

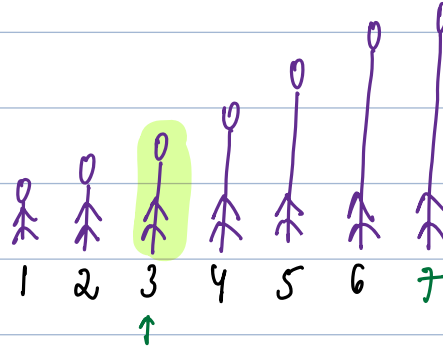
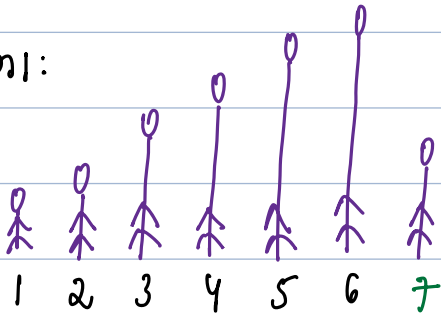
## Today's Content

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

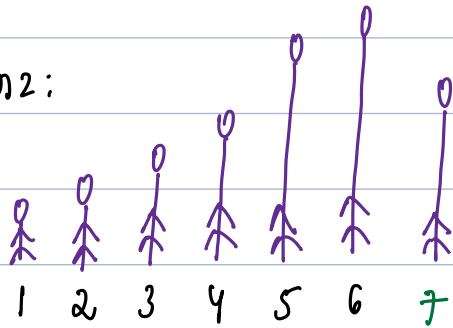
## Insertion Sort:

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

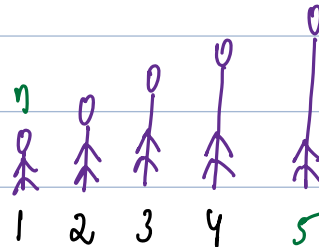
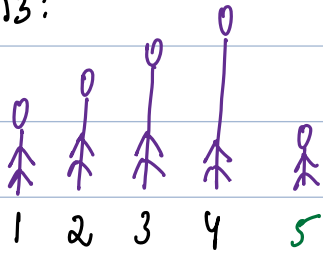
fn1:



fn2:



fn3:



while (  $n\_per$  not at start  $\&\&$   $p\_per > n\_per$  ) {

    swap  $p\_per$   $\&\&$   $n\_per$

}

Insertion Sort:

$arr[] = \{ 10 \mid 9 \quad 4 \quad 8 \quad 6 \quad 2 \}$

Pass Run:

$i=0$     Sort    Insert     $arr[] = \{ 9 \mid 10 \mid 4 \quad 8 \quad 6 \quad 2 \}$   
 $j \leftarrow j$

$i=1$     Sort    Insert     $arr[] = \{ 4 \mid 9 \mid 10 \mid 8 \quad 6 \quad 2 \}$   
 $j \leftarrow j \leftarrow j$

$i=2$     Sort    Insert     $arr[] = \{ 4 \mid 8 \mid 9 \mid 10 \mid 6 \quad 2 \}$   
 $j \leftarrow j \leftarrow j$

$i=3$     Sort    Insert     $arr[] = \{ 4 \mid 6 \mid 8 \mid 9 \mid 10 \mid 2 \}$   
 $j \leftarrow j \leftarrow j \leftarrow j$

$i=4$     Sort    Insert     $arr[] = \{ 2 \mid 4 \mid 6 \mid 8 \mid 9 \mid 10 \}$   
 $j \leftarrow j \leftarrow j \leftarrow j \leftarrow j$

Note: Stability maintained in insertion sort

void Insertion(int ar[], int N) { TC:  $O(N^2)$  SC:  $O(1)$ : Anplau

for(int i=0; i < N; i++) {

# [0..i] sorted insert ar[i];

int j = i-1;

while(j >= 0 && ar[j] > ar[j+1]) {

swap ar[j] & ar[j+1];

j--; # update new position

}

}

}

TODO: Take an almost sorted array & apply Bubble Sort, Selection Sort, Insertion Sort & compare iterations

# Final Observations:

Decreasing order of speed

Insertion Sort > Bubble Sort > Selection Sort

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

$A[4] : \begin{matrix} 0 & 1 & 2 & 3 \\ \{ 7 & 10 & 11 & 14 \} \end{matrix}$

$A[4] : \begin{matrix} 0 & 1 & 2 \\ \{ 3 & 6 & 10 \} \end{matrix}$

$B[3] : \begin{matrix} 0 & 1 & 2 \\ \{ 3 & 8 & 9 \} \end{matrix}$

$B[3] : \begin{matrix} 0 & 1 & 2 & 3 \\ \{ 5 & 14 & 20 & 25 \} \end{matrix}$

$C[7] : \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \{ 3 & 7 & 8 & 9 & 10 & 11 & 14 \} \end{matrix}$

