:

$$T(n) = T(n/2) + T(n/2) + n \quad \text{(Merge)}$$

$$T(n) = 2T(n/2) + n \rightarrow 2^1 T(n/2^1) + 1 \times n$$

$$T(n/2) = 2T(n/4) + \frac{n}{2}$$

$$= 2\left[2T(n/4) + \frac{n}{2}\right] + n$$

$$= 4T(n/4) + 2n \rightarrow 2^2 T(n/2^2) + 2n$$

$$T(n/4) = 2T(n/8) + n/4$$

$$= 4\left[2T(n/8) + n/4\right] + 2n$$

$$= 8T(n/8) + 3n \rightarrow 2^3 T(n/2^3) + 3n$$

3 3 2 2

3 3 2 2

3 3 2 2

3 3 2 2

2 2 3 3

⋮

General form

$$T(n) = 2^k T(n/2^k) + kn. \qquad T(1) = 1$$

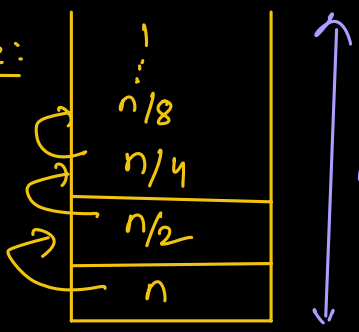
$$\frac{n}{2^k} = 1 \Rightarrow 2^k = n \qquad k = \log_2 n$$

$$= nT(1) + n\log n$$

$$= n + n\log n$$

$$T(n) = O(n\log n)$$

Space:



$\log n$ + n (Merge fn) = $O(n)$

Not in-place + Stable.