

Todays Content: → 9:05 pm Start

- a) Sorting Intro
- b) Sorting Algorithms
 - a) Bubble Sort
 - b) Insertion Sort
 - c) MergeSort → Recursion 9:07pm

New Students:

- a) Pre-Requests 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 = factor

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	Name	Marks
→ Mohit	0	mohit	0
Aman	100	kajal	32
Shrush	45 ↪	S Algo1: Aayush 45	S Algo2: Shrush 45
Kajal	32	Shrush 45	Aayush 45
Aayush	45 ↪		
Aditya	92		

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 :
Inplace

Bubble Sort:

1

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:

ite₁ { 2 4 -1 6 7 5 8 -1 10 } last 2 ele in correct pos.

ite₂ { 2 -1 4 6 5 7 -1 8 10 } last 2 ele in correct pos.

ite₃ { -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(N²) 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, $ar[j] > ar[j+1] \Rightarrow ar[N-1] > ar[N]$: Err

if ($ar[j] > ar[j+1]$) {

Swap $ar[j]$ & $ar[j+1]$

swap++;

Dry Run = 0 1 2 3 4

$ar[5] = \{ 3 1 6 10 8 \}$

i=0 | it₁ = 1 3 6 8 10 : 2 swaps

i=1 | it₂ = 1 3 6 8 10 : 0 swaps

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

}

} if (swap == 0) {

} // data sorted;

} break;

Check Stability: Parameters same: Relative order maintained

0 1 2 3

$ar[4] = \{ 2 4 2 1 \}$

BubbleSort

ite₁ = 2 2 1 4

a) StableSort

ite₂ = 2 1 2 4

b) InplaceSort

ite₃ = { 1 2 2 4 }

Insertion Step:

Given an $\text{arr}[N]$, first $N-1$ elements are sorted, sort entire $\text{arr}[]$.

Note: No Extra Space is allowed.

Examples:

$0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5$ $\text{arr}[6] = \{ 2 \quad 6 \quad 10 \quad 14 \quad 20 \quad 4 \}$ After Sorting $= \{ 2 \quad 4 \quad 6 \quad 10 \quad 14 \quad 20 \}$	$N=8$ $\text{arr}[8] = \{ 2 \quad 7 \quad 12 \quad 19 \quad 24 \quad 30 \quad 34 \quad 4 \}$ $\text{arr}[7] = \{ 2 \quad 4 \quad 7 \quad 12 \quad 19 \quad 24 \quad 30 \quad 34 \}$
---	---

 $0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$ $\text{arr}[7] = \{ 3 \quad 6 \quad 10 \quad 14 \quad 20 \quad 24 \quad 7 \}$ $= \{ 3 \quad 6 \quad 7 \quad 10 \quad 14 \quad 20 \quad 24 \}$

Tracing: $[0 \dots 5]$ sorted:

$j \quad \text{arr}[j] > \text{arr}[j+1] \quad T/F \quad \text{Swap}$ $5 \quad \text{arr}[5] > \text{arr}[6] \quad T \quad \text{Swap}$ $4 \quad \text{arr}[4] > \text{arr}[5] \quad T \quad \text{Swap}$ $3 \quad \text{arr}[3] > \text{arr}[4] \quad T \quad \text{Swap}$ $2 \quad \text{arr}[2] > \text{arr}[3] \quad T \quad \text{Swap}$ $1 \quad \text{arr}[1] > \text{arr}[2] \quad F \quad \text{break}$

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

```
void SortStep(int arr[]){
    int N = arr.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 (arr[j] > arr[j+1]) {
        swap arr[j] & arr[j+1];
    } else {
        break;
    }
}
```

3

Insertion Step:

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

3Q) Given $\text{arr}[N]$ sort it using Insertion Step.

