Data Structure & Algorithms

Nilesh Ghule

Quick Sort

4 7 9 2 8 1 6 3 5

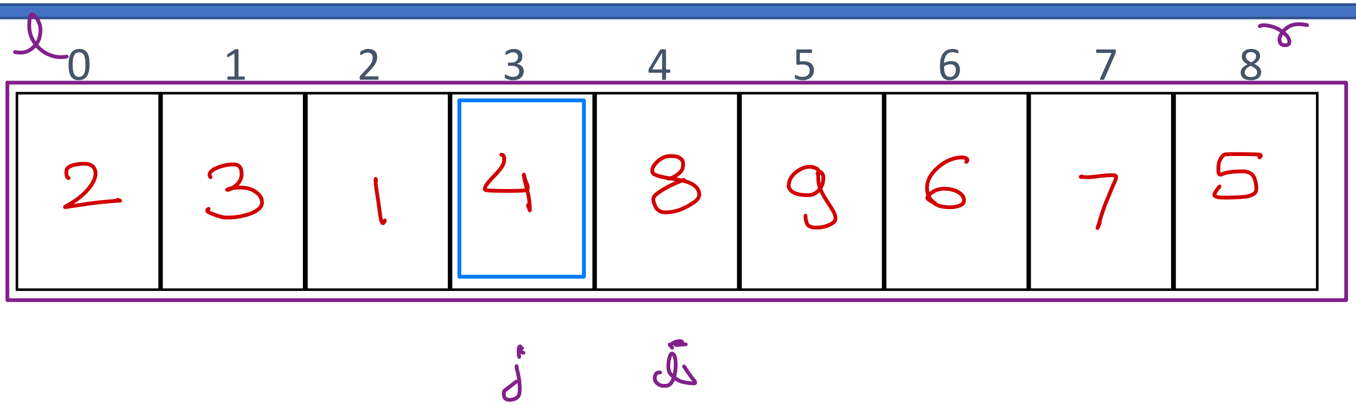
```
if(l >= r)
    return;
// a[l] -> pivot
i = l;  j = r;
while (i < j) {
    while (i<size && a[i] <= a[l])
        i++;
    while (a[j] > a[l])
        j--;
    if (i<j)
        swap(a[i], a[j]);
}
swap(a[j], a[l]);
quicksort(a, l, j-1);
quicksort(a, j+1, r);
```

