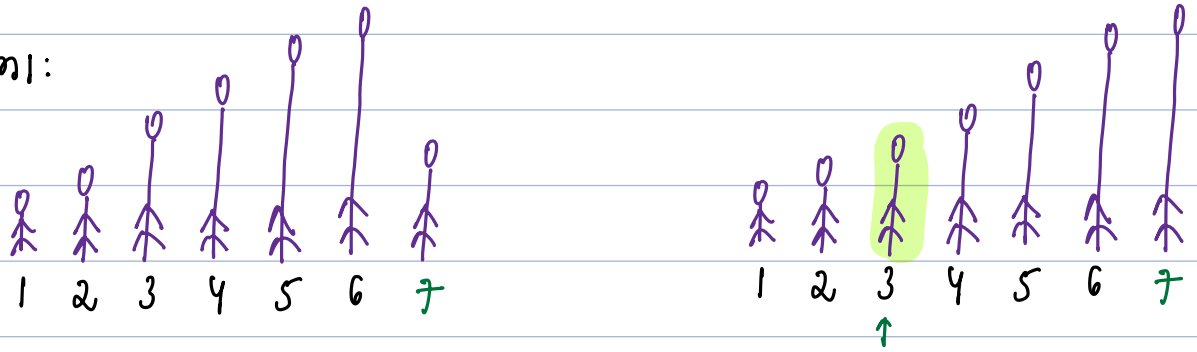
Todays Content

1. Insertion sort
2. Merge 2 sorted arrays
3. Merge 2 sorted subarrays.

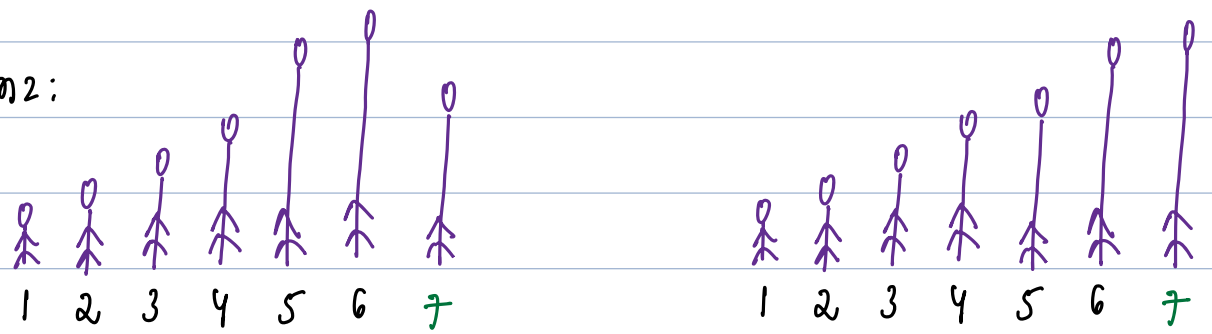
Insertion Sort:

We insert 1 element in existing sorted data to make entire data sorted

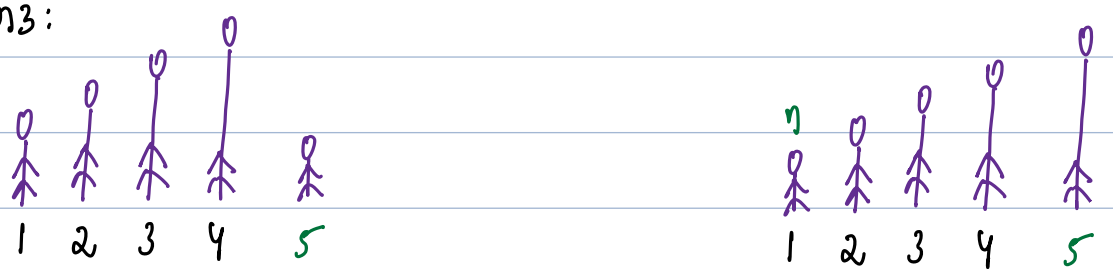
Eg1:



Eg2:



Eg3:



while (n_per not at start && p_per > n_per){

    swap p_per & n_per

}

Insertion Sort:

	0	1	2	3	4	5
ar[ ] = {	10	9	4	8	6	2 }

Dry Run:

	Sort	Insert	ar[ ] = {	0	1	2	3	4	5 }
i = 0	[0 0]	ar[1]		9	10	4	8	6	2
				j ← j					
i = 1	[0 1]	ar[2]		4	9	10	8	6	2
				j ← j ← j					
i = 2	[0 2]	ar[3]		4	8	9	10	6	2
				j ← j ← j					
i = 3	[0 3]	ar[4]		4	6	8	9	10	2
				j ← j ← j ← j					
i = 4	[0 4]	ar[5]		2	4	6	8	9	10
				j ← j ← j ← j ← j ← j					