$\text{arr}[] =$	0	1	2	3	4	5
	10	3	6	8	2	5

Step 1: $[0-0]$ sorted, Insert $\text{arr}[1]$, Sort $[0-1]$: Insertion Step

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

Step 2: $[0-1]$ sorted, Insert $\text{arr}[2]$, Sort $[0-2]$ Insertion Step

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

Step 3: $[0-2]$ sorted, Insert $\text{arr}[3]$, Sort $[0-3]$ Insertion Step

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

Step 4: $[0-3]$ sorted, Insert $\text{arr}[4]$, Sort $[0-4]$ Insertion Step

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

Step 5: $[0-4]$ sorted, Insert $\text{arr}[5]$, Sort $[0-5]$ Insertion Step

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

void insertionSort (int arr[]) { TC: O(N²) SC: O(1) } \rightarrow Inplace Sorting

int N = arr.length;

stable \rightarrow TODO / Yes Try Example.

for (int i=1; i < N; i++) {

// Insert $\text{arr}[i]$ in sorted data $\{0, \dots, i-1\}$ sorted, Insertion Step.

for (int j=i-1; j >= 0; j--) {

$\{0, 1, 2, 3, \dots, i-2, i-1, i\}$

if ($\text{arr}[j] > \text{arr}[j+1]$) {

swap $\text{arr}[j]$ & $\text{arr}[j+1]$

else { break }

}

}

Q8) Given 2 sorted arrays $A[N]$, $B[M]$, create $C[N+M]$ which contains overall sorted data.

$$A[4] = \underline{7 \ 10 \ 11 \ 14}$$

$$A[3] = \underline{3 \ 6 \ 10}$$

$$B[3] = \underline{3 \ 8 \ 9}$$

$$B[3] = \underline{5 \ 8 \ 14}$$

$$C[7] = \underline{3 \ 7 \ 8 \ 9 \ 10 \ 11 \ 14}$$

$$C[6] = \underline{3 \ 5 \ 6 \ 8 \ 10 \ 14}$$

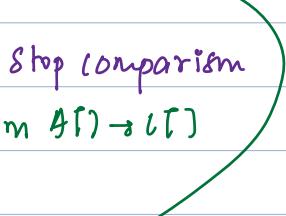
Eg1: P_1

$$\begin{array}{ccccccc} & & & & & & \\ 0 & 1 & 2 & 3 & 4 & 5 & \\ \hline A[6] = & \cancel{7} & \cancel{8} & \cancel{9} & 14 & 19 & 21 \end{array}$$

$B[5] = \cancel{3} \ \cancel{8} \ \cancel{8} \ \cancel{14} \ \cancel{19}$ P_2 : Out of index B, Stop comparison
Add rem ele from $A[1] \rightarrow C[1]$

$$C[11] = \underline{2 \ 3 \ 5 \ 6 \ 8 \ 9 \ 11 \ 13 \ 14 \ 19 \ 21}$$

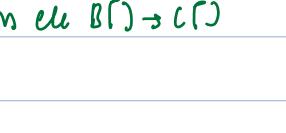
while($P_1 < N$) {
 |
 3
 | } 

while($P_2 < M$) {
 |
 *
 | } 

Eg2:

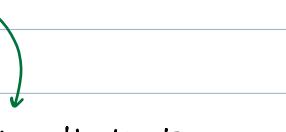
$$A[6] = \cancel{7} \ \cancel{8} \ \cancel{X} \ \cancel{14} \ \cancel{19} \ \cancel{21} \quad P_1 : \text{Out of index A: Stop comp}$$

Add rem ele $B[1] \rightarrow C[1]$

while($P_1 < N$) {
 |
 *
 | } 

$$B[8] = \cancel{3} \ \cancel{8} \ \cancel{8} \ \underline{11 \ 13 \ 17 \ 20 \ 24}$$

$$C[14] = \underline{1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 8 \ 8 \ 10 \ 11 \ 13 \ 17 \ 20 \ 24}$$

while($P_2 < M$) {
 |
 *
 | } 

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;

508) Given $\text{arr}[N]$ elements & 3 indices s, m, e

→ Subarray $[s \ m]$ is sorted

→ Subarray $[m+1, e]$ is sorted

Sort Entire Subarray from [s...e]

A diagram illustrating a stack structure. It consists of four colored rectangular boxes arranged horizontally. From left to right: a green box labeled 'S', a blue box labeled 'M', a light blue box labeled 'M+1', and a purple box labeled 'e'. To the right of the 'e' box is a bracket spanning all four boxes, with the label 'n-1' positioned above it. Below the green and blue boxes is a dashed horizontal line, and below the light blue and purple boxes is another dashed horizontal line.