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

l 0	1	2	r 3	4
5	2	4	1	
			j	i

l 0	1	2	3 r	4
1	2	4	3	
j	i			

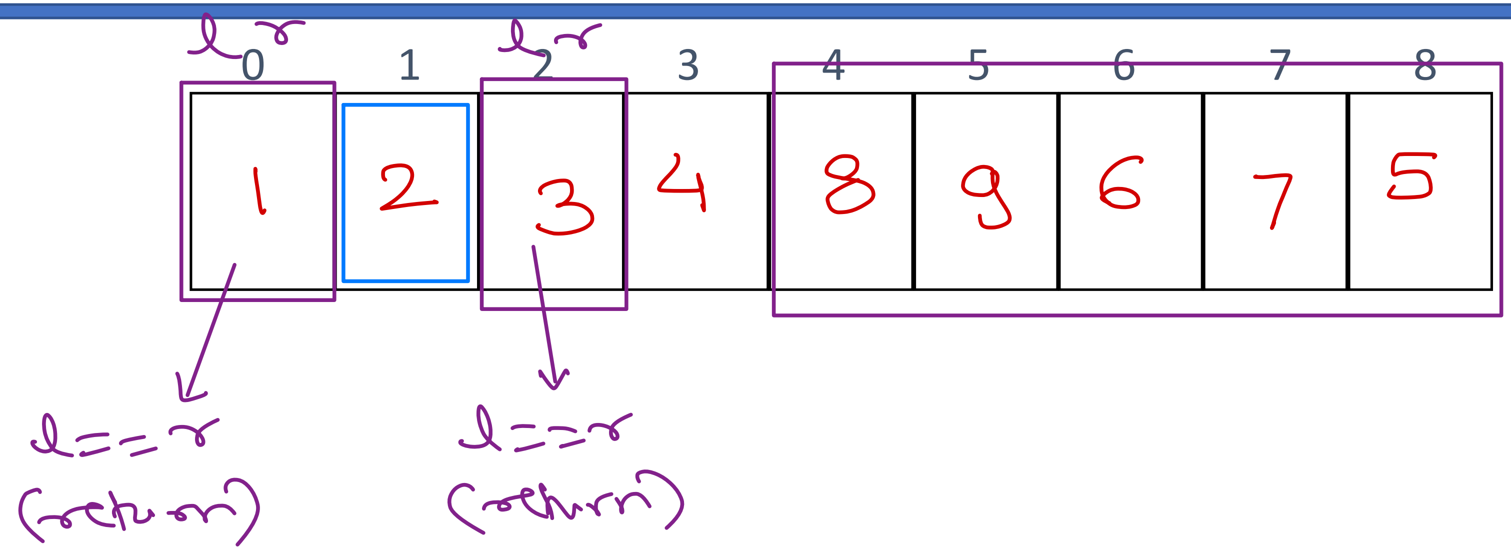
Quick Sort



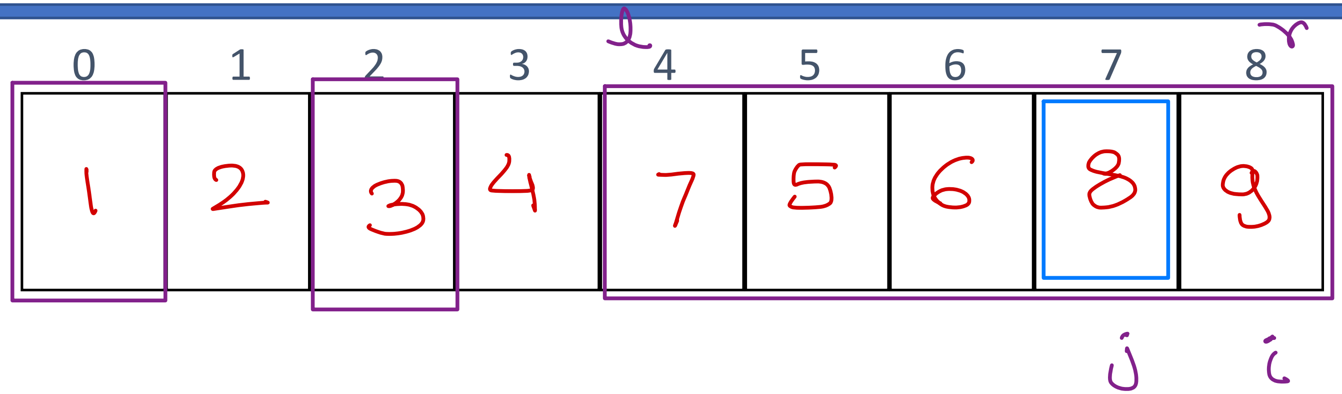
Quick Sort

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

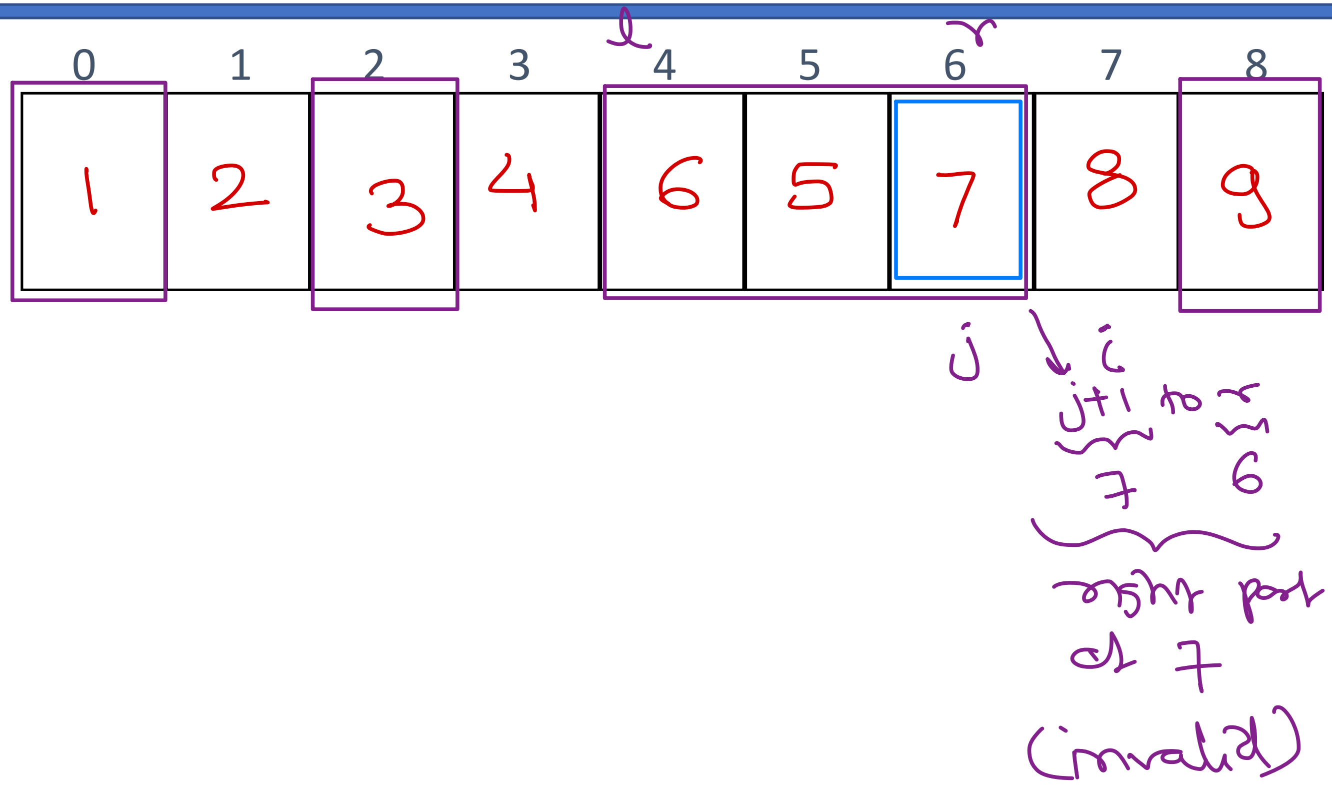