Note: Stability maintained in insertion sort

```
void InsertionSort(int[] ar, int N){   TC: O(N^2)  SC: O(1) : Inplace
    for(int i=0; i<N; i++){
        # [0..i] sorted insert ar[i+1];
        int j = i+1;
        while(j>0 && ar[j-1] > ar[j]){
            swap ar[j-1] & ar[j];
            j--;   # update new element position
        }
    }
}
```

TODO: Take an almost sorted array & apply BubbleSort, SelectionSort, InsertionSort & compare iterations

Final Observations:

Decreasing order of speed.

InsertionSort > BubbleSort > SelectionSort

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

	0	1	2	3
A[4] :	7	10	11	14

	0	1	2
A[4] :	3	6	10

	0	1	2
B[3] :	3	8	9

	0	1	2	3
B[3] :	5	14	20	25

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

	0	1	2	3	4	5	6
C[7] :	3	5	6	10	14	20	25

#Ideal:

1. Create C[N+M]
2. Copy A[] → C & B[] → C
3. Sort C[] - /BS/IS/SS

TC: $O(N + M + (N+M)^2) = O(N+M)^2$

↳ Inbuilt Sort

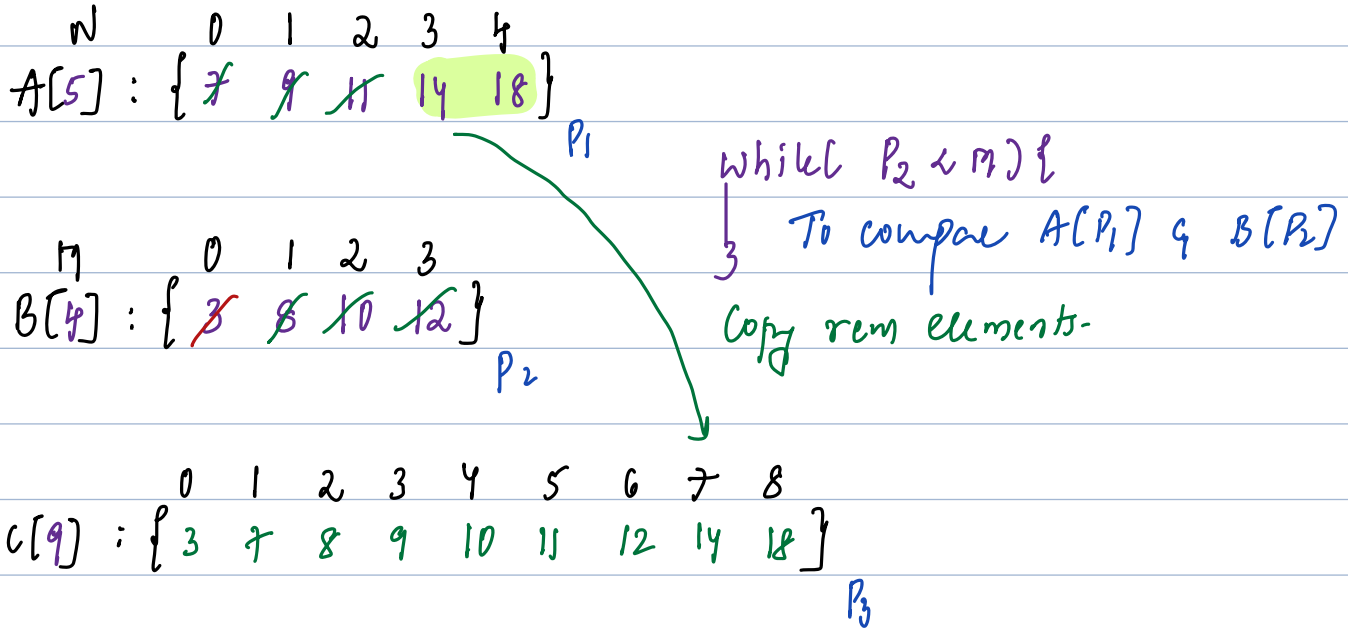
TC: $O(N + M + (N+M)\log(N+M)) = O((N+M)\log(N+M))$

#Idea2: At each iteration cmp smallest in A[] & B[]