The diagram shows the state of the array after the first few comparisons. The array has 12 slots, indexed 0 to 11. Elements at indices 2, 3, 4, 5, and 6 are crossed out as they are greater than 12. The search continues to index 7, where it finds the value 5. A red circle highlights the value 5 at index 7.

$$\# \text{ele} = l - S + 1 \quad 0 \quad 1 \quad x \quad x \quad 4 \quad 5 \quad 6 \quad 7 \quad 8$$

tmp[9] -1 2 3 4 7 8 9 11 13;

Copy $\text{tmp}[\cdot]$ \Rightarrow $\text{arr}[s \dots e]$

		<i>s</i>		<i>m</i>		<i>e</i>
0	1	2	3	4	5	6
4	8	-1	2	3	4	7

PseudoCode:

$\rightarrow A[]$: Subarray $[s \dots m]$ is sorted
 : Subarray $[m+1 \dots e]$ is sorted
 : Sort entire subarray from $[s \dots e]$

void merge(int A[], int s, int m, int e) { $Tc: O(N)$ $Sc: O(N)$ }

$\text{int temp}[e-s+1]; // [s \underset{P_1}{\cancel{s+1 \dots m}} \underset{P_2}{\cancel{m+1 \dots e}}]$
 $\text{int } P_1 = s, P_2 = m+1, P_3 = 0;$

$\text{while}(P_1 <= m \text{ and } P_2 <= e) \{ // \text{Compare till both are valid}$
 $\quad \text{if } (A[P_1] < A[P_2]) \{ \text{temp}[P_3] = A[P_1]; P_1++; P_3++ \}$
 $\quad \text{else} \{ \text{temp}[P_3] = A[P_2]; P_2++; P_3++ \}$

$\text{while}(P_1 <= m) \{ // \text{Copy rem ele to temp[]}$
 $\quad \text{temp}[P_3] = A[P_1]; P_1++; P_3++$

$\text{while}(P_2 <= e) \{ // \text{Copy rem ele to temp[]}$
 $\quad \text{temp}[P_3] = A[P_2]; P_2++; P_3++$

$\text{// temp[] sorted } \xrightarrow{\text{Copy}} A[s \dots e]$
 $\text{int } k = 0;$
 $\text{for } (\text{int } i=s; i<=e; i++) \{$
 $\quad A[i] = \text{temp}[k];$
 $\quad k++;$

$\text{temp}[e-s+1]:$	0	1	2	$e-s$	TODD
	\downarrow	\downarrow	\downarrow			\downarrow
$A[s \dots e]:$	s	$s+1$	$s+2$	e	
	\downarrow	\downarrow	\downarrow			

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

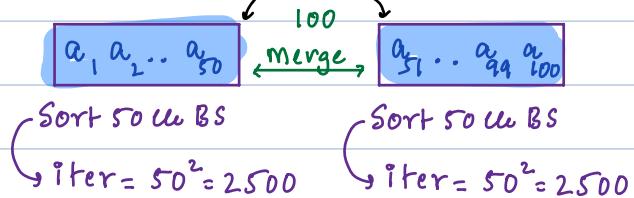
1

Given $a[100]$ elements sort them.