Quick Sort



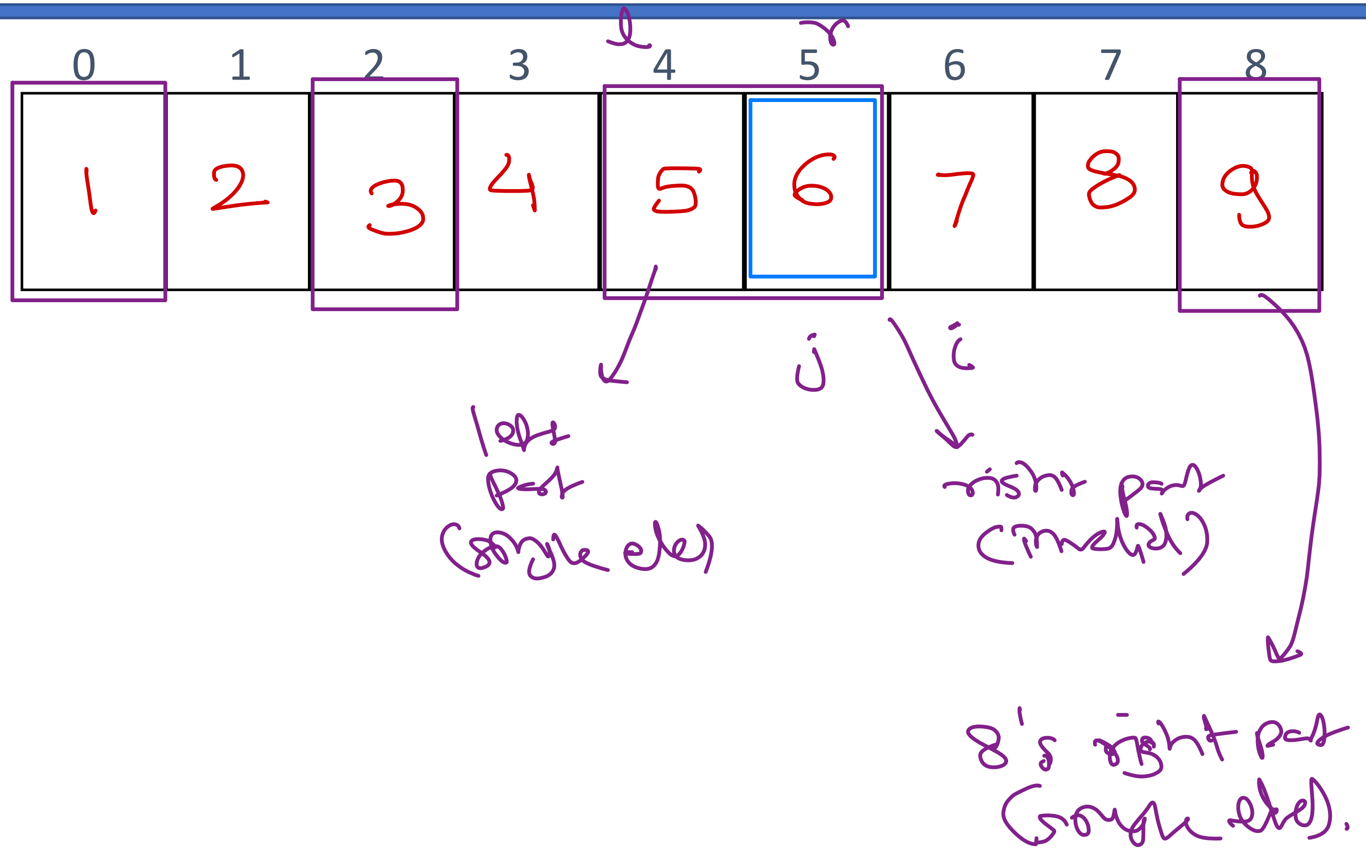
Quick Sort



Quick Sort

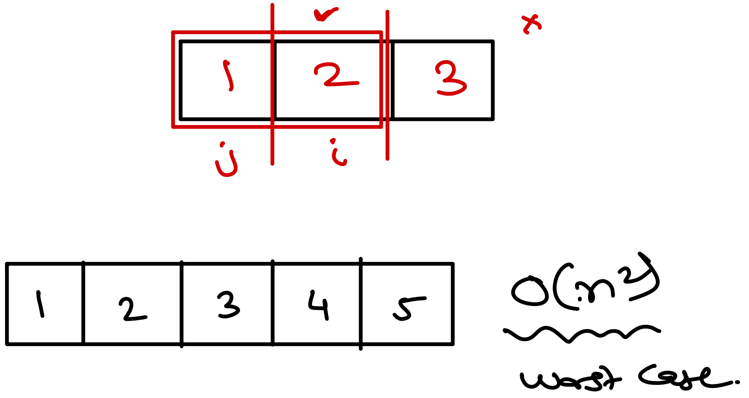


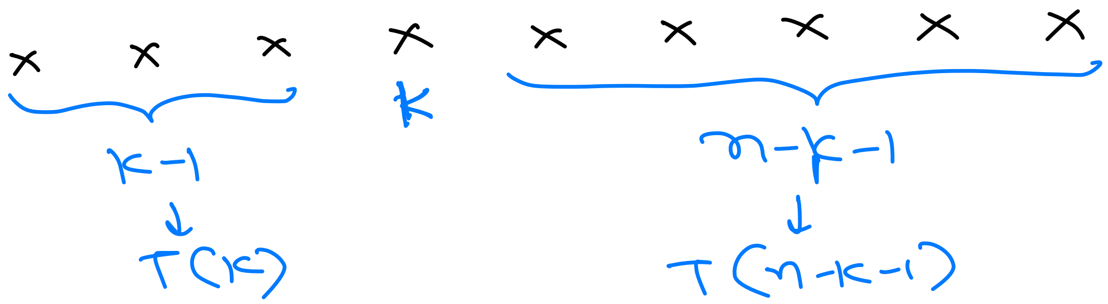
Quick Sort


0 1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
l
r
j
i
left part
(single ele)
right part
(invalid)
8's right part
(single ele).

