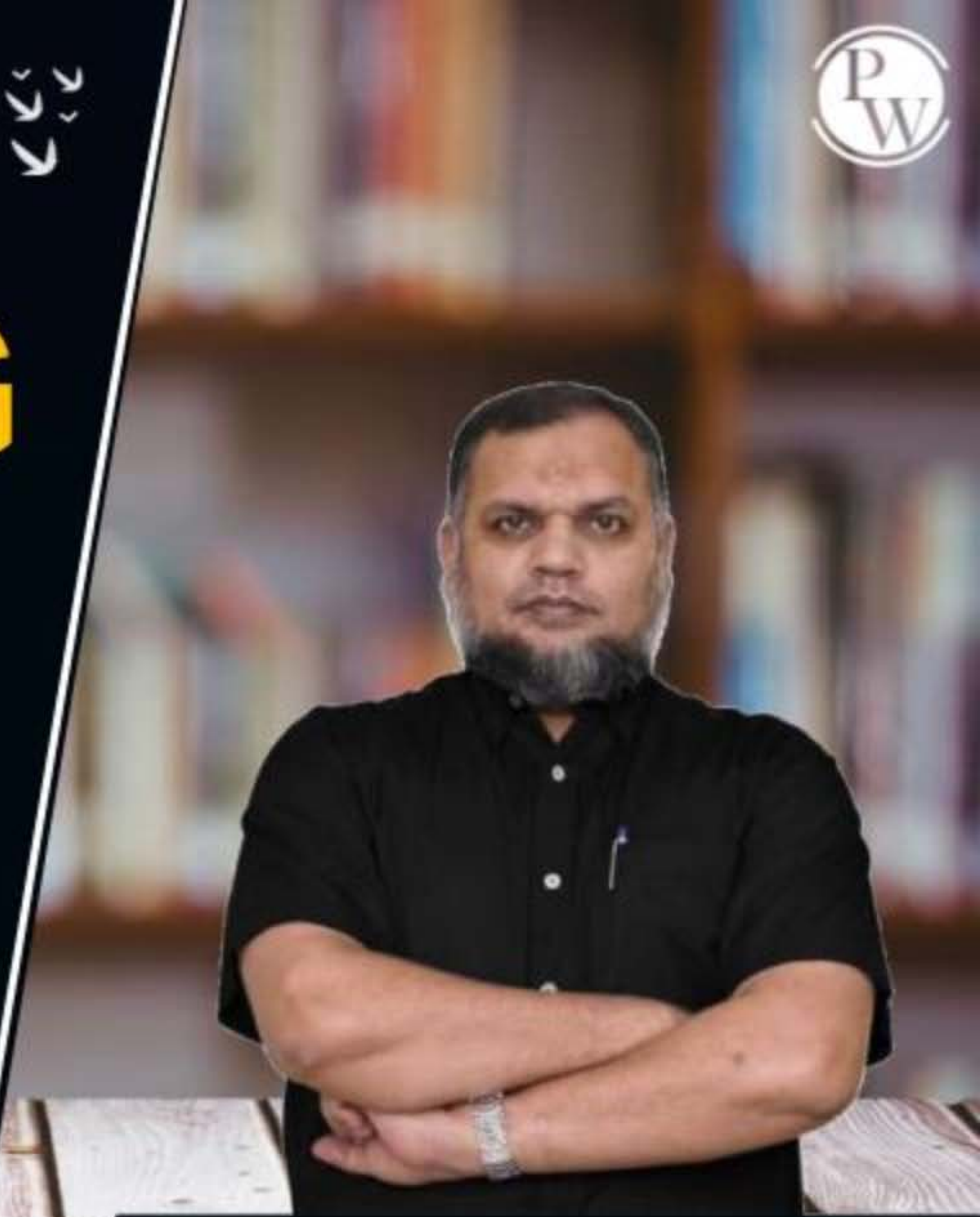
CS & IT ENGINEERING

Algorithms

Graph Algorithms

Lecture No.- 03



By- Dr. Khaleel Khan Sir

Recap of Previous Lecture

Topic Graph Techniques

Topics to be Covered

Topic Sorting Methods

SORTING TECHNIQUES

Classification :

(i) Internal vs External Sort

(ii) ✓Comparison vs Non-Comparison based

<Radix Sort>

(iii) Recursive vs Iterative

(iv) Inplace vs NOT-IN-PLACE

<Space Reqd in gen $O(1)$

or atmost $O(\log n)$

for Rec. Stack>

↳ Merge Sort : $O(n)$

(v) Stable vs Unstable ⟨QS; Heap Sort⟩

If the relative order of Non-Distinct elements are maintained;

