Sunbeam Institute of Information Technology
Pune and Karad

Algorithms and Data structures

Trainer - Devendra Dhande
Email – devendra.dhande@sunbeaminfo.com

arr	26	14	3	6	8	18	21	9	5
	1	2	3	4	5	6	7	8	9

SIZE	ci	pi
0		
1	1	0
2	~~2~~ 1	~~1~~ 0
3	3	1
4	~~4~~ ~~2~~ 1	~~2~~ ~~1~~ 0

18	14	6	9	8	3	5	21	26
1	2	3	4	5	6	7	8	9

SIZE	pi	ci
9		
8	~~1~~ ~~3~~ 7	~~2~~ 3 ~~6~~ ~~7~~, 14
7	~~1~~ 3	~~2~~ ~~3~~ ~~6~~ 7

Merge sort

1. Divide array in two parts
2. Sort both partitions individually (by merge sort only)
3. Merge sorted partitions into temporary array
4. Overwrite temporary array into original array

$m = \frac{l+r}{2}$ size $= (r-l)+1$

left partition $= l \rightarrow m$ (i)

right partition $= m+1 \rightarrow r$ (j)

6	1	9	7	3	8	2	4	5
0	1	2	3	4	5	6	7	8
l				m				r

6	1	9	7	3
0	1	2	3	4

8	2	4	5
5	6	7	8

i j

1	3	6	7	9
0	1	2	3	4

2	4	5	8
5	6	7	8

k

temp

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

No. of levels $= \log n$

No. of comps/level $\approx n$

Total comps $= n \log n$

Time $\propto$ comp

Time $\propto n \log n$

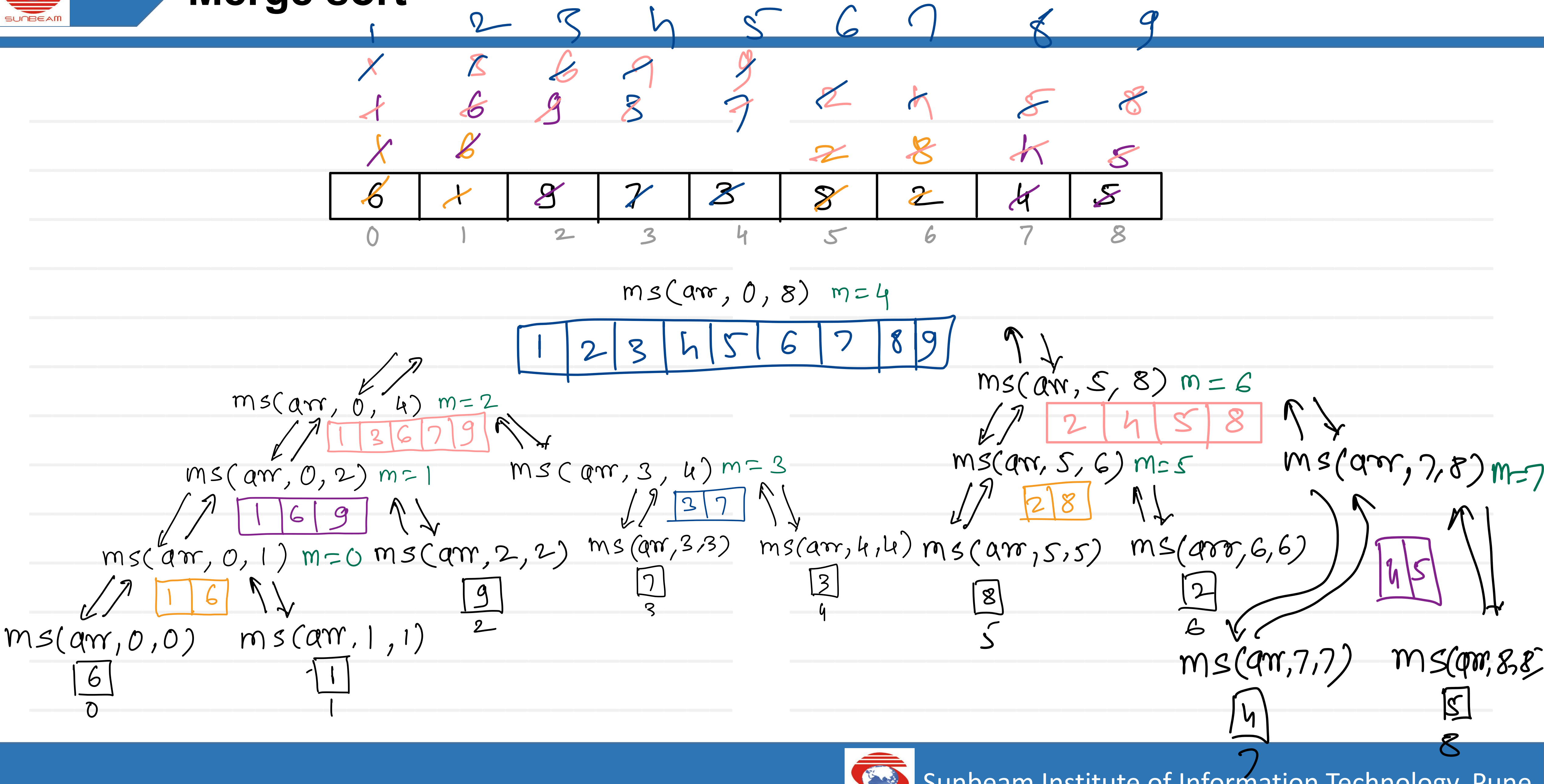
Best Avg Worst $\boxed{T(n) = O(n \log n)}$