Quick Sort – Time complexity

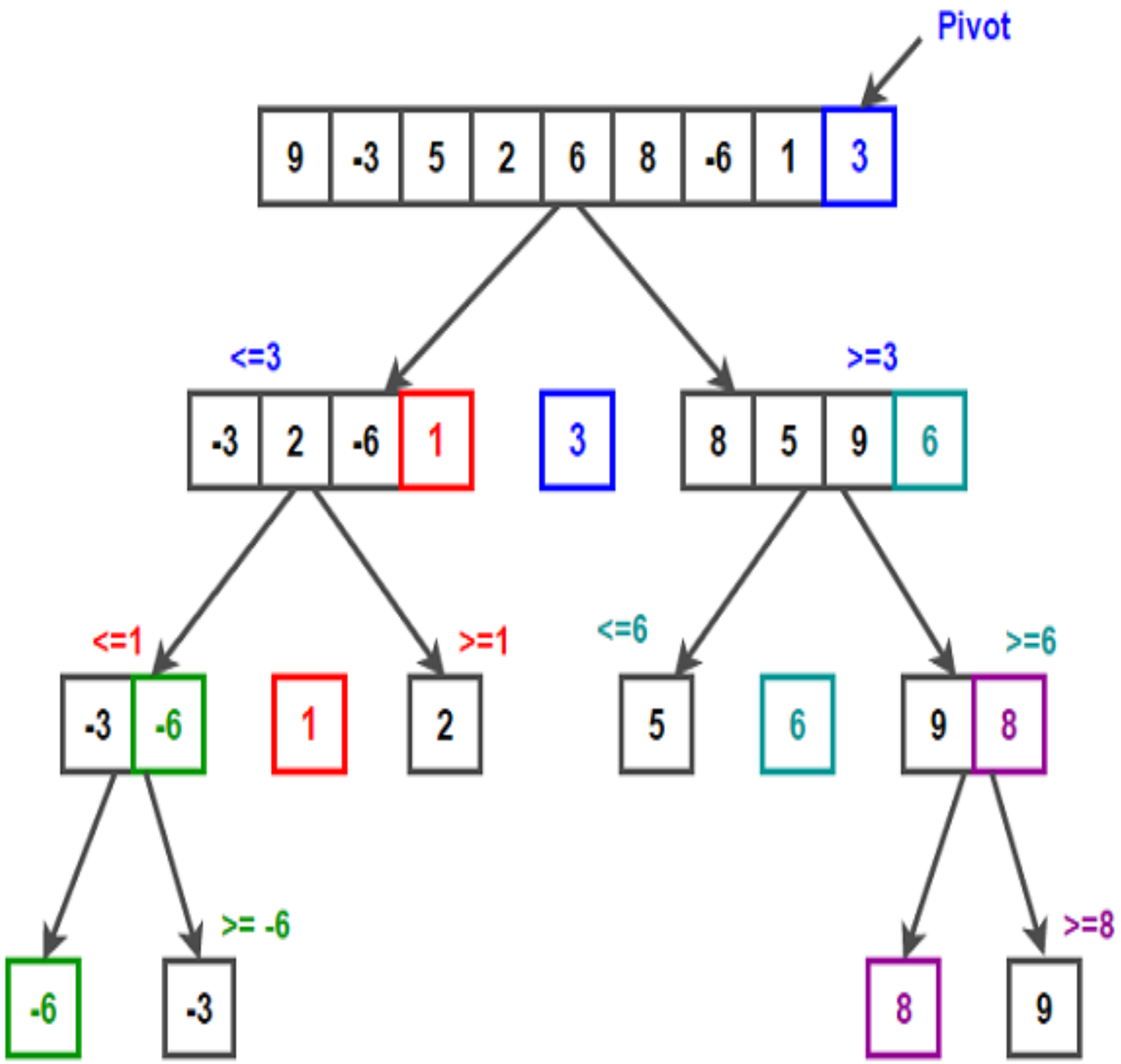
- Quick sort pivot element can be
  - First element or Last element
  - Random element
  - Median of the array ← though seems efficient, calc median is time taking.
- Quick sort time
  - Time to partition as per pivot – T(n)
  - Time to sort left partition – T(k)
  - Time to sort ~~left~~ right partition – T(n-k-1)
- Worst case
  - $T(n) = T(0) + T(n-1) + O(n) \Rightarrow O(n^2)$
- Best case
  - $T(n) = T(n/2) + T(n/2) + O(n) \Rightarrow O(n \log n)$
- Average case
  - $T(n) = T(n/9) + T(9n/10) + O(n) \Rightarrow O(n \log n)$





Recursion – QuickSort

- Algorithm
  1. If single element in partition, return.
  2. Last element as pivot.
  3. From left find element greater than pivot (x^{th} ele).
  4. From right find element less than pivot (y^{th} ele).
  5. Swap x^{th} ele with y^{th} ele.
  6. Repeat 2 to 4 until x < y.
  7. Swap y^{th} ele with pivot.
  8. Apply QuickSort to left partition (left to y-1).
  9. Apply QuickSort to right partition (y+1 to right).
- QS(arr, 0, 8)
  - QS(arr, 0, 3)
    - QS(arr, 0, 1)
      - QS(arr, 0, 0)
    - QS(arr, 3, 3)
  - QS(arr, 5, 8)
    - QS(arr, 5, 5)
    - QS(arr, 7, 8)
      - QS(arr, 9, 9)