Let $A[1..n]$ be an array of n-elements

Let i, j be two indices $(i \neq j)$

if $(A[i] = A[j])$ & if $A[i]$ precedes $A[j]$ in the Pre-Sorted list, then the Sorting method is Stable, iff $A[i]$ Precedes $A[j]$ in the Final Sorted list also;

$A[2] = A[4]$

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

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

⟨Stable⟩

Parameters that influence the Time Complexity of Comparison Based Sort

Time-Complexity (Comp. based Sort) = f(No. of Comparisons ; No. of Swaps)

(Inversion of an Array)

as a size of Input (n)

$f(n^2 ; n\log n)$

Time = $O(n^2)$

Mergesort

↓

$\Theta(n \cdot \log n)$

Swaps Comparisons

Inversion of an Array :

→ Let $A[1 \ldots n]$ be an array, 'i' & 'j' be indices $(i \neq j)$

if $\big((i < j)$ and $(A[i] > A[j])\big)$ then the pair $\langle i, j \rangle$ is known as inversion of the array;

	1	2	3	4	5	6
A	8	9	4	5	1	2

$\langle 1,3 \rangle \langle 1,4 \rangle \langle 1,5 \rangle \langle 1,6 \rangle = 4$

$\langle 2,3 \rangle \langle 2,4 \rangle \langle 2,5 \rangle \langle 2,6 \rangle = 4$

$\langle 3,5 \rangle \langle 3,6 \rangle = 2$

$\langle 4,5 \rangle \langle 4,6 \rangle = 2$

$\underline{12}$ ✓

Q) Given an array of size 'n' elements

a) What is the Lower Bound (Min) # of Inversions :

	1	2	3	4	5
A	8	10	12	15	25

'0'

b) What is the Max (Upper Bound) # of Inversions :

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

$\langle 1,2\rangle \langle 1,3\rangle \langle 1,4\rangle \langle 1,5\rangle, \langle 1,6\rangle = 5$

$\langle 2,3\rangle \langle 2,4\rangle \langle 2,5\rangle \langle 2,6\rangle = 4$

$\langle 3,4\rangle \langle 3,5\rangle \langle 3,6\rangle = 3$

$\langle 4,5\rangle \langle 4,6\rangle = 2$

$\langle 5,6\rangle = 1$

'n'

$\rightarrow \boxed{\frac{n(n-1)}{2} = O(n^2)}$

BUBBLE SORT

```
Algorithm BUBBLE_SORT (A, n)
{                                   int flag;
    1. for pass ← n down to 1
       {                            flag = 0;
    2.     for i ← 1 to (pass –1)
           {
               if (A[i] > A[i + 1])
               {
                   swap(A[i], A[i + 1]);
                   flag=1;
               }
           }
       ⇒ if(flag==0) break;
       }
}
```

Inplace

Stable

Space : O(1)

A: ⟨ 80 60 20 15 40 10 ⟩ (indices 1 2 3 4 5 6)

It 1:

	1	2	3	4	5	6
A:	80	60	20	15	40	10
i = 1	60	80	20	15	40	10
i = 2	60	20	80	15	40	10
i = 3	60	20	15	80	40	10
i = 4	60	20	15	40	80	10
i = 5	60	20	15	40	10	$\boxed{80}$

It 2:

Time Complexity:

	Comparisons	Swaps	
1) Inc →	$n(n-1)/2$ →	0	$= O(n^2)$
2) Dec →	$n(n-1)/2$ →	$\frac{n(n-1)}{2}$	$= O(n^2)$

Slide 4

Optimized Bubble Sort (flag)

(i) Best Case : $O(n)$: Inc. order

(ii) Worst Case : $O(n^2)$: Dec. order

$(n^2 + n^2)$

Let $T(n)$ repr. Time Complexity of Bubble_Sort(n) (Worst Case)

B.C

$$\boxed{T(n) = n-1+c} = O(n)$$

$$\boxed{T(n) = n + T(n-1)} = O(n^2)$$

SELECTION SORT

1. Find the smallest element in the list.
2. Swap it with the value in the current ith position. $i = 1, n-1$
3. Repeat this process for all elements until the array is sorted.

