Todays Content:

- → Sorting Basics , Stable Sorting
- → Selection Sort
- → Bubble Sort
- → Merge Sort
    - a) Merge 2 Sorted arr[]
    - b) Merge 2 sorted arr[] in single arr[]

Sorting: arranging data in inc/dec order based on parameter

1 2 3 5 7 : inc, parameter = value

9 6 3 2 1 : dec, parameter = value

1 7 2 9 24 : inc based on no: of factors
1 2 2 3 8

Why sorting: Searching becomes easier

Stable/unstable: When 2 data points have same parameter value & their relative order is same after sorting ⇒ Stable sorting Algo

↳ parameter

: sort data increasing order based on marks

Name	Marks
✓ Mohit	0
Aman	100
Shrest	45
✓ kajal	32
Aayush	45
Aditya	92

Name	Marks
Mohit	0
kajal	32
Shrest	Aayush
Aayush	Shrest

Case-I sort A → Stable Sorting

Case-II sort B

Eg2: ar[5] = 1 5 2 6 2

Stable / Not

After Sort1 = 1 2 2 5 6 : Stable

After Sort2 = 1 2 2 5 6 : Unstable

Inplace Sorting: If Sorting algo take O(1) space in that case it is considered as inplace sorting algo

IQ) Given ar[N] elements, find k^{th} smallest elements : {repeat}

↳ Return ar[k-1] index element, when ar[] sorted?

	0	1	2	3	4	5	6	7	8	k=3	k=4	k=7	k=2	k=1
ar[8] =	2	8	4	-1	6	7	5	10	-1					
	2		3	0	5	6	4		1	2	4	7	-1	-1

Idea: Sort ar[] & return ar[k-1] TC: O(N log N)

Idea2: Iterate on ar[] find min keep it at its correct pos
repeat above proc k times TC: O(N*k) SC: O(1)

k=4:

	0	1	2	3	4	5	6	7	8	min	ind	swap(_ _)
ar[8] =	2	8	5	-1	6	7	4	10	-1			
i=0 →										-1	3	swap(ar[3], ar[0])
	-1	8	5	2	6	7	4	10	-1	-1	8	swap(ar[8], ar[1])
i=1 →												
	-1	-1	5	2	6	7	4	10	8	2	3	swap(ar[3], ar[2])
i=2 →												
	-1	-1	2	5	6	7	4	10	8	4	6	swap(ar[6], ar[3])
i=3 →												
	-1	-1	2	4	6	7	5	10	8	: return ar[3]		

```
__ SelectionSort(int ar[N], int k){ TC: O(N*N) SC: O(1)
                                    ↳ [inplace sorting algorithm]
    i=0; i<N; i=i+1){
      // Iterate from [i, N-1] & calculate min & ind
      min = ar[i], ind = i
      j=i; j<n; j++){
        if(ar[j] < min){
          min = ar[j], ind = j
        }
      }
      Swap(ar[ind], ar[i]) // swap do on your own
    }
}
```

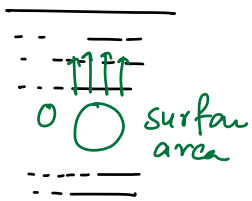
Ex:

	0	1	2	3	min	ind
	2	4	2	1	1	3
	1	4	2	2	2	2
	1	2	4	2	2	3
	1	2	2	4		

not stable algo → make it. TODO stable

2Q) ar[N], we can only swap adj elements sort arr[] inc

	0	1	2	3	4	5	6	7	8
ar[9] =	2	8	4	-1	6	7	5	10	-1
	2	4	-1	6	7	5	8	-1	10
	2	-1	4	6	5	7	-1	8	10
	-1	2	4	5	6	-1	7	8	10



Repeat above procn N times to make entire data sorted

```
bubbleSort(int ar[N]) { TC: O(N^2)  SC: O(1)
                                    └ inplace sorting algo
    i=0; i<N; i++) {
        int c=0                          → TODO: modifications possible
        j=0; j<N-1; j++) {
                                         → incorrect order
            if( ar[j] > ar[j+1] ) { j=N-1 ⇒ ar[N-1] & ar[N]
                                            └ err
                swap(ar[j] & ar[j+1])
                c=c+1
            }
        }
        if(c==0) {break}
    }
}
```