Merge Sort

merge two sorted Partitions into a single array

4 7 9 2 8 | 1 6 3 5

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

i j

temp

1	2	3	4	5	6	7	8	9

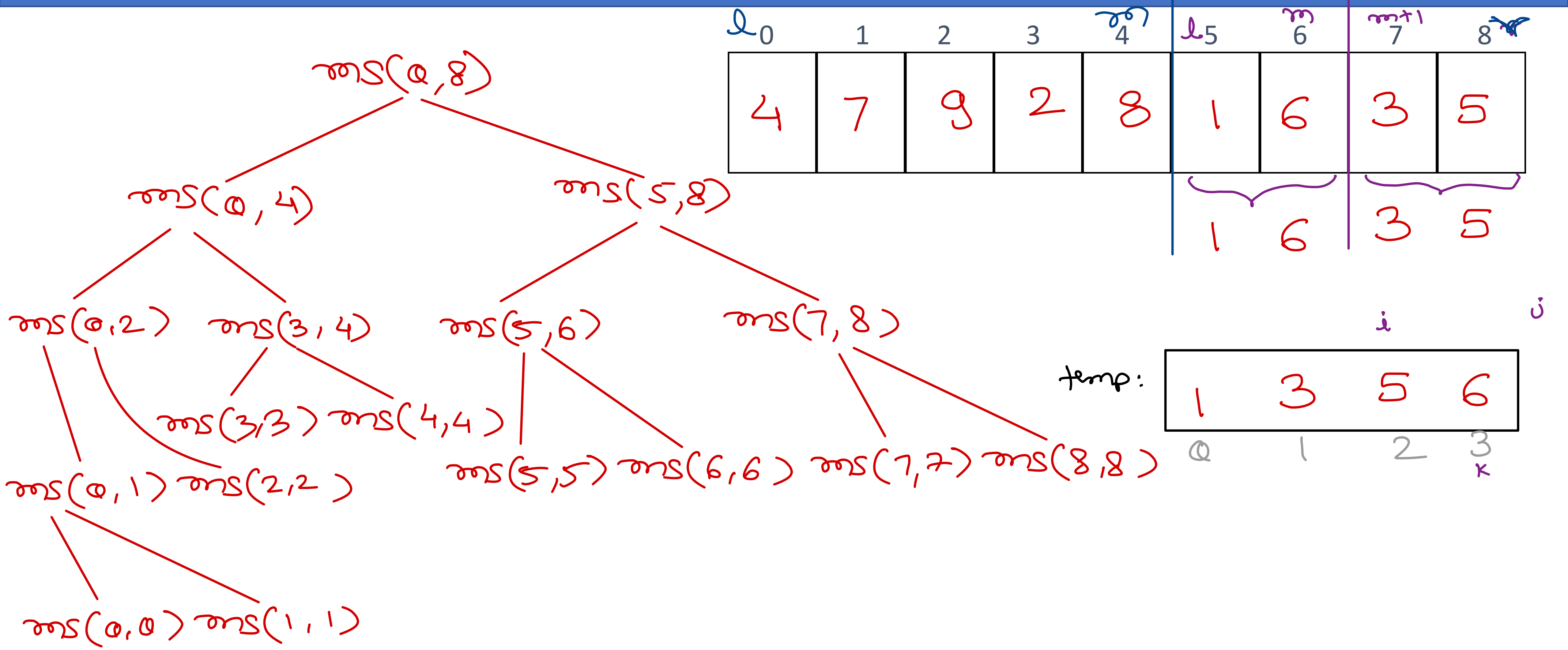
k

```
if(l >= r)
    return;
m = (l+r)/2;
mergesort(arr, l, m);
mergesort(arr, m+1, r);
i=left; j=m+1; k=0;
temp = new int[r-l+1];
while(i<=m && j<=r) {
    if(a[i] < a[j]) {
        temp[k] = a[i];
        i++; k++;
    } else {
        temp[k] = a[j];
        j++; k++;
    }
}

while(i<=m) {
    temp[k] = a[i];
    i++; k++;
}
while(j<=r) {
    temp[k] = a[j];
    j++; k++;
}

for(k=0; k<temp.length; k++)
    a[l+k] = temp[k];
```

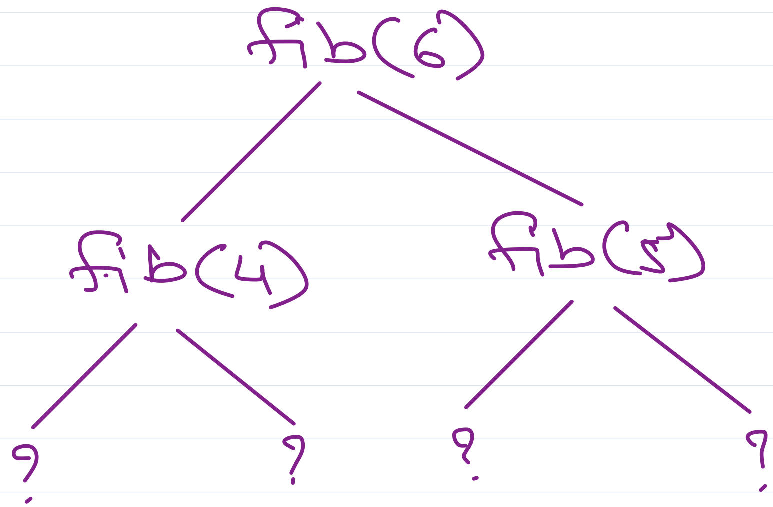
Merge Sort - Recursion Tree



```
int fib(int n) {
    if(n==1 || n==2)
        return 1;
    int r = fib(n-1) + fib(n-2);
    return r;
}
```

1 1 2 3 5 8 13 21 ...