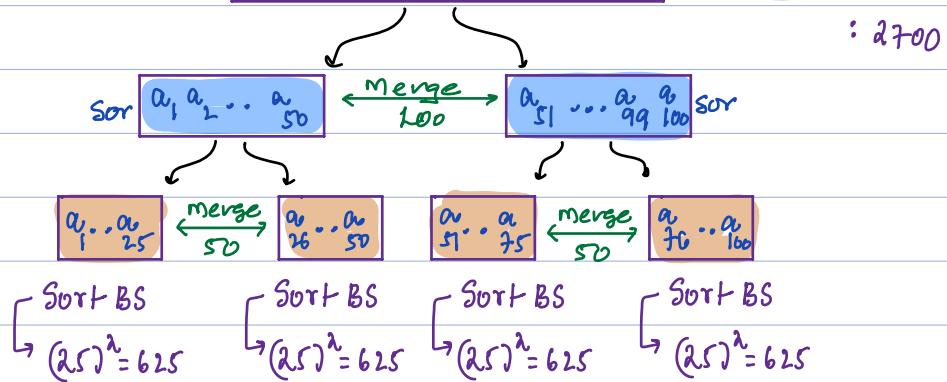
Case1 : $a_1 a_2 a_3 \dots a_{50} a_{51} \dots a_{99} a_{100}$ Iterations : 10^4 iterations

1. Apply IS/BS to sort all ele = $O(N^2) = 100^2 = 10000 = 10^4$

Case2 : $a_1 a_2 a_3 \dots a_{50} a_{51} \dots a_{99} a_{100}$ Iterations : $2500 + 2500 + 100$
: 5100 iterations



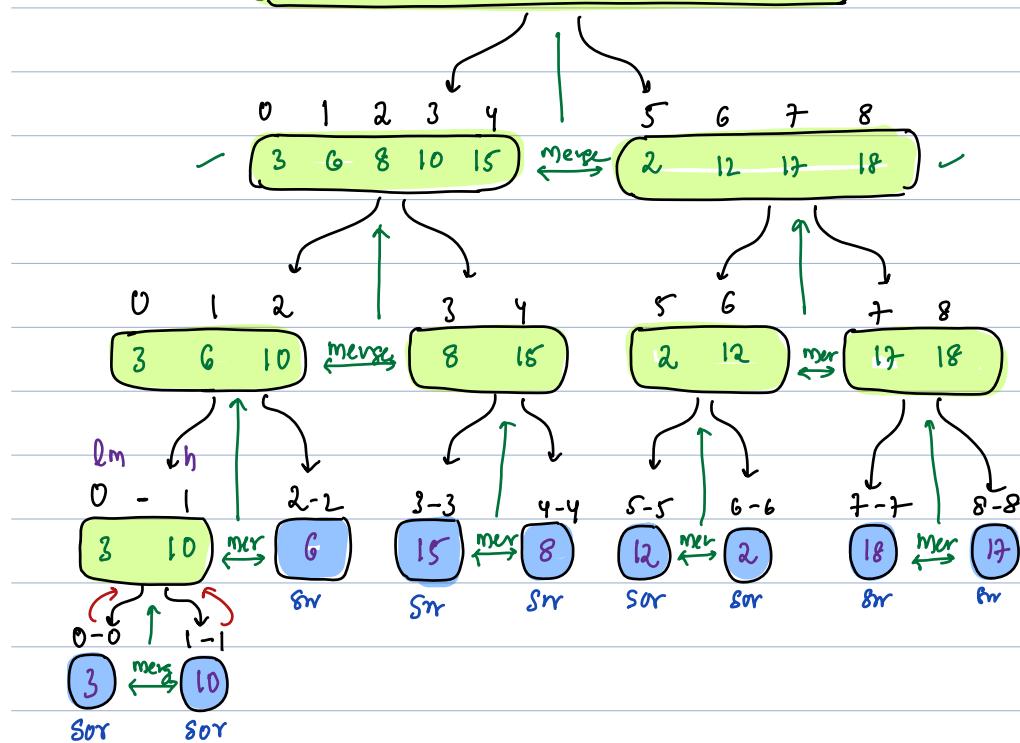
Case3 : $a_1 a_2 a_3 \dots a_{50} a_{51} \dots a_{99} a_{100}$ Iterations : $625 * 4 + 50 * 2 + 100$



Idea: keep dividing arr(), till it contains 1 element & merge them.