$C[7] : \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \{ 3 & 5 & 6 & 10 & 14 & 20 & 25 \} \end{matrix}$

# Idea:

1. Create  $C[N+M]$

2. Copy  $A[] \rightarrow C$  &  $B[] \rightarrow C$

3. Sort  $C[]$  - /BS/AS/SS

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

→ Inbuilt Sort

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

# Idea 2: At each iteration copy smallest in A[] & B[]

Note 1: To track smallest in A[] & B[] use 2 variables

Note 2: To track index in C[] use 1 variable

# Dry Run 1:

N  
A[5]: { ~~7~~ ~~9~~ ~~11~~ 14 18 }

P<sub>1</sub>

M  
B[4]: { ~~3~~ ~~8~~ ~~10~~ ~~12~~ }

P<sub>2</sub>

while (P<sub>2</sub> < M) {

} To compare A[P<sub>1</sub>] & B[P<sub>2</sub>]

Copy rem elements-

C[9]: { 3 7 8 9 10 11 12 14 18 }

P<sub>3</sub>

# Dry Run 2:

A[4]: { ~~7~~ ~~9~~ ~~11~~ ~~14~~ }

P<sub>1</sub>

while (P<sub>1</sub> < N) {

} To compare A[P<sub>1</sub>] & B[P<sub>2</sub>]

Copy rem elements-

B[8]: { ~~3~~ ~~8~~ ~~10~~ ~~12~~ 15 16 18 20 }

P<sub>2</sub>

C[12]: { 3 7 8 9 10 11 12 14 15 16 18 20 }

P<sub>3</sub>

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;

}

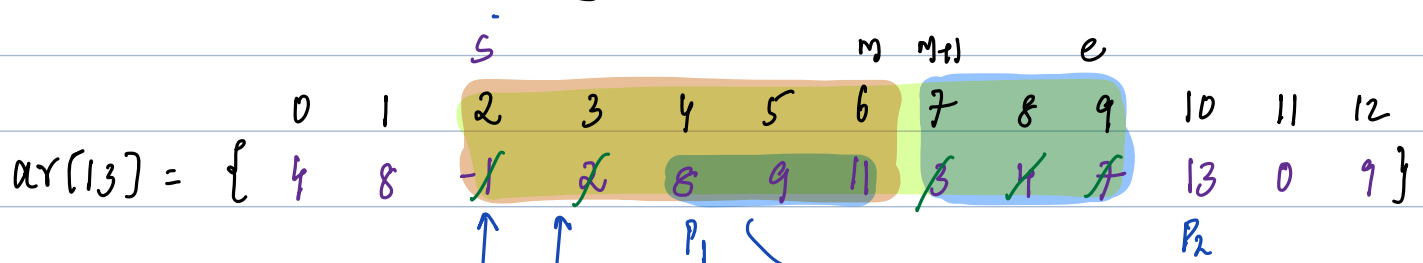
38: Merge 2 consecutive sorted subarrays # Include

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

$tmp[e-s+1] =$   
 $tmp[8] =$

0	1	2	3	4	5	6	7
-1	2	3	4	7	8	9	11

Diagram shows the temporary array  $tmp$  and the indices  $s, m, e$  used to copy the sorted subarray back to the original array.

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

Diagram shows the copying process where the sorted subarray from  $tmp$  is copied back to  $ar[s..e]$ .

Diagram showing the final state of the array  $ar[12]$  after merging the sorted subarrays.

0	1	2	3	4	5	6	7	8	9	10	11	12
4	8	-1	2	3	4	7	8	9	11	13	0	9



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

Sort entire subarray {s..e}

T.C:  $O(N \log N) = O(N)$  S.C:  $O(N)$

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

int c[e-s+1];

int p1=s, p2=m+1, p3=0;

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

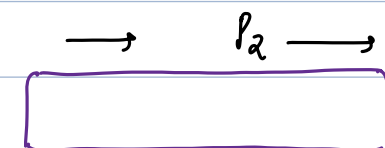
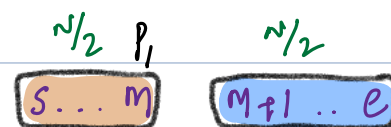
# Copy c[]  $\rightarrow$  A[s..e]

for(int i=s; i <= e; i++) {

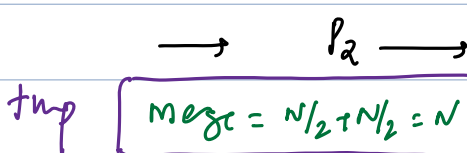
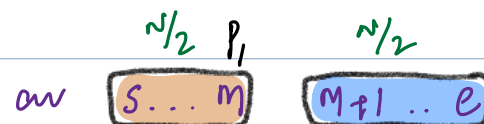
A[i] = c[i-s];

}

}



tmp



tmp

copy tmp[]  $\rightarrow$  arr[]

T.C:  $O(N)$  S.C:  $O(N)$

#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] ..