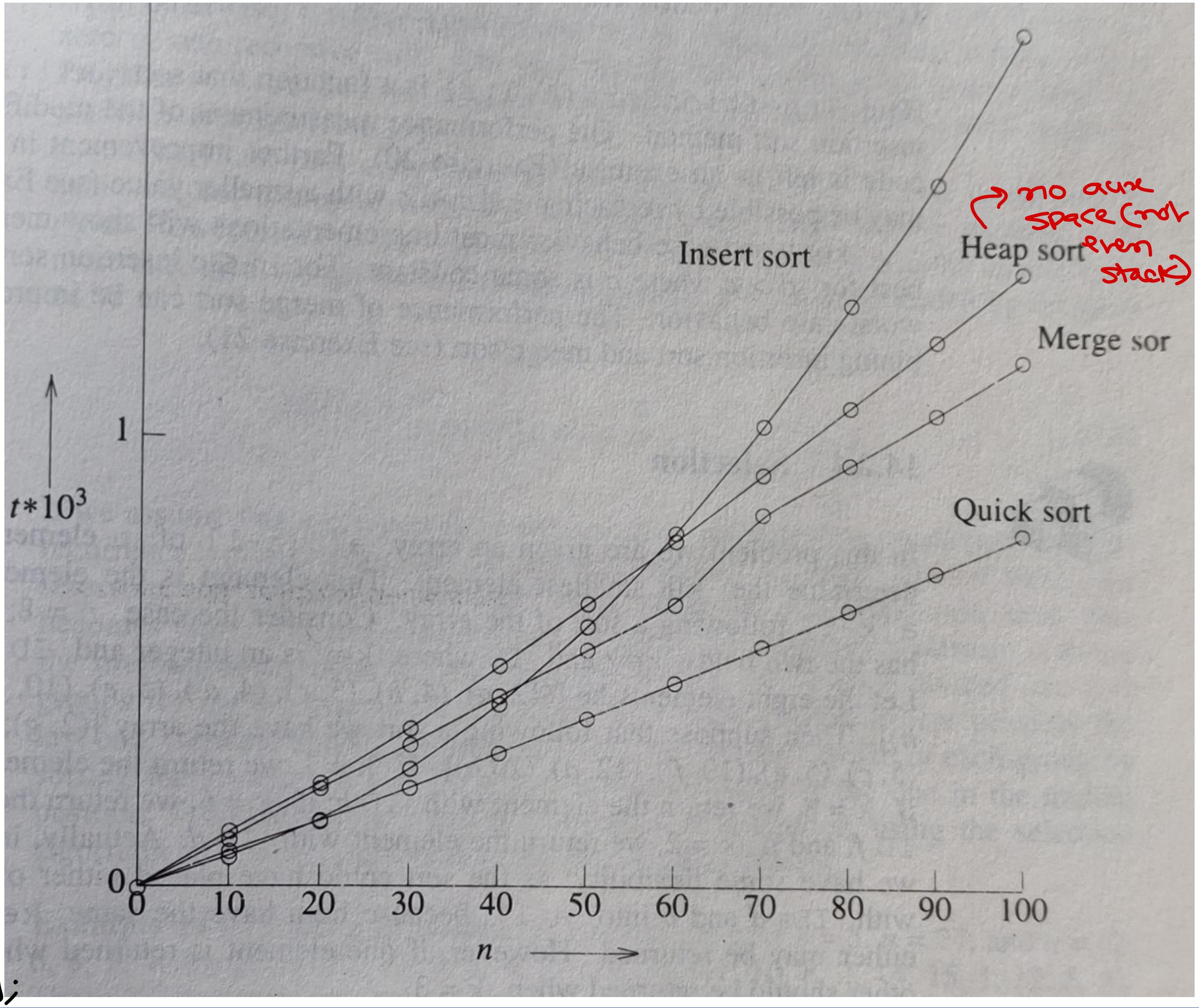
1 2 3 4 5 6



Sorting Algorithm Comparison

- Selection sort algorithm is too simple, but performs poor and no optimization possible.
- Bubble sort can be improved to reduce number of iterations.
- Insertion sort performs well if number of elements are too less. Good if adding elements and resorting.
- Quick sort is stable if number of elements increase. However worst case performance is poor. $O(n^2)$
- Merge sort also perform good, but need extra auxiliary space.

temp = new int (r - l + 1);

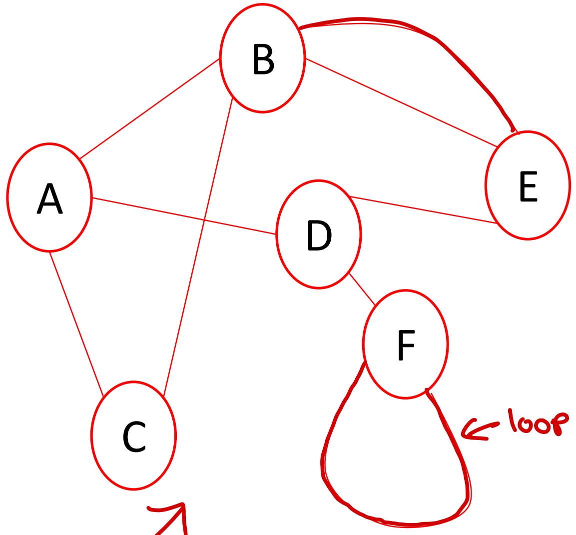


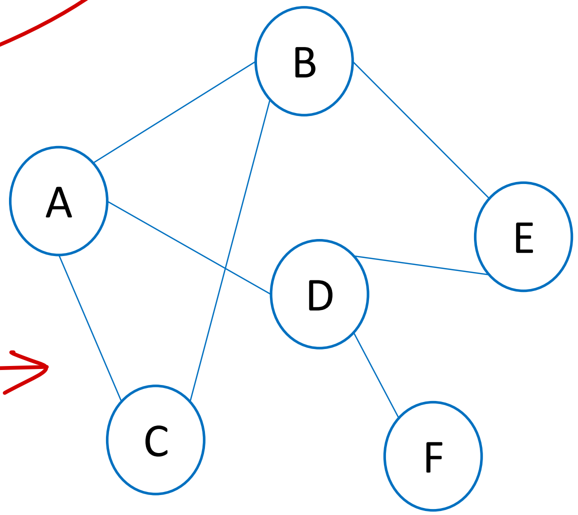
Graph

- Graph is a non-linear data structure.
- Graph is defined as set of vertices and edges. Vertices (also called as nodes) hold data, while edges connect vertices and represent relations between them. $V = \{A, B, C, D, E, F\}$
  - G = { V, E }

