Todays Content: ——————→ 9:05 pm Start

a) Sorting Intro

b) Sorting Algorithms

   a) Bubble Sort

   b) Insertion Sort

   c) Merge Sort → Recursion  9:07pm

New Students:

   a) Pre-Requists Recursion

   b) Even if you are not aware you will get 80%

**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, parameter = factr

fact: 1 2 2 3 8 :

**Why Sorting:** It helps in searching data.

Property: 1

Sort data increasing order based on marks (parameter)

| Name | Marks |
|---|---|
| Mohit | 0 |
| Aman | 100 |
| Shrush | 45 |
| Kajal | 32 |
| Aayush | 45 |
| Aditya | 92 |

S Algo1: Aayush 45
Shrush 45

| Name | Marks |
|---|---|
| Mohit | 0 |
| Kajal | 32 |
| Shrush | 45 |
| Aayush | 45 |

S Algo2: Shrush 45, Aayush 45 → Stable Sorting

**Stable Sorting:** When 2 data points have same parameter value, their relative order should be same, before & after sorting.

Ex: ar[5] = { 1 5 2 6 2 } sort inc based on value.

Sort1 = { 1 2 2 5 6 } Sort1 is not stable

Sort2 = { 1 2 2 5 6 } Sort2 is Stable

**Inplace Sorting:**

Sorting algorithm with SC: O(1), it's known Inplace.

Sorting : → Stable
→ Inplace

## Bubble Sort:

Given ar[N], Sort ar[] in inc order, by swapping only adjacent elements.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| ar[9] = { | 8 | 2 | 4 | -1 | 6 | 7 | 5 | 10 | -1 } | obs: |
| itr1 { | 2 | 4 | -1 | 6 | 7 | 5 | 8 | -1 | 10 } | last 2 ele in correct pos. |
| itr2 { | 2 | -1 | 4 | 6 | 5 | 7 | -1 | 8 | 10 } | last 2 ele in correct pos. |
| itr3 { | -1 | 2 | 4 | 5 | 6 | -1 | 7 | 8 | 10 } | last 3 ele in correct pos. |

obs: Repeat above iterations N times, entire ar[] sorted in inc order

```
void BubbleSort(int ar[]){ TC: O(N^2) SC: O(1)
    int N = ar.length;
    for(int i=0; i<N; i++){
        int swap=0
        for(int j=0; j<N-1; j++){ // j=N-1, ar[j]>ar[j+1] => ar[N-1]>ar[N] : Error
            if(ar[j]>ar[j+1]){
                swap ar[j] & ar[j+1]
                swap++;
            }
        }
        if(swap==0){
            // data sorted;
            break;
        }
    }
}
```

Dry Run =

| | 0 | 1 | 2 | 3 | 4 | |
|---|---|---|---|---|---|---|
| ar[5] = { | 3 | 1 | 6 | 10 | 8 } | |
| i=0 itr1 = | 1 | 3 | 6 | 8 | 10 | : 2 swaps |
| i=1 itr2 = | 1 | 3 | 6 | 8 | 10 | : 0 swaps |

obs: In a full iteration, if there are no swaps, data is sorted.

Check Stability: Parameter same: Relative order maintained

| | 0 | 1 | 2 | 3 | |
|---|---|---|---|---|---|
| ar[4] = { | 2 | 4 | 2 | 1 } | BubbleSort |
| itr1 = | 2 | 2 | 1 | 4 | a) Stable Sort |
| itr2 = | 2 | 1 | 2 | 4 | b) Inplace Sort |
| itr3 = { | 1 | 2 | 2 | 4 } | |

## Insertion Step:

Given an ar[N], first N-1 elements are sorted, sort entire ar[].

Note: No Extra Space is allowed.