10:25 → 10:35 break

Ex: Stability

2	4	2	1
2	2	1	4
2	1	2	4
1	2	2	4

Stable sorting algo

Note:

ar[5]:	0	1	2	3	4
	3	1	6	10	8
	1	3	6	8	10 : 2 swaps
	1	3	6	8	10 : 0 swaps

In a iteration swap = 0, data is already sorted so, break

3Q) Given 2 sorted arrays, a[N], b[M], create c[N+M] which contains overall sorted data

Eg: a[4] : { 7 10 11 14 }
b[3] : { 3 8 9 }
c[7] : { 3 7 8 9 10 11 14 }

a[3] = { 3 6 10 }
b[2] = { 5 9 }
c[5] = { 3 5 6 9 10 }

Ideas 1 : a[N] ⟶ c[] : O(N)
b[M] ⟶ c[] : O(M)
sort c[] inbuild sort : [N+M] $\log_2$(N+M)

TC: O(N+M) $\log_2$(N+M)

Idea 2:

P_1 ⟶ P_1 // Copy remaining ele

Eg: a[4] : { 7 10 11 14 }
0 1 2 3

P_2 ⟶ P_2 ⟶ P_2 ⟶ P_2 out of array

b[3] : { 3 8 9 }
0 1 2

c[7] : { 3 7 8 9 10 11 14 }
0 1 2 3 4 5 6

Eg:

P_1 ⟶ P_1 ⟶ P_1 ⟶ P_1 ⟶ P_1

a[4] : { 2 8 10 14 } { out of array }
0 1 2 3

P_2 ⟶ P_2 ⟶ P_2 // copy remaing data

b[5] : { 3 7 16 20 24 }
0 1 2 3 4

c[9] : { 2 3 7 8 10 14 16 20 24 }
0 1 2 3 4 5 6 7 8

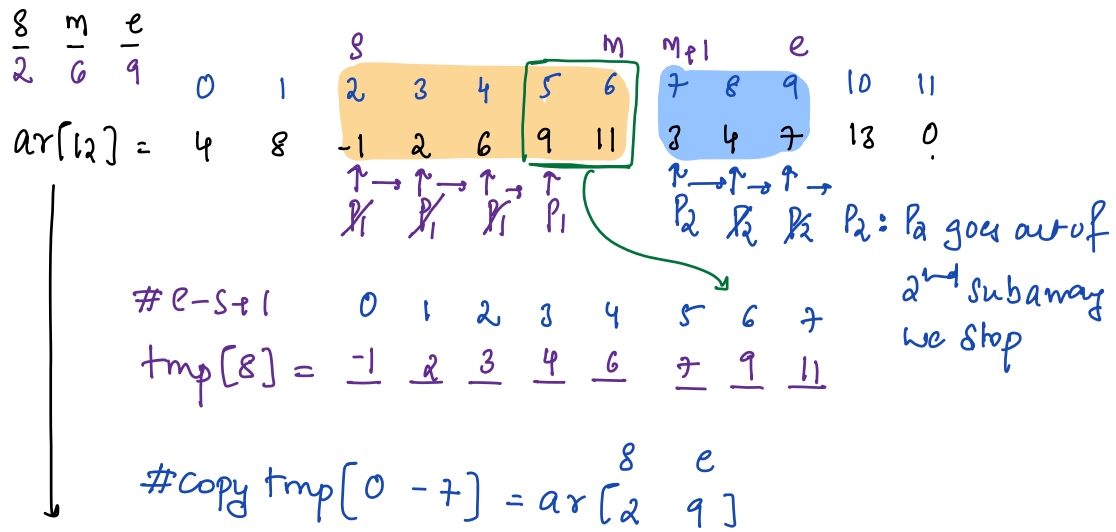
```
int[] merge2( int a[N], int b[M] )   TC: O(N+M)   SC: O(1)

    P1 = 0, P2 = 0, P3 = 0

    int c[N+M]

    while( P1 < N && P2 < M ){ // we break, if one of array completed  // Both P1 & P2 should be in order
        if( a[P1] <= b[P2] ){ // a[P1] should come first
            c[P3] = a[P1], P1 = P1+1  P3 = P3+1
        }
        else{
            c[P3] = b[P2], P2 = P2+1  P3 = P3+1
        }
    }
    // copy remaining elements
    while( P1 < N ){ // elements in a[] are leftout
        c[P3] = a[P1], P1 = P1+1  P3 = P3+1
    }
    while( P2 < M ){ // elements in b[] are leftout
        c[P3] = b[P2], P2 = P2+1  P3 = P3+1
    }
    return c[]
}
```