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

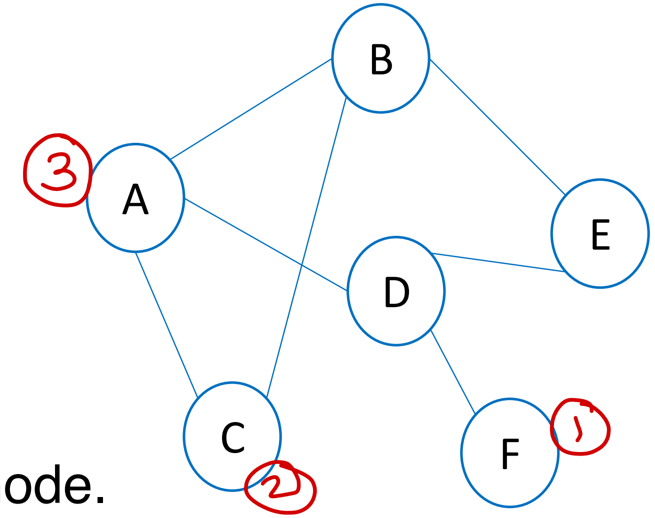
- Vertices hold the data and Edges represents relation between vertices.
- When there is an edge from vertex P to vertex Q, P is said to be adjacent to Q.
- Multi-graph
  - Contains multiple edges in adjacent vertices or loops (edge connecting a vertex to it-self).
- Simple graph
  - Doesn't contain multiple edges in adjacent vertices or loops.





Graph

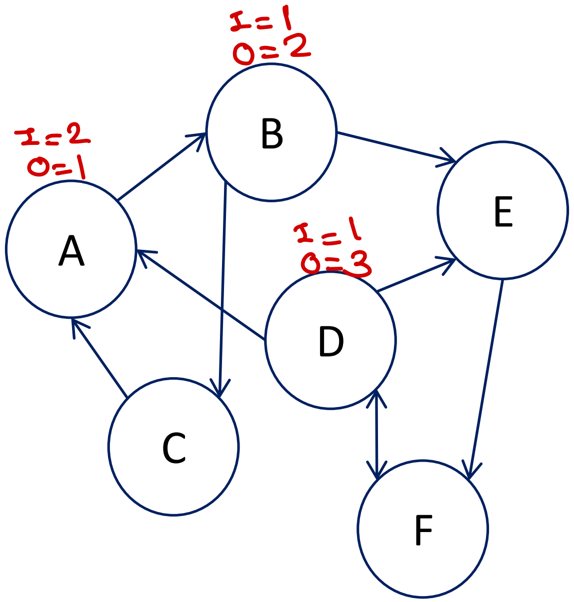
- Graph edges may or may not have directions.



- Undirected Graph: G = { V, E }
  - V = { A, B, C, D, E, F}
  - E = { (A,B), (A,C), (A,D), (B,C), (B,E), (D,E), (D,F) }
  - If P is adjacent to Q, then Q is also adjacent to P.
  - Degree of node: Number of nodes adjacent to the node.
  - Degree of graph: Maximum degree of any node in graph.

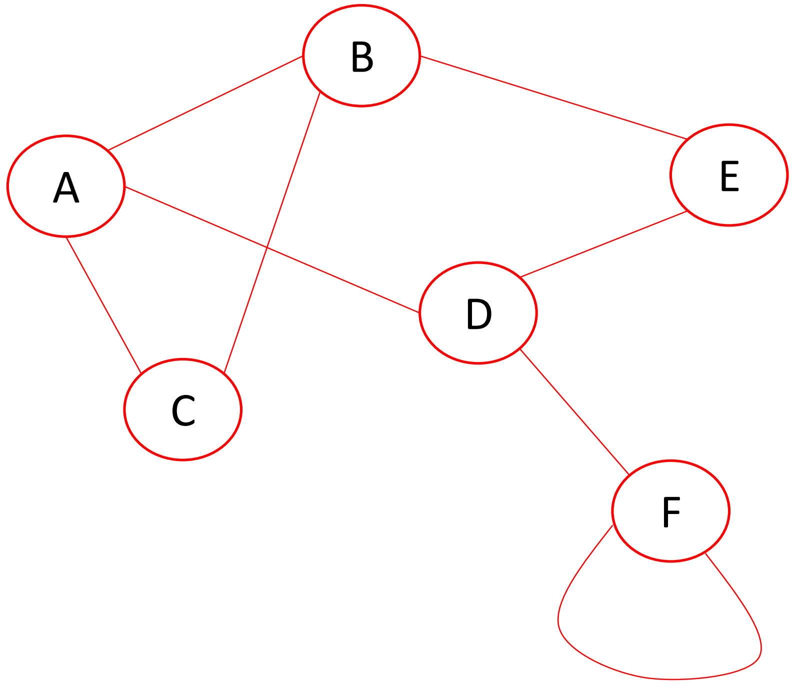
3

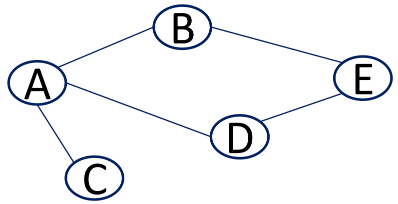
- Directed Graph: G = { V, E } Di-graph
  - V = { A, B, C, D, E, F}
  - E = {<A,B>, <B,C>, <B,E>, <C,A>, <D,A>, <D,E>, <D,F>, <E,F>, <F,D>}
  - If P is adjacent to Q, then Q is may or may not be adjacent to P.
  - Out-degree: Number of edges originated from the node
  - In-degree: Number of edges terminated on the node