Examples:

| | 0 | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|---|
| ar[6] = { | 2 | 6 | 10 | 14 | 20 | 4 | } |

After Sorting

= { 2 4 6 10 14 20 }

N=8

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| ar[8] = { | 2 | 7 | 12 | 19 | 24 | 30 | 34 | 4 | } |

ar[] = { 2 4 7 12 19 24 30 34 }

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| ar[7] = { | 3 | 6 | 10 | 14 | 20 | 24 | 7 | } |

= { 3 6 7 10 14 20 24 }

Tracing : [0 5] sorted :

| j | ar[j] > ar[j+1] | T/F | Swap |
|---|---|---|---|
| 5 | ar[5] > ar[6] | T | swap |
| 4 | ar[4] > ar[5] | T | swap |
| 3 | ar[3] > ar[4] | T | swap |
| 2 | ar[2] > ar[3] | T | swap |
| 1 | ar[1] > ar[2] | F | break |

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

```
void SortStep(int ar[]){
    int N = ar.length;
    /* First N-1 elements sorted.
       Means from [0... N-2]
       initialize j at last sorted index */
    for(int j=N-2; j>=0; j--){
        if(ar[j] > ar[j+1]){
            swap ar[j] & ar[j+1]
        else{ break;}
    }
}
```

## Insertion Step:

Insert 1 single ele in sorted data to make entire data sorted its called

38) Given ar[N] sort it using Insertion Step.

ar[] =

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 10 | 3 | 6 | 8 | 2 | 5 |

Step1: [0-0] sorted, Insert ar[1], Sort [0-1] : Insertion Step

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 3 | 10 | 6 | 8 | 2 | 5 |

Step2: [0-1] sorted, Insert ar[2], Sort [0 2] Insertion Step

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 3 | 6 | 10 | 8 | 2 | 5 |

Step3: [0-2] sorted, Insert ar[3], Sort [0 3] Insertion Step

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 3 | 6 | 8 | 10 | 2 | 5 |

Step4: [0-3] sorted, Insert ar[4], Sort [0 4] Insertion Step

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 2 | 3 | 6 | 8 | 10 | 5 |

Step5: [0-4] sorted, Insert ar[5], Sort [0 5] Insertion Step

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 2 | 3 | 5 | 6 | 8 | 10 |

void InsertionSort (int ar[]){ TC: O(N²) SC: O(1) → Inplace Sorting

Stable → TODO / Yes Try Example.

```
int N = ar.length;
for(int i=1; i<N; i++){
    // Insert ar[i] in sorted data {0..i-1} sorted, Insertion Step.
    for(int j=i-1; j>=0; j--){
        if(ar[j] > ar[j+1]){
            swap ar[j] & ar[j+1]
        }else{ break;}
    }
}
```

{ 0 1 2 3 .. i-2 i-1 i }

48) Given 2 sorted arrays A[N], B[M], create C[N+M] which contains overall sorted data.