2Q) Given ar[N] elements & 3 indices, s, m, e

and subarray [s m] is sorted

subarray [m+1 e] is sorted

Sort subarray from [s e] ?



ar[12] = 4 8 -1 2 3 4 6 7 9 11 13 0

Pseudo Code:

given ar[s m] sorted & ar[m+1, e] is sorted, sort {s → e}

```
void merge(int A[], int s, int m, int e){   TC: O(N)  SC: O(N)

    p1 = s, p2 = m+1, p3 = 0
    int c[e-s+1]
                          // Both p1 & p2 should be in index
    while( p1 <= m && p2 <= e){ // we break, if one of array completed
        if( a[p1] <= a[p2] ){ // a[p1] should come first
            c[p3] = a[p1], p1 = p1+1  p3 = p3+1
        }
        else{
            c[p3] = a[p2], p2 = p2+1  p3 = p3+1
        }
    }
    // copy remaining elements
    while( p1 <= m ){ // elements in a[] are leftout
        c[p3] = a[p1], p1 = p1+1  p3 = p3+1
    }
    while( p2 <= e ){ // elements in b[] are leftout
        c[p3] = a[p2], p2 = p2+1  p3 = p3+1
    }

    // copy c[0...] → ar[s...e]
    i = s, j = 0
    while( i <= e ){
        ar[i] = c[j]  i = i+1, j = j+1
    }
}
```

s m m+1 e

p1 p2

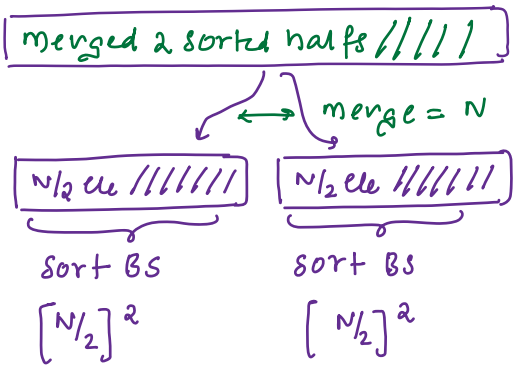
j i

SQ) Given ar[N] sort them, no inbuilt sort

↳ N = 100

TC: operations

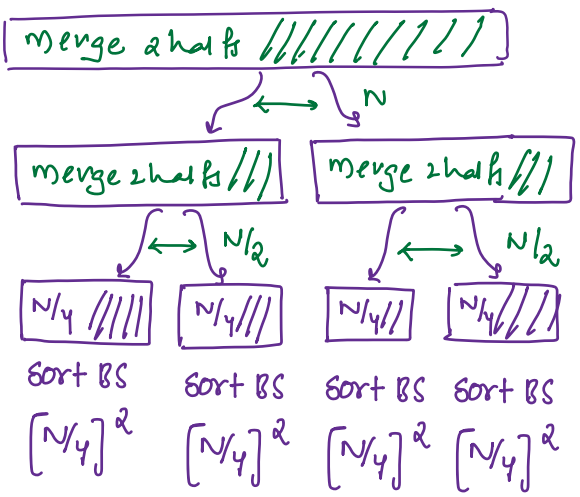
Case-I :  : $O(N^2)$ ⇒ 10^4

Case-II : 

: $\left[\frac{N}{2}\right]^2 * 2 + N$

: $\frac{N^2}{2} + N \Rightarrow$

: $\frac{10^4}{2} + 100 \Rightarrow 5100$

Case-III: 