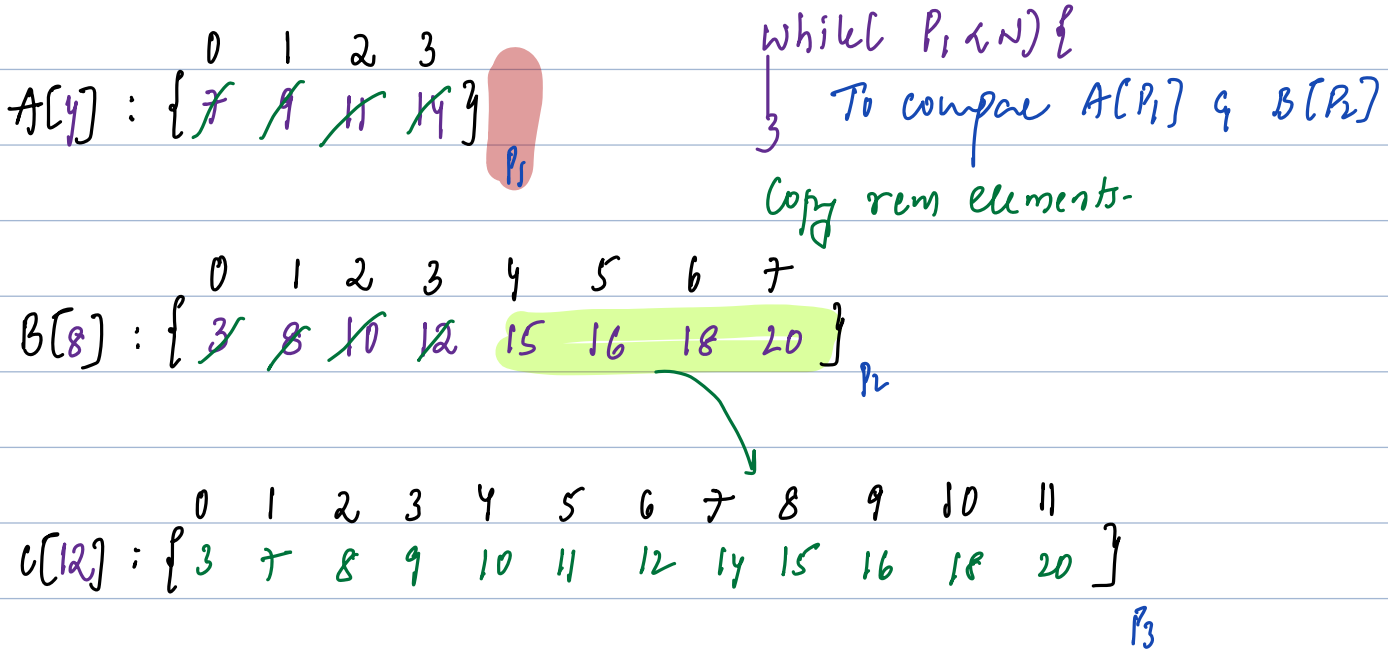
Note1: To track smallest in A[] & B[] use 2 variables

Note2: To track index in C[] use 1 variable

#Dry Run1:



#Dry Run2:



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

```
vector<int> merge(vector<int>&A, vector<int>&B){

    int N = A.size(), M = B.size();
    vector<int> C(N+M, 0);

    int P1=0, P2=0, P3=0;
    while(P1<N && P2<M){ # Both P1 & P2 should be in array
        if(A[P1] < B[P2]){ C[P3] = A[P1]; P3++; P1++; }
        else{ C[P3] = B[P2]; P3++; P2++;} # A[P1] >= B[P2]
    }

    while(P1<N){ # if P1 != N; Elements left out in A[]
        C[P3] = A[P1]; P3++; P1++;
    }

    while(P2<M){ # if P2 != M; Elements left out in B[]
        C[P3] = B[P2]; P3++; P2++;
    }

    return C;

}
```

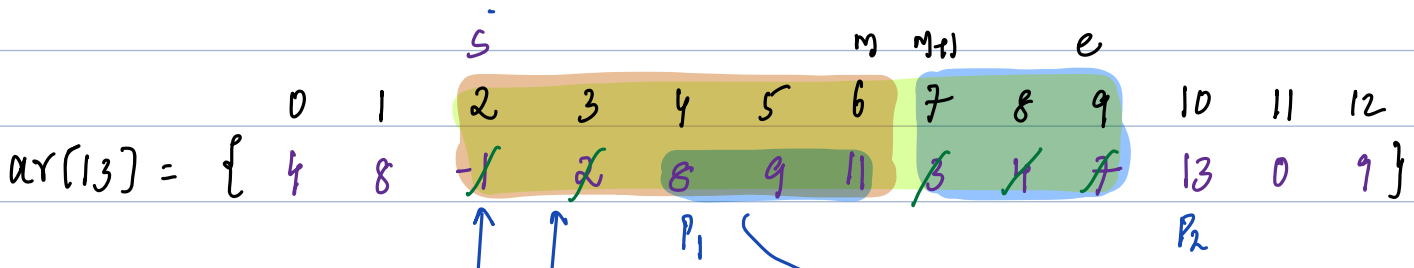
38: Merge 2 consecutive sorted subarrays #Include