Idea: $l = 0, 1, 2, 3, 4, 5, 6, 7, 8, h = m + 1$ $m = \frac{l+h}{2} = \frac{0+8}{2} = 4$

$ar[9] : [2, 3, 6, 8, 10, 12, 15, 17, 18]$



int[] solve(int ar[]) {

```

int n = ar.length;
mergeSort(ar, 0, N-1) // ar[] is sorted
return ar
    }
```

1. Ass: Sort arr[] from [s → e]

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

```
    if(s == e) { return } → 1
```

```
    int m = (s+e)/2 → 1
```

```
    mergeSort(arr, s, m) → f(N/2)
```

```
    mergeSort(arr, m+1, e) → f(N/2)
```

```
    merge(arr, s, m, e) → N
```

```
}
```

ar: [s → sorted ← e] $m = s + e / 2$



sort subproblems

TC: Assume Sort N ele takes = $f(N)$

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

TC: $O(N \log N)$ SC: $O(N)$

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

```
    int temp[e-s+1]; //  $\underbrace{[s \dots s+1 \dots m]}_{P_1} \quad \underbrace{[m+1 \dots e]}_{P_2}$ 
```

```
    int P1 = s, P2 = m+1, P3 = 0;
```

```
    while(P1 <= m && P2 <= e) { // Compare till both are valid
```

```
        if(A[P1] < A[P2]) { temp[P3] = A[P1]; P1++; P3++ }
```

```
    } else { temp[P3] = A[P2]; P2++; P3++ }
```

```
    while(P1 <= m) { // Copy rem ele to temp[]
```

```
        temp[P3] = A[P1]; P1++; P3++ }
```

```
    while(P2 <= e) { // Copy rem ele to temp[]
```

```
        temp[P3] = A[P2]; P2++; P3++ }
```

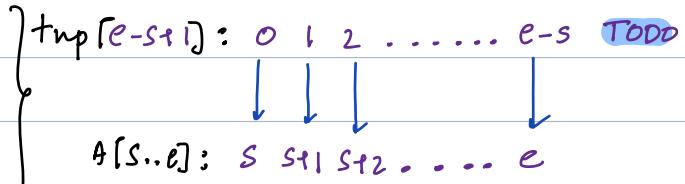
```
// temp[] sorted  $\xrightarrow{\text{Copy}} A[s \dots e]$ 
```

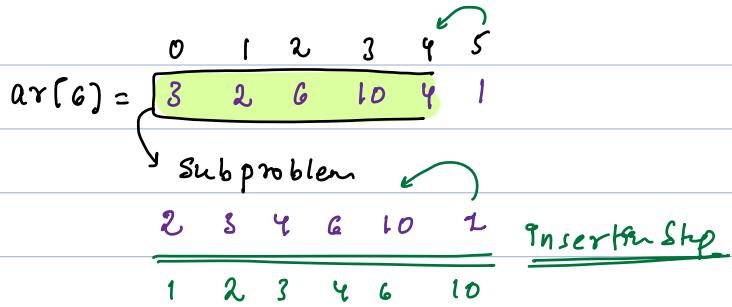
```
int k = 0;
```

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

```
    A[i] = temp[k];
```

```
    k++;
}
```





Ass: Sort $ar[]$ from $[0..e]$

```
void insertionRec(int ar[], int e) {
    if (e == 0) { // single element return}
        insertionRec(ar, e-1) // sort ar[0..e-1]
    for (int j = e-1; j >= 0; j--) { // in sorted [0..e-1] → ar[e]
        if (ar[j] > ar[j+1]) {
            swap ar[j] & ar[j+1]
        } else { break }
    }
    // entire ar[] from [0..e] is sorted
}
```

3

int[] solve(int ar[]) {

```
    int n = ar.length;
    insertionRec(ar, n-1);
    return ar
}
```

3