: $\left[\frac{N^2}{4}\right] 4 + 2N$

: $\frac{N^2}{4} + 2N$

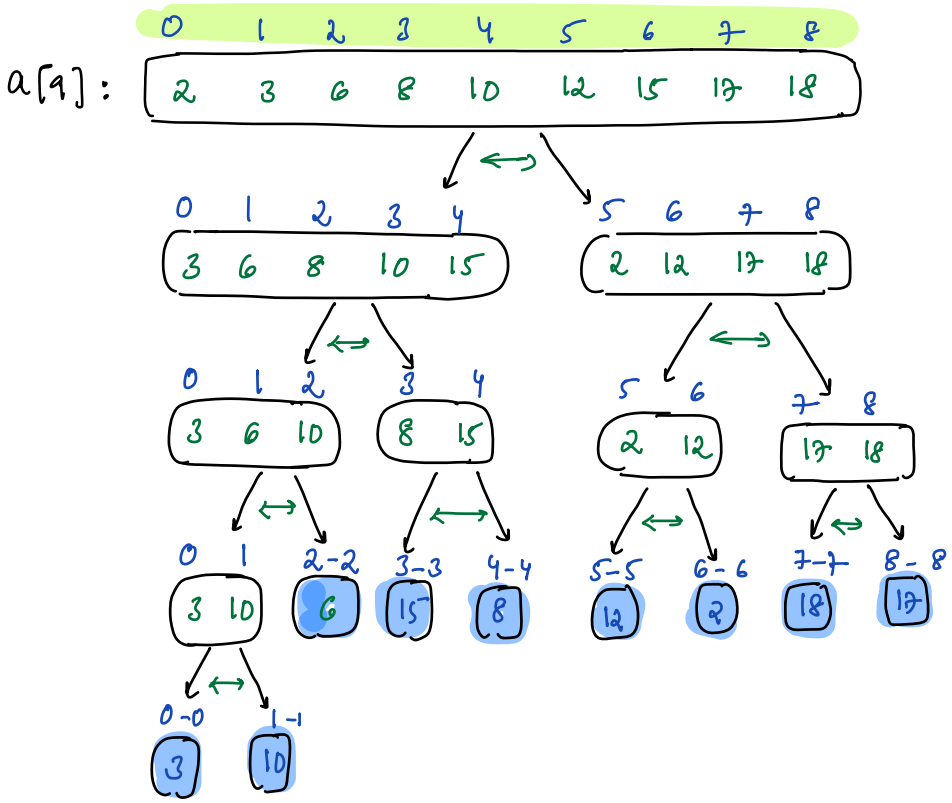
: $N = 100$

: $\frac{10^4}{4} + 2*100$

: 2700

Obs: keep dividing ar[] until it has only single element & start merging → {Idea of Merge Sort}

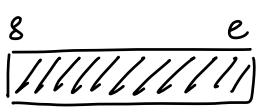
Idea: MergeSort



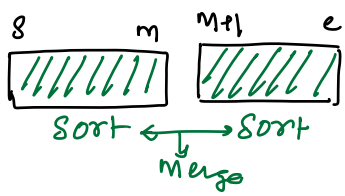
Ass: Given ar[], sort subarray from [s-e]

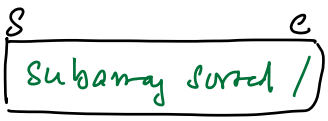
// pass by reference, same array from all function calls

```
void mergeSort(int ar[], int s, int e) {
    if(s==e) { // single element
        return;
    }
    m = (s+e)/2
    mergeSort(ar, s, m) : f(N/2)
    mergeSort(ar, m+1, e) : f(N/2)
    merge(ar, s, m, e) : N
}
```



m = (s+e)/2





$f(N) = 2f(N/2) + N \quad f(1) = 1$

TC: O(N log N)

SC: O(N)

$f(N) = f(N/2) + f(N/2) + N$

$f(N) = 2f(N/2) + N \quad f(1) = 1$

$f(N/2) = 2f(N/4) + N/2$

$= 2[2f(N/4) + N/2] + N$

$= 4f(N/4) + 2N = 2^2 f(N/2^2) + 2N$

$f(N/4) = 2f(N/8) + N/4$

$= 4[2f(N/8) + N/4] + 2N$

$= 8f(N/8) + 3N = 2^3 f(N/2^3) + 3N$

$f(N/8) = 2f(N/16) + N/8$

$= 8[2f(N/16) + N/8] + 3N$

$= 16f(N/16) + 4N = 2^4 f(N/2^4) + 4N$

After k Substitutions =

$f(N) = 2^k f(N/2^k) + kN \rightarrow f(1) = 1$

$N/2^k = 1 \Rightarrow 2^k = N \Rightarrow k = \log_2^N$

$Nf(1) + \log_2^N N \Rightarrow O(N \log N)$

$f(N) = O(N \log N)$

Beginner → May 22 Java Monday - 1

Intermediate → Jun22 Intermediate Morning - 1

Raise it in support & get recordings