temp[n] - auxillary space to merge array partitions

space $\propto n$

$\boxed{S(n) = O(n)}$

Merge sort



1 2 3 4 5 6 7 8 9
6 1 9 7 3 8 2 4 5
0 1 2 3 4 5 6 7 8
ms(arr, 0, 8) m=4
1 2 3 4 5 6 7 8 9
ms(arr, 0, 4) m=2
1 3 6 7 9
ms(arr, 5, 8) m=6
2 4 5 8
ms(arr, 0, 2) m=1
1 6 9
ms(arr, 3, 4) m=3
3 7
ms(arr, 5, 6) m=5
2 8
ms(arr, 7, 8) m=7
4 5
ms(arr, 0, 1) m=0
1 6
ms(arr, 2, 2)
ms(arr, 3, 3)
ms(arr, 4, 4)
ms(arr, 5, 5)
ms(arr, 6, 6)
ms(arr, 0, 0)
ms(arr, 1, 1)
ms(arr, 7, 7)
ms(arr, 8, 8)

Quick sort

1. Select pivot/axis/reference element from array
2. Arrange lesser elements on left side of pivot
3. Arrange greater elements on right side of pivot
4. Sort left and right side of pivot again (by quick sort)

1. extreme left / right element
2. middle element
3. median — random 3 element / random 5 element
4. dual pivot

