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[]
- Diwali Announcements: Oct 21 → Oct 26
  cross verify

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

1 2 3 5 7 : inc

9 6 3 2 1 : dec

1 7 2 9 24 : inc based factors

1 2 2 3 8

Why sorting: Searching becomes easier

Stable/unstable: When 2 data points have same parameter & their order is retained after sorting, then it's stable sorting

: sort data increasing order based on marks

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

same parameter

Name	Marks
Mohith	0
kajal	32

Case-I
Shrest : 45
Aayush : 45

Case-II
Aayush : 45
Shresh : 45

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 : Not Stable

Inplace Sorting: Sorting performed without extra space SC: O(1)

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

k=3, k=4 ?

2 4

Idea: Sort ar[] & return ar[k-1], TC: O(NlogN) SC: O(1)

Idea2: Apply selection step only 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	4	-1	6	7	5	10	-1	-1	3	ar[0] & ar[ind]
	-1	8	4	2	6	7	5	10	-1	-1	8	ar[1] & ar[ind]
	-1	-1	4	2	6	7	5	10	8	2	3	ar[2] & ar[3]
	-1	-1	2	4	6	7	5	10	8	4	3	ar[3] & ar[3]
	-1	-1	2	4	6	7	5	10	8			return ar[3]

If we repeat process ar[] will be sorted : N times

```
____ selectionSort(int ar[N]){ TC: O(N^2) SC: O(1)
    for(i=0; i<n; i++){
        // iterate from i, n-1 & get min & ind
        minv = ar[i], ind = i
        for(j=i; j<n; j++){
            if(minv > ar[j]){ minv = ar[j], ind = j}
        }
        swap(ar[ind], ar[i])
    }
}
```

↳ Inplace: ✓

: Stable: ✗

: TODO make it stable

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

	0	1	2	3	4	5	6	7	8
ar[8] =	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 n times ar[] will be sorted

→ inplace ✓ stability ✓

__ bubbleSort(int ar[N]) TC: O(N²) SC: O(1)

```
for(i=0; i<n; i++){
    c=0
    for(j=0; j<n-1; j++){ // j=n-1, j+1=n out of bounds
        if(ar[j] > ar[j+1]){
            swap(ar[j], ar[j+1]), c=c+1
        }
    }
    if(c==0) {break}
}
```

TODO: We can change condition ?

Ex: 2 4 2 1

2 2 1 4

2 1 2 4

1 2 2 4

If ar[N]: _ _ _ k k _ _

↓ We won't swap if same data

Note:

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

: 0 swaps

If 0 swaps, ar[] is sorted, break it

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

Ex: a[4] : { 7 10 11 14 } a[3] = { 3 6 10 }

b[3] : { 3 8 9 } b[2] = { 5 9 }

c[7] : { 3 7 8 9 10 11 14 } c[5] = { 3 5 6 9 10 }

Ideas 1 : a[N], b[M], c[N+M]

Copy a[] → c[] : O(N)
Copy b[] → c[] : O(M)
Sort c[] : (N+M) log(N+M)

TC: (N+M) log(N+M)
SC: O(1)

Idea 2:

Ex: a[4] : { 7 10 11 14 } : copy remaining elements

P1 → P1
0 1 2 3

b[3] : { 3 8 9 } : out of bounds

P2 → P2 → P2 → P2
0 1 2

c[7] : { 3 7 8 9 10 11 14 }

P3 → P3 → P3 → P3 → P3 → P3 → P3
0 1 2 3 4 5 6

Ex: a[4] : { 2 8 10 14 } [out of bounds]

P1 → P1 → P1 → P1 → P1
0 1 2 3

b[5] : { 3 7 16 20 24 } : copy remaining ele =

P2 → P2 → P2
0 1 2 3 4

c[9] :

0	1	2	3	4	5	6	7	8
2	3	7	8	10	14	16	20	24

```
int[] merge2(int a[N], b[M]) { TC: O(N+M) SC: O(1)
    int c[N+M]
    P1 = 0, P2 = 0, P3 = 0        { even if P1, P2 one of them goes to out of bounds we stop }
    while (P1 < N || P2 < M) {
        if (a[P1] <= b[P2]) {
            c[P3] = a[P1]; P3++, P1++
        }
        else {
            c[P3] = b[P2]; P3++, P2++
        }
    }
    while (P1 < N) { c[P3] = a[P1]; P3++, P1++ }
    while (P2 < M) { c[P3] = b[P2]; P3++, P2++ }
    return c[]
}
```

8:30 —lomins→ 8:40

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] ?

s m e
2 6 9

	0	1	2	3	4	5	6	7	8	9	10	11
ar[12] =	4	8	-1	2	6	9	11	3	4	7	18	0

e-s+1 = 9

tmp[8]:	0	1	2	3	4	5	6	7
	-1	2	3	4	6	7	9	11

Copy tmp[0 e-s] → original arr[s e]

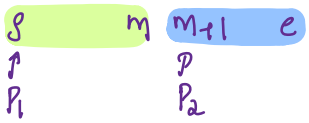
	0	1	2	3	4	5	6	7	8	9	10	11
ar[12] =	4	8	-1	2	3	4	6	7	9	11	18	0