$$\begin{array}{lcccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ A: & \langle 80 & 60 & 20 & 15 & 40 & 10 \rangle \\ i=1; & \langle \boxed{10} & 60 & 20 & 15 & 40 & 80 \rangle \\ i=2 & \langle \boxed{10} & \boxed{15} & 20 & 60 & 40 & 80 \rangle \end{array}$$

min ← ~~6~~ ~~2~~ ~~3~~ 4

min = ~~1~~ 4

Space complexity: $O(1)$

In place; Stable

Algorithm SELECTION_SORT (A, n) $(n-1)+(n-2)$

```
{
  1. for i ← 1 to n – 1 do
  {
      min ← i;
      for j ← i + 1 to n
      {
          if (A[j] < A[min]) then
          {
              min ← j;
          }
      }
      swap(A[min], A[i]);
  }
}
```

outer

Comp

'n'

Time

Comp's: $\frac{n(n-1)}{2}$, $O(n^2)$

Swaps: 'n'

: $O(n^2)$

Best worst

Note $\Longrightarrow$ Selection is the only Sorting Technique in Comparison based Sorting, that requires least no. of Swaps ($O(n)$) in the Worst-Case.

INSERTION SORT

[On-line Sort + External Sort]
↳ exhibit the property

– Effective for small data sets.

– If the input list is pre-sorted, then it takes time of O(n + d); d = number of inversions.

0, n^2

– It is On-line.

Each pass of insertion sort removes an element from the input data set, inserts it into the correct position in the already sorted list.

<Partial Sorted list> <Unsorted list>

$\langle a_1 \ a_2 \ \cdots \ a_k \rangle$ | x y k p t $\cdots\cdots$ online

File <disk>

Pass

$\langle a_1 \ a_2 \ \boxed{x} \ a_3 \ \cdots \ a_k \rangle$

Partial Sorted list

$A : \langle 8 \quad 2 \quad 4 \quad 9 \quad 3 \quad 6 \rangle$

$\quad\;\; 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$

	1	2	3	4	5	6
Pass 1	8	2	4	9	3	6
Pass 1	⟨2	8⟩	4	9	3	6
2	⟨2	4	8⟩	9	3	6
3	⟨2	4	8	9⟩	3	6
4	⟨2	3	4	8	9⟩	6
5	⟨2	3	4	6	8	9⟩

INSERTION SORT

Algorithm **INSERTION_SORT** (A, n)

```
{
  1.  for j ← 2 to n
      {
          key ← A[j];
          i ← j –1;
          while (i > 0 and A[i] > key) {
                A[i + 1] ← A[i];
                i ← i – 1;
          }
          A[i + 1] ← key;
      }
}
```

Stable
Inplace

Time - Complexity : opt. BS

1) Best Case : Inc. order : $O(n)$

⟨4 8 10 11 12 15⟩

– 0 – Swaps ; # of Inv's : 0

2) Worst Case :

→ decr. order

⟨18 15 10 8 6 2⟩

$n^2 + n^2 = O(n^2)$

Radix Sort < Non Comparison - based Sorting >

Base: $\langle r = 10 \rangle$

	1	2	3	4	5	6	7	8	9	10
A	723	64	99	83	124	4	7	65	333	545
Pass 1	723	83	333	64	124	4	65	545	7	99
Pass 2	4	7	723	124	333	545	64	65	83	99
Pass 3	4	7	64	65	83	99	124	333	545	723

Time: $O(d*(n+b)) \sim O(d*n)$

b = base; d = (No. of bits (digits for largest No))

Topic VII. Sorting Methods

Q). Which of the following Sorting algorithms has lowest worst-case complexity?

i. Bubble Sort : n^2
ii. Merge Sort : $n \log n$
iii. Quick Sort : n^2
iv. Selection Sort : n^2

Q). Which of the following in place Sorting algorithm needs minimum number of swaps? (W.C)

(a) Selection Sort
(b) Insertion Sort
(c) Heap Sort
(d) Quick Sort

Topic VII. Sorting Methods

Q). What would be the worst case complexity of Insertion Sort if the inputs are restricted to permutation of 1 to n with at most 'n' Inversions?