No. of levels = $\log n$
comps / level = n
total comps = $n \log n$

Best
Avg
$\boxed{T(n) = O(n \log n)}$

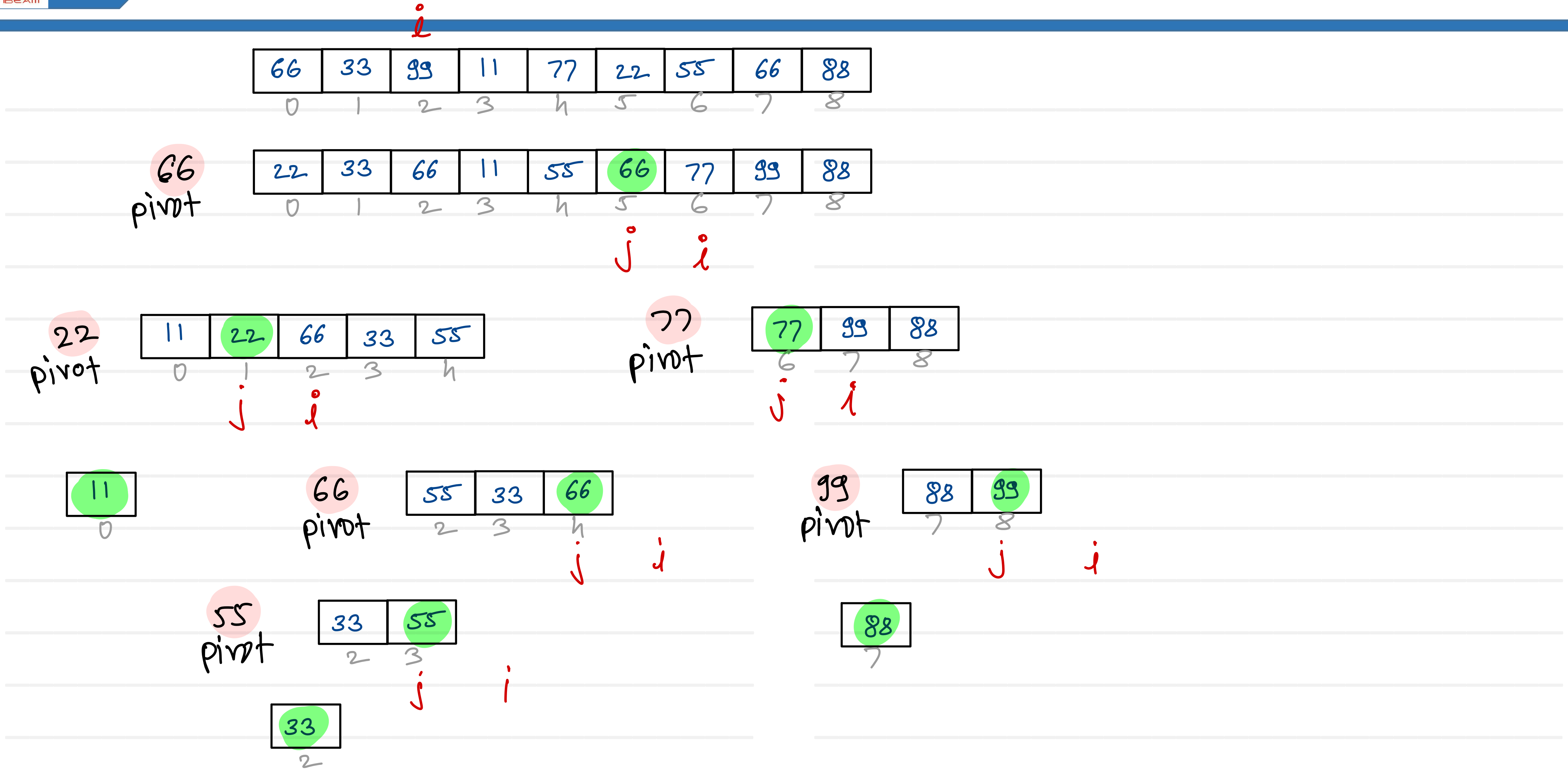
11 22 33 44 55
22 33 44 55
33 44 55
44 55
55

No. of levels = n
comps / level = n
Total comps = n^2

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

time complexity of quick sort is dependent on selection of pivot.

Quick sort



66 33 99 11 77 22 55 66 88
0 1 2 3 4 5 6 7 8
i
66 pivot
22 33 66 11 55 66 77 99 88
0 1 2 3 4 5 6 7 8
j i
22 pivot
11 22 66 33 55
0 1 2 3 4
j i
77 pivot
77 99 88
6 7 8
j i
11
0
66 pivot
55 33 66
2 3 4
j i
99 pivot
88 99
7 8
j i
55 pivot
33 55
2 3
j i
88
7
33
2

	Space	Time Best	Time Avg	Time Worst
Selection sort		$O(n^2)$	$O(n^2)$	$O(n^2)$
bubble sort		$O(n)$	$O(n^2)$	$O(n^2)$
insertion sort	$O(1)$ inplace sorting algorithm	$O(n)$	$O(n^2)$	$O(n^2)$
Heap sort		$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick sort		$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Merge sort	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