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

#Subarray [s..m] is sorted

#Subarray [m+1..e] is sorted # s..m m+1..e

Sort entire subarray from [s..e] in ar[]

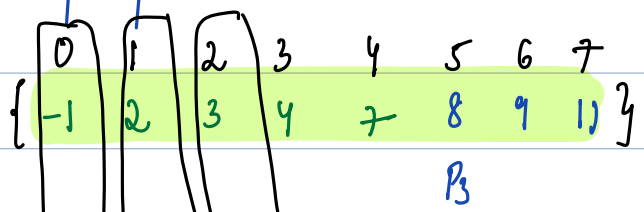


s m e

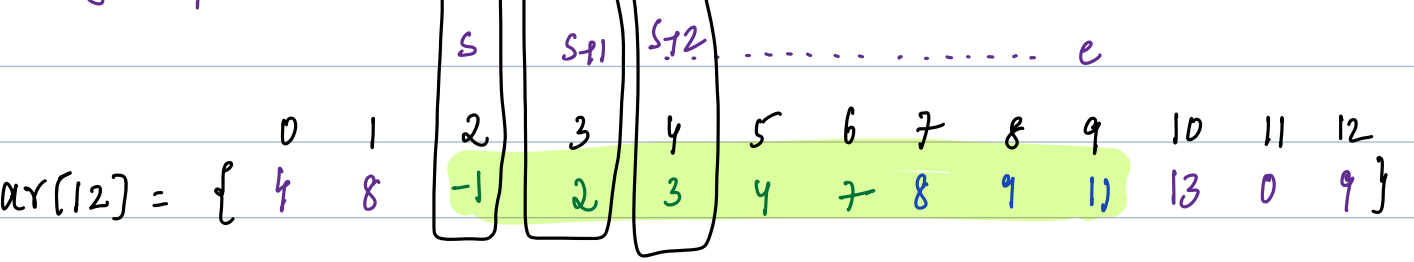
2 6 9

tmp[e-s+1] =

tmp[8] =



#Copy tmp[ ] to ar[s..e]

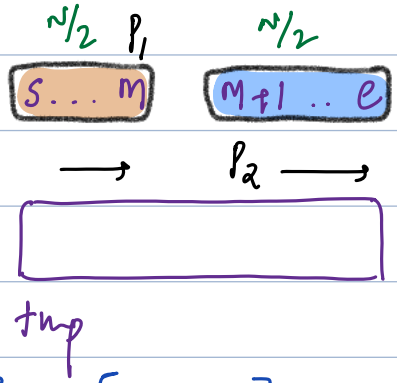


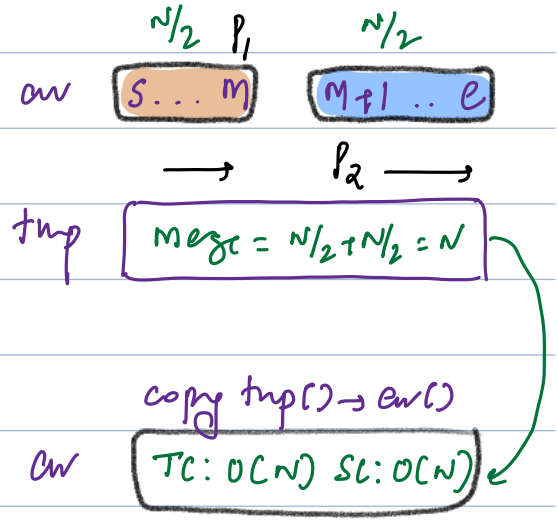
#arr : {s..m} is sorted {m+1..e} is sorted

Sort entire subarray {s..e}

TC: O(N+N) = O(N) SC: O(N)

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

    int c[e-s+1];
    int p1=s, p2=m+1, p3=0;

    while(p1<=m && p2<=e){ # 1st sub: [s..m]  2nd sub = [m+1..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[] ---> A[s..e]

    for(int i=s;i<=e;i++){
        A[i]= c[i-s];
    }

}
```





#Dry Run:

i	A[i] = c[i-s]
s	A[s] = c[s-s], c[0]
s+1	A[s+1] = c[s+1-s], c[1]
s+2	A[s+2] = c[s+2-s], c[2] ..