Pseudo Code:

given ar[s m] sorted & ar[m+1, e] is sorted, s ≤ e

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

```
c[e-s+1]
```



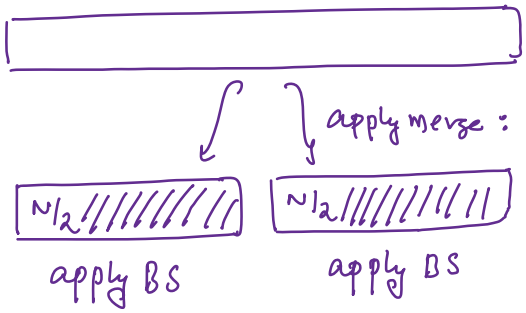

```
p1 = s  p2 = m+1, p3 = 0
while( p1<=m && p2<=e){
    if( A[p1] <= A[p2]) { c[p3] = A[p1], p3++, p1++}
    else { c[p3] = A[p2], p3++, p2++}
}
while(p1<=m) { c[p3] = A[p1], p3++, p1++}
while (p2<=e) { c[p3] = A[p2], p3++, p2++}
//Copy c[] → ar[]
i=s, j=0 ; i<=e; i++, j++){
    ar[i] = c[j]
}
}
```

SQ) Given ar[N] Sort them

↳ N = 100

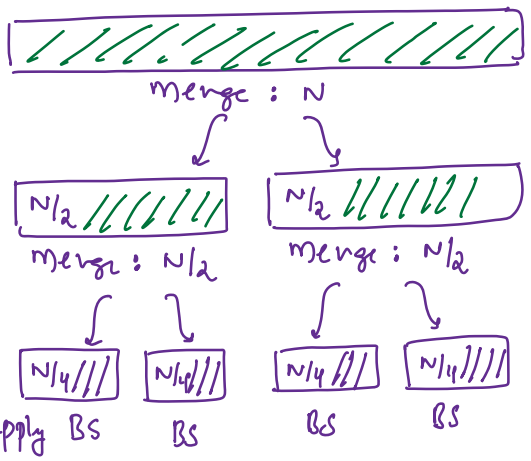
Case-I :  : N ⇒ $O(N^2) \approx 10^4$

apply Bubble Sort on ar[]

Case-II :  : TC: $2\left[\frac{N}{2}\right]^2 + N$

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

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

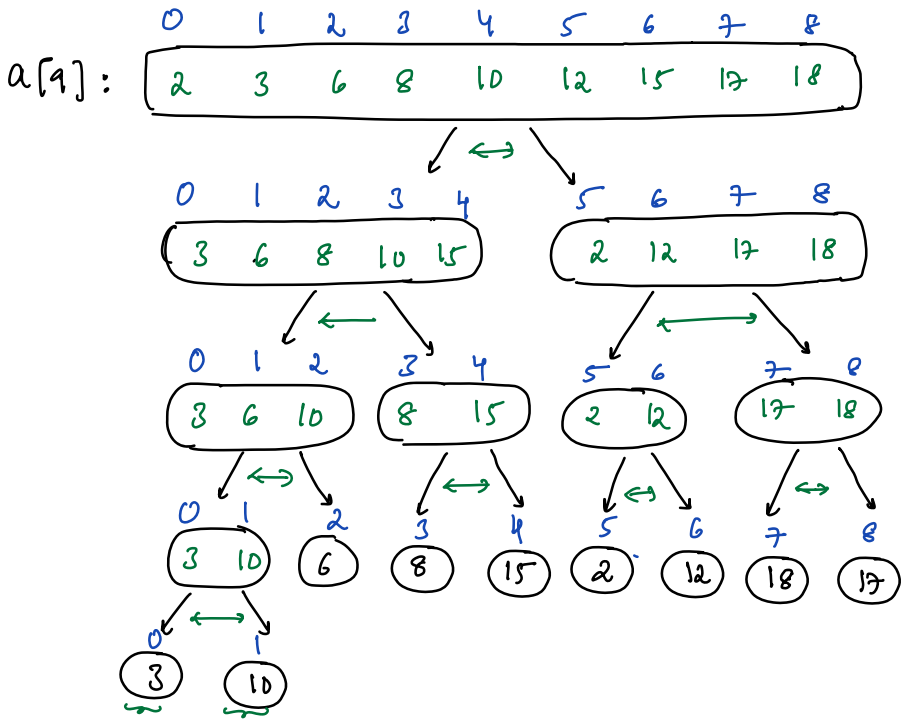
Case-III :  : TC: $\left[\frac{N}{4}\right]^2 * 4 + 2N$

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

$\Rightarrow \frac{10^4}{4} + 2 * 10^2 = 2700$

Idea: Keep breaking untill there is only single e now start merging

Idea:

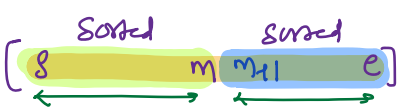


Ass: Given ar[], s, e → sort it from [s → e]

pass by reference:

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

ar[]



Sort ar[s - e] by merging

↳ recursive stack space

↳ Space of merge part

TC: O(NlogN) SC: O(logN + N) ≈ O(N)

TC: O(NlogN) SC: O(N) inplace* Stable: TODO?

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

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

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

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

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

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

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

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

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

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

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

After k Substitutions =

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

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

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

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