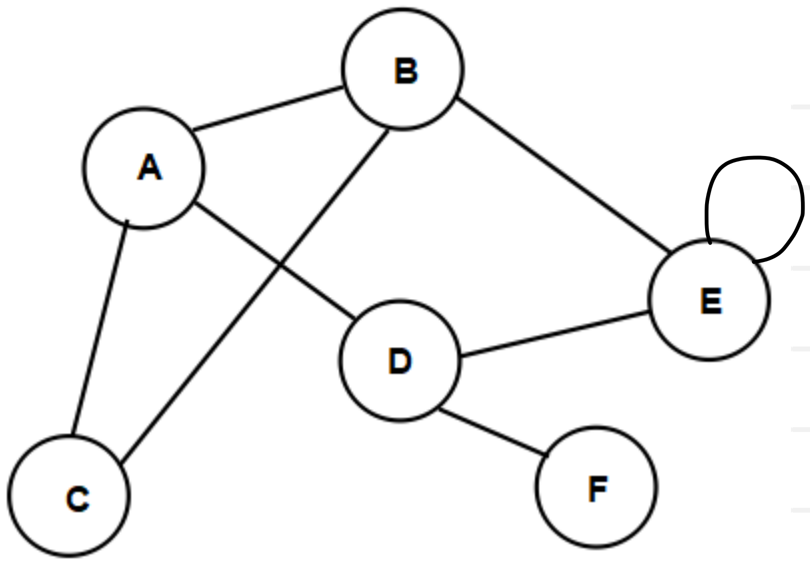
Graph : Terminologies

- **Graph** is a non linear data structure having set of vertices (nodes) and set of edges (arcs).
  - G = {V, E}

    Where V is a set of vertices and E is a set of edges
  - **Vertex (node)** is an element in the graph

    V = {A, B, C, D, E, F}
  - **Edge (arc)** is a line connecting two vertices

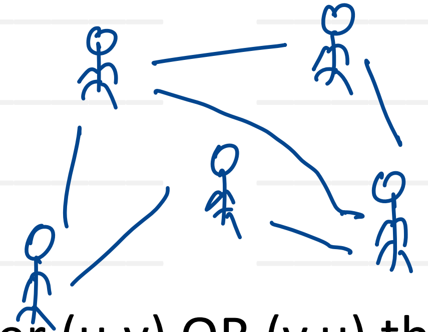
    E = {(A,B), (A,C), (B,C), (B,E), (D, E), (D,F),(A,D)}

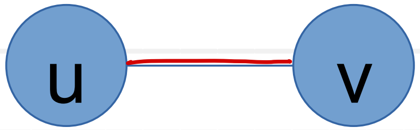


- Vertex A is set be **adjacent** to B, if and only if there is an edge from A to B.
- **Degree of vertex :-** Number of vertices adjacent to given vertex
- **Path :-** Set of edges connecting any two vertices is called as path between those two vertices.
  - Path between A to D = {(A, B), (B, E), (E, D)}
- **Cycle :-** Set of edges connecting to a node itself is called as cycle.
  - {(A, B), (B, E), (E, D), (D, A)}
- **Loop :-** An edge connecting a node to itself is called as loop. Loop is smallest cycle.

Graph : Types

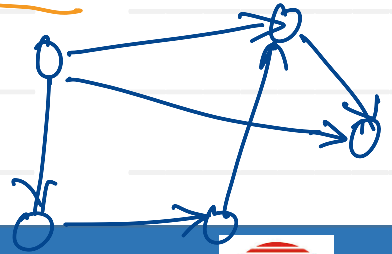
- **Undirected graph.**
  - If we can represent any edge either (u,v) OR (v,u) then it is referred as **unordered pair of vertices i.e. undirected edge.**
  - **graph which contains undirected edges referred as undirected graph.**

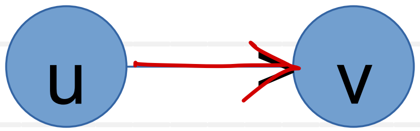




(u, v) == (v, u)

- **Directed Graph (Di-graph)**
  - If we cannot represent any edge either (u,v) OR (v,u) then it is referred as an **ordered pair of vertices** i.e. directed edge.
  - **graph which contains set of directed edges referred as directed graph (di-graph).**
  - graph in which each edge has some direction