A[4] = 7 10 11 14

B[3] = 3 8 9

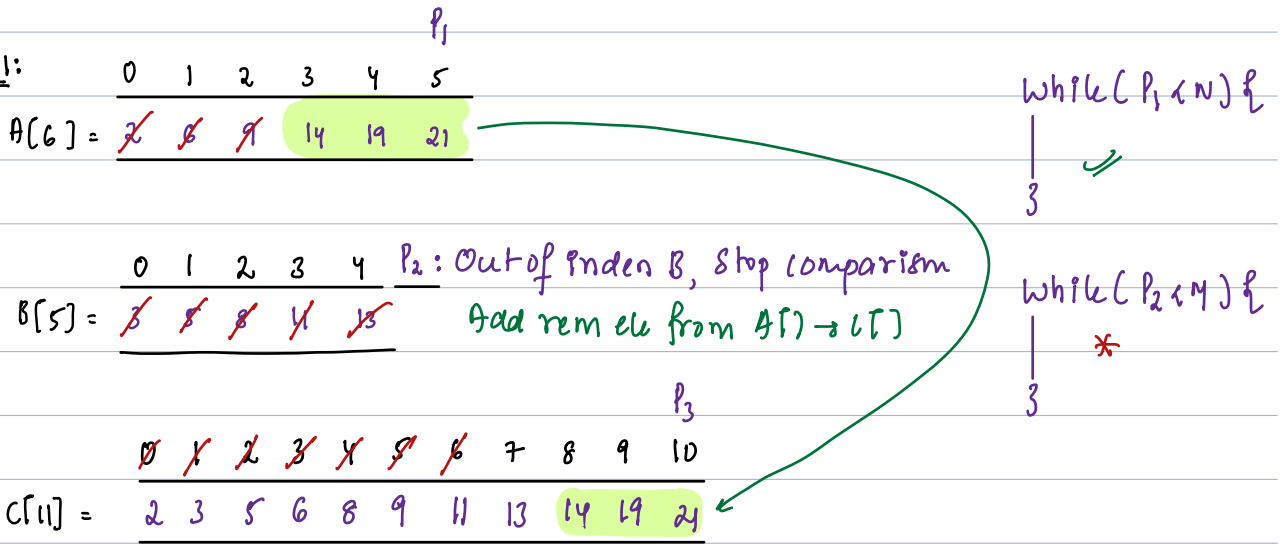
C[7] = 3 7 8 9 10 11 14

A[3] = 3 6 10

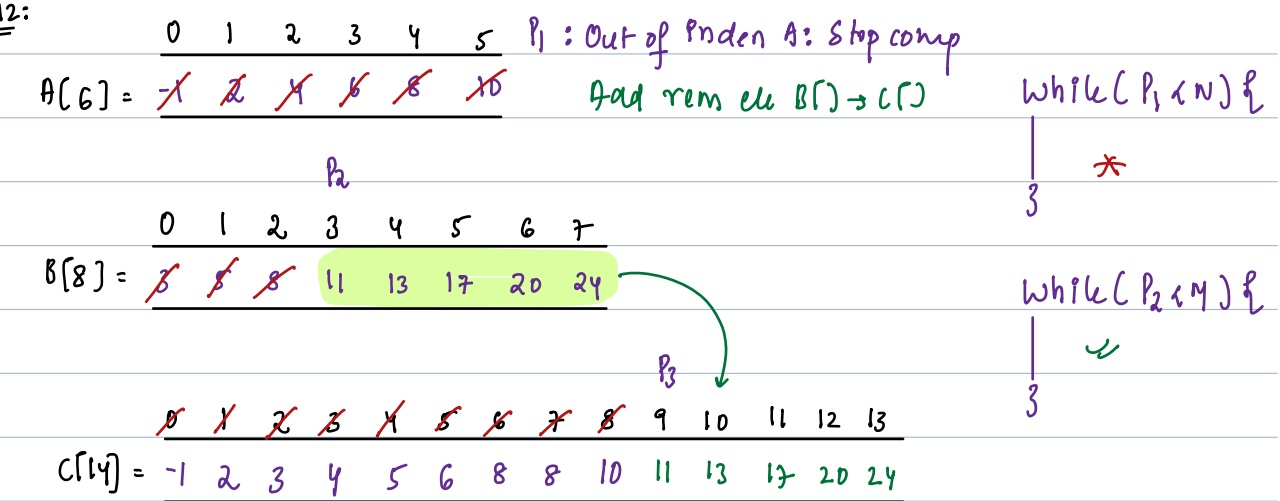
B[3] = 5 8 14

C[6] = 3 5 6 8 10 14

Ex1:




Ex2:



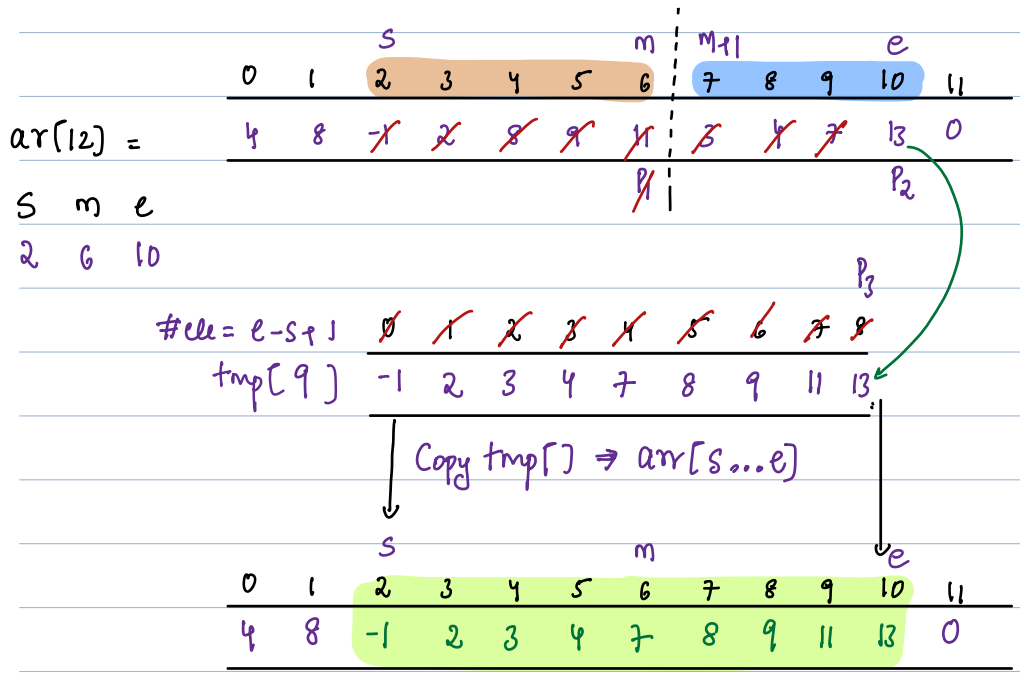
```
int[] merge(int A[], int B[]){  TC: O(N+M)  SC: O(1)
    int N = A.length
    int M = B.length
    int C[N+M]; ---> // asked by question.
    int P1 = 0  P2 = 0  P3 = 0.

    while(P1 < N && P2 < M){ // Compare till both are valid
        if(A[P1] < B[P2]){ C[P3] = A[P1]; P1++; P3++ }
    } else { C[P3] = B[P2]; P2++; P3++ }

    while(P1 < N){ // Elements in A[] are remaining
        C[P3] = A[P1]; P1++; P3++
    }

    while(P2 < M){ // Elements in B[] are remaining
        C[P3] = B[P2]; P2++; P3++
    }

    return C;
}
```

Q8) Given ar[N] elements & 3 indices s, m, e

→ Subarray [s m] is sorted

→ Subarray [m+1 e] is sorted

Sort Entire Subarray from [s...e]

## Pseudocode:

A[]: Subarray [s m] is sorted
: Subarray [m+1 e] is sorted
: Sort entire subarray from [s...e]

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

```
int temp[e-s+1]; // [ s  s+1 .. m | m+1 ... e ]
                    p1              p2
int p1 = s, p2 = m+1, p3 = 0;

while( p1 <= m && p2 <= e ) { // Compare till both are valid
    if( A[p1] < A[p2] ) { tmp[p3] = A[p1]; p1++; p3++ }
    else { tmp[p3] = A[p2]; p2++; p3++ }
}

while( p1 <= m) { // Copy rem ele to tmp[]
    tmp[p3] = A[p1]; p1++; p3++
}

while( p2 <= e) { // Copy rem ele to tmp[]
    tmp[p3] = A[p2]; p2++; p3++
}

// tmp[] sorted  Copy  A[s-e]
int k = 0;
for(int i = s; i <= e; i++){
    A[i] = tmp[k];
    k++;
}
}
```

tmp[e-s+1]: 0 1 2 ..... e-s TODO

A[s..e]: s s+1 s+2 .... e

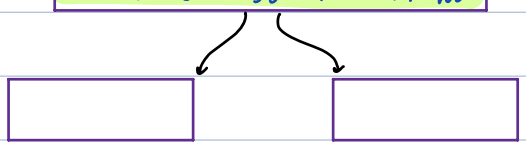
## Idea of Merge Sort : { Tomorrow 9PM : 9:40PM }

Given ar[100] elements sort them.