Time : $O(n+d)$

d = no. of Inversions

$d = n$

$O(n+n) = O(n)$ ✓

Topic VII. Sorting Methods

Inc

	1	2	3	4	5	6	7
A	5	8	10	15	21	40	85

Q). Let 'S' be a Sorted Array of 'n' integers and T(n) denote the time taken for the most efficient algorithm to determine if there are 2 elements in the Array with the sum <1000.

if (A[1] + A[2] < 1000) Print (yes)
else
if (A[n] + A[n-1] < 1000) Print (yes)
else Print (no)

	1	2	3	4	5	6	7
A	85	40	21	15	10	8	5

(a) T(n) is O(1)

(b) $n \le T(n) \le n \log n$

(c) $T(n) = {}^nC_2$ $O(n^2)$

(d) $n\log n <= T(n) <= {}^nC_2$

Q). The traditional Insertion Sort to Sort an Array can be of n elements uses linear search to identify the position where an element is to be inserted into already Sorted part of the Array, if instead binary search is used to identify the position of newly inserted element then the worst case complexity will be order of $O(n^2)$.

(LS) W.C: $n^2 + n^2 = O(n^2)$

(BS) W.C: $n \cdot \log n + n^2$

$O(n^2)$

L.S

Partial Sorted list | unsorted list

$\langle a_1\, a_2 \cdots a_k \rangle$	x

Topic VII. Sorting Methods

Q). Which one of the following in place sorting algorithms needs the minimum number of swaps?

(a) Quick sort

(b) Insertion sort

(c) Selection sort

(d) Heap sort

Topic VII. Sorting Methods

n^2 $n \log n$ n^2

Q). The worst case running times of Insertion sort, Merge sort and Quick sort, respectively, are:

(a) $\Theta(n \log n)$, $\Theta(n \log n)$, and $\Theta(n^2)$
(b) $\Theta(n^2)$, $\Theta(n^2)$, and $\Theta(n \log n)$
(c) $\Theta(n^2)$, $\Theta(n \log n)$, and $\Theta(n \log n)$
(d) $\Theta(n^2)$, $\Theta(n \log n)$, and $\Theta(n^2)$

(i) D and C Merge ✓

(ii) (Heap) ✓

Q). You have n lists, each consisting of m integers sorted in ascending order. Merging these lists into a single sorted list will take time:

H/W

(a) O(nm log n)

(b) O(mn log m)

(c) O(m + n) ×

(d) O(mn)

Min → Max

Q). If we use Radix Sort to sort n integers in the range $[n^{k/2}, n^k]$, for some k > 0 which is independent of n, the time taken would be

(a) $\Theta(n)$

(b) $\Theta(kn)$

~~(c)~~ $\Theta(n \log n)$ ✓

(d) $\Theta(n^2)$

$O(d * n)$

$O(n * \log n)$

$\log n^K$

$d = K * \log n$

$(n * m)$

2	3	4	8
1	5	6	10
7	9	11	15

Topic VII. Sorting Methods

Q). The worst case running times of Insertion sort, Merge sort and Quick sort, respectively are:

(a) $\Theta(n \log n)$, $\Theta(n \log n)$ and $\Theta(n^2)$

(b) $\Theta(n^2)$, $\Theta(n^2)$ and $\Theta(n \log n)$

(c) $\Theta(n^2)$, $\Theta(n \log n)$ and $\Theta(n \log n)$

(d) $\Theta(n^2)$, $\Theta(n \log n)$ and $\Theta(n^2)$

Topic VII. Sorting Methods

$n^3 \Rightarrow 3 \cdot \log n$

(n)

Q). Following algorithm(s) can be used to sort n integers in the range [1..n^3] in O(n) time

(a) Heap sort (b) Quick sort

(c) Merge sort (d) Radix sort

$O(d \cdot n)$

$O(3 \cdot n) = O(n)$

Q). For merging two sorted lists of sizes m and n into a sorted list of size m+n, we require comparisons of

(a) O(m) (b) O(n)

(c) O(m+n) (d) O(log m+ log n)

Topic VII. Sorting Methods

Q). Give the correct matching for the following pairs:

(A) $O(\log n)$	(P) Selection sort
(B) $O(n)$	(Q) Insertion sort
(C) $O(n \log n)$	(R) Binary search
(D) $O(n^2)$	(S) Merge sort

Topic VII. Sorting Methods

Q). Assume that the algorithms considered here sort the input sequences in ascending order. If the input is already in ascending order, which of the following are TRUE ?

I. Quicksort runs in $\Theta(n^2)$ time

II. Bubble sort runs in $\Theta(n^2)$ time

III. Merge sort runs in $\Theta(n)$ time

IV. Insertion sort runs in $\Theta(n)$ time

(a) I and II only

(b) I and III only

(c) II and IV only

(d) I and IV only

THANK - YOU