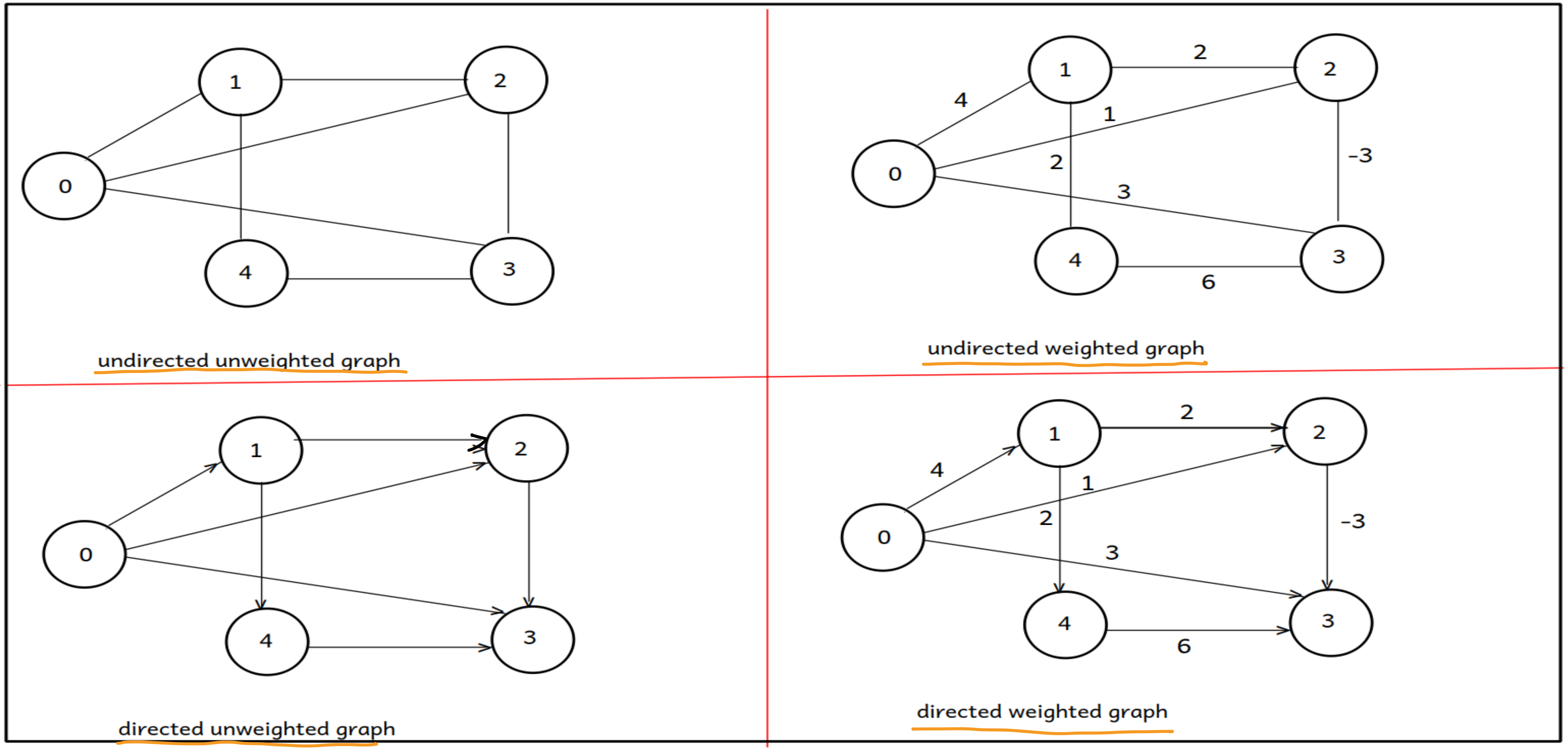




(u, v) != (v, u)

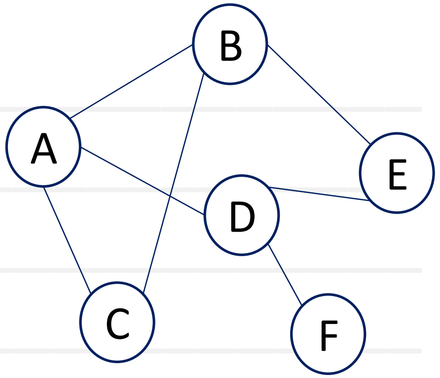
Graph : Types

- **Weighted Graph**
  - A graph in which edge is associated with a number (ie weight)