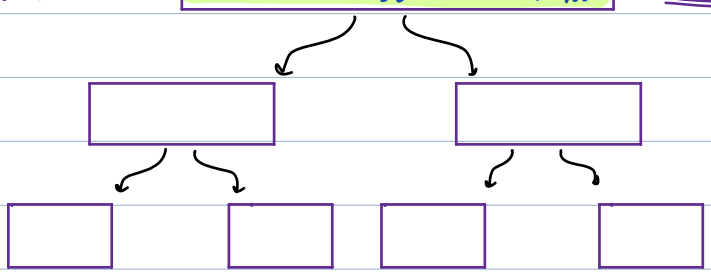
Case1 : $a_1\ a_2\ a_3\ ..\ a_{50}\ a_{51}\ ...\ a_{99}\ a_{100}$ Iterations : $10^4$

1. Apply BS/IS on above ele : $O(N^2) = 100^2 = 10^4$

Case2 : $a_1\ a_2\ a_3\ ..\ a_{50}\ a_{51}\ ...\ a_{99}\ a_{100}$ Iterations

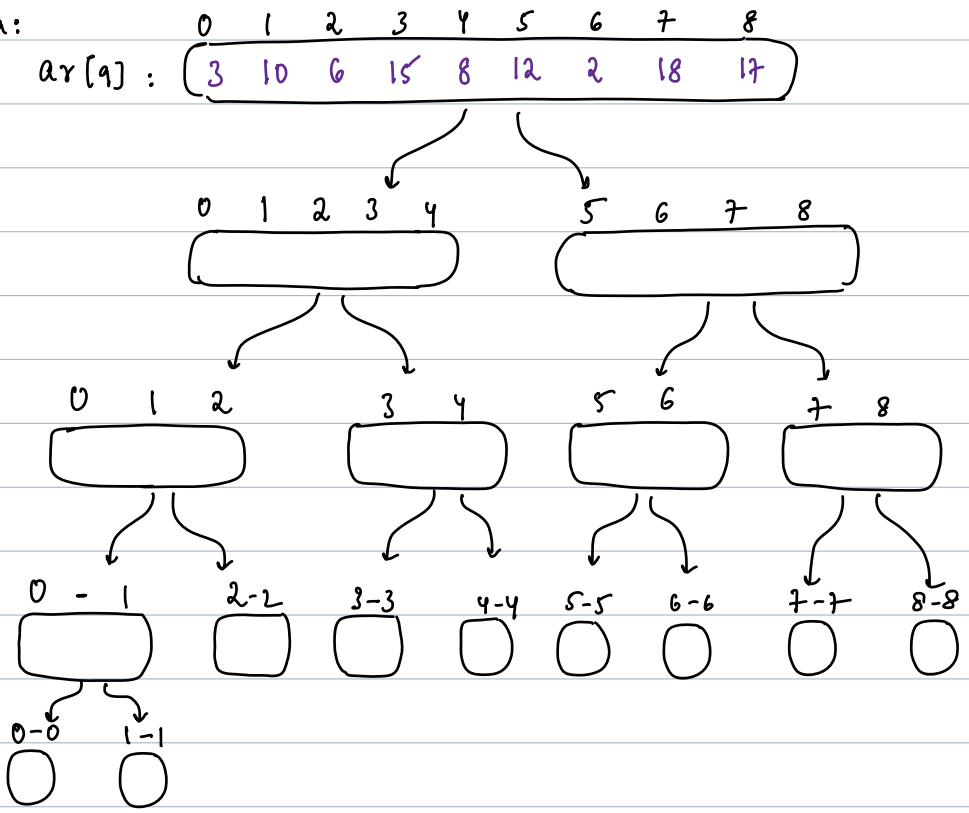


Case3 : $a_1\ a_2\ a_3\ ..\ a_{50}\ a_{51}\ ...\ a_{99}\ a_{100}$ Iterations



Idea: Keep dividing ar[], till it contains 1 element & merge them.

Idea:




```
main ( ) {
    // Given ar[N] sort entirely
     mergeSort(ar, 0, n-1)
}
```

1. Ass: Given subarray from [s..e] sort it.

```
void mergeSort ( int ar[], int s, int e){

}
```

A[ ]: Subarray [s m] is sorted
: Subarray [m+1 e] is sorted
: Sort entire subarray from [s...e]

```
void merge(int A[], int s, int m, int e){

}
```