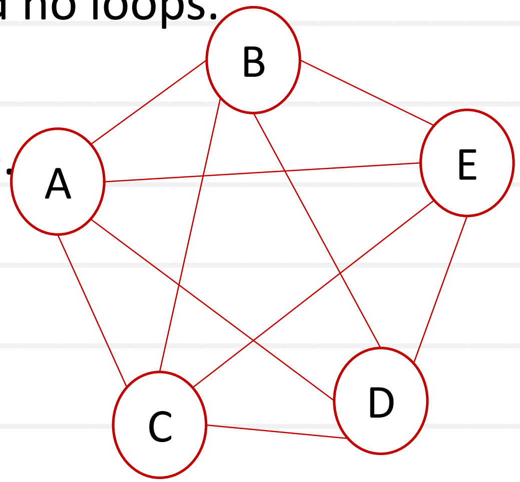


Graph : Types

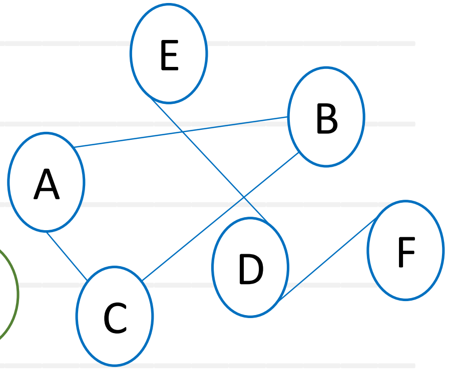
- **Simple Graph**
  - Graph not having multiple edges between adjacent nodes and no loops.



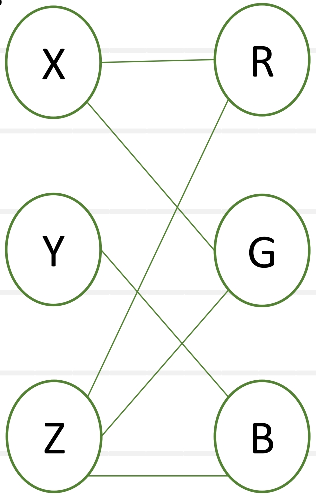
- **Complete Graph**
  - Simple graph in which node is adjacent with every other node.
  - Un-Directed graph: Number of Edges = n (n -1) / 2
    where, n – number of vertices
  - Directed graph: Number of edges = n (n-1)



- **Connected Graph**
  - Simple graph in which there is some path exist between any two vertices.
  - Can traverse the entire graph starting from any vertex.

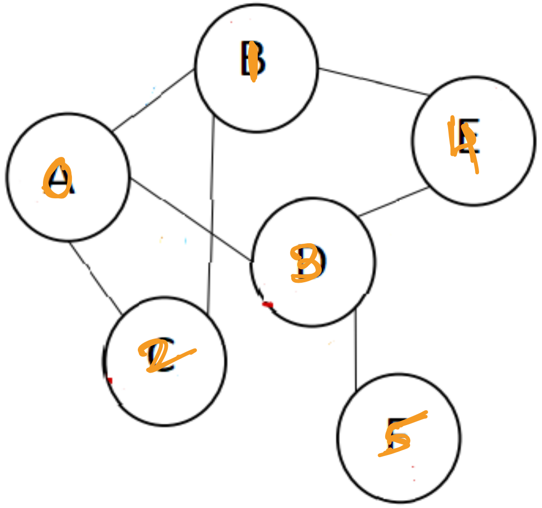


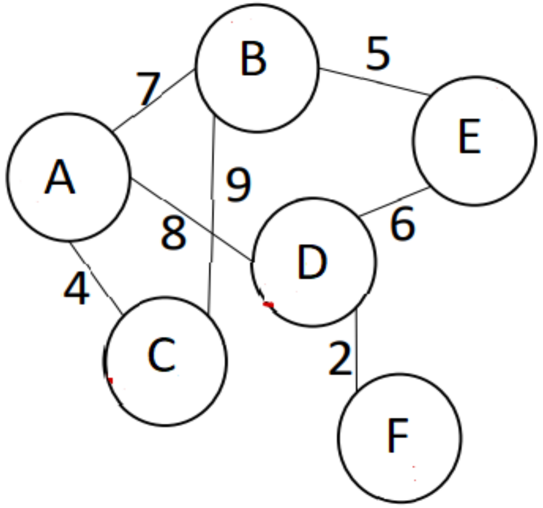
- **Bi-partite graph**
  - Vertices can be divided in two disjoint sets.
  - Vertices in first set are connected to vertices in second set.
  - Vertices in a set are not directly connected to each other.



Graph Implementation – Adjacency Matrix

- If graph have V vertices, a V x V matrix can be formed to store edges of the graph.
- Each matrix element represent presence or absence of the edge between vertices.
- For non-weighted graph, 1 indicate edge and 0 indicate no edge.
- For weighted graph, weight value indicate the edge and infinity sign ∞ represent no edge.
- For un-directed graph, adjacency matrix is always symmetric across the diagonal.
- Space complexity of this implementation is O(V^2).



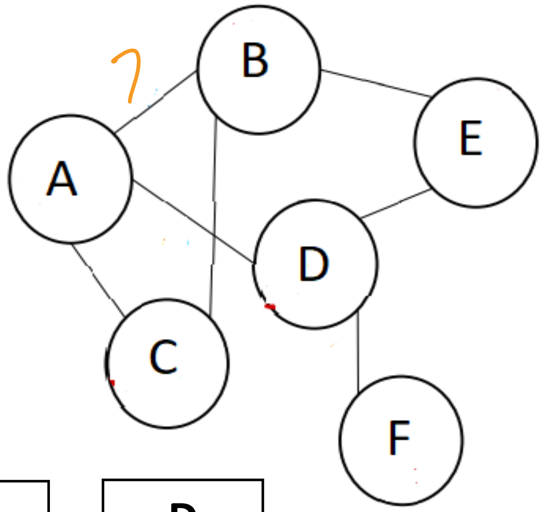


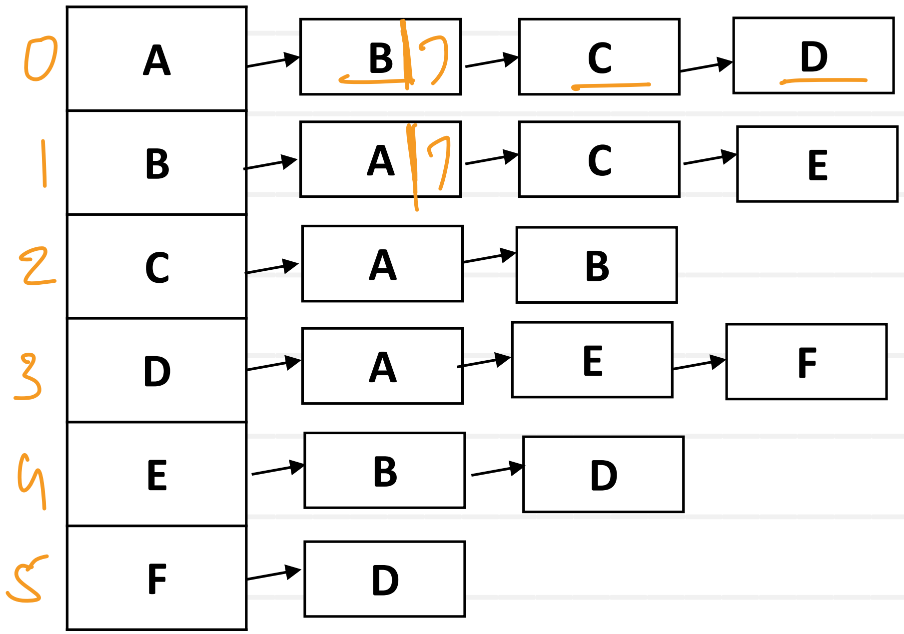
	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	1	0	1	0
C	1	1	0	0	0	0
D	1	0	0	0	1	1
E	0	1	0	1	0	0
F	0	0	0	1	0	0

	A	B	C	D	E	F
A	∞	7	4	8	∞	∞
B	7	∞	9	∞	5	∞
C	4	9	∞	∞	∞	∞
D	8	∞	∞	∞	6	2
E	∞	5	∞	6	∞	∞
F	∞	∞	∞	2	∞	∞

Graph Implementation – Adjacency List

- Each vertex holds list of its adjacent vertices.
- For non-weighted graphs only, neighbor vertices are stored.
- For weighted graph, neighbor vertices and weights of connecting edges are stored.
- Space complexity of this implementation is O(V+E).
- If graph is sparse graph (with fewer number of edges), this implementation is more efficient (as compared to adjacency matrix method).





Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com