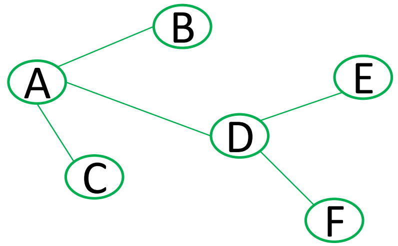


Graph

- Path: Set of edges between two vertices. There can be multiple paths between two vertices.
  - A – D – E
  - A – B – E
  - A – C – B – E
- Cycle: Path whose start and end vertex is same.
  - A – B – C – A
  - A – B – E – D – A
- Loop: Edge connecting vertex to itself. It is smallest cycle.
  - F – F
- Sub-Graph: A graph having few vertices and few edges in the given graph, is said to be sub-graph of given graph.



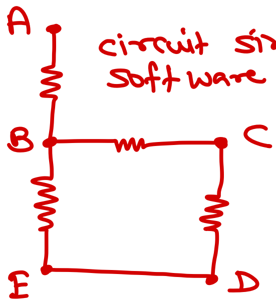




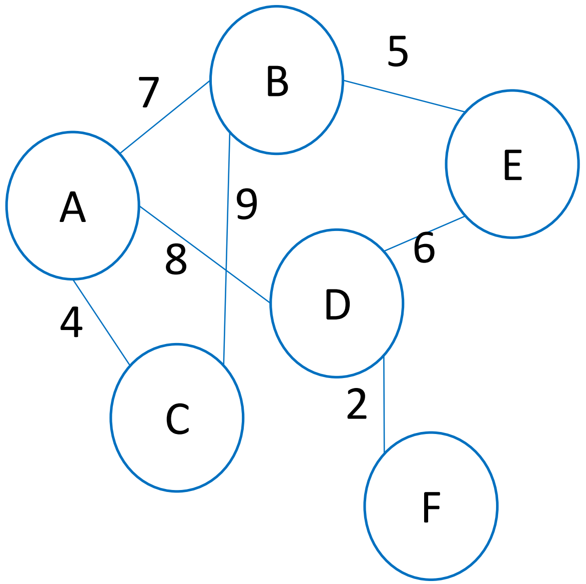
Graph

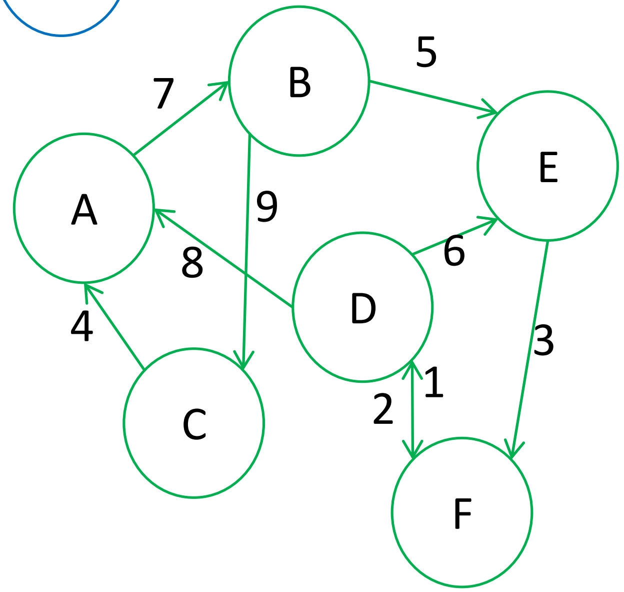
- Weighted graph
  - Graph edges have weight associated with them.
  - Weight represent some value e.g. distance, resistance.
- Directed Weighted graph (Network)
  - Graph edges have directions as well as weights.
- Applications of graph
  - Electronic circuits
  - Social media
  - Communication network
  - Road network
  - Flight/Train/Bus services
  - Bio-logical & Chemical experiments
  - Deep learning (Neural network, Tensor flow)
  - Graph databases (Neo4j)

circuit simulation software



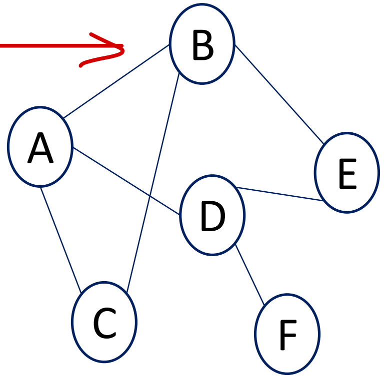
telecom / LAN



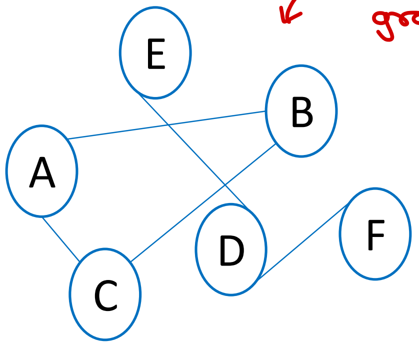


Graph

- Connected graph
  - From each vertex some path exists for every other vertex.
  - Can traverse the entire graph starting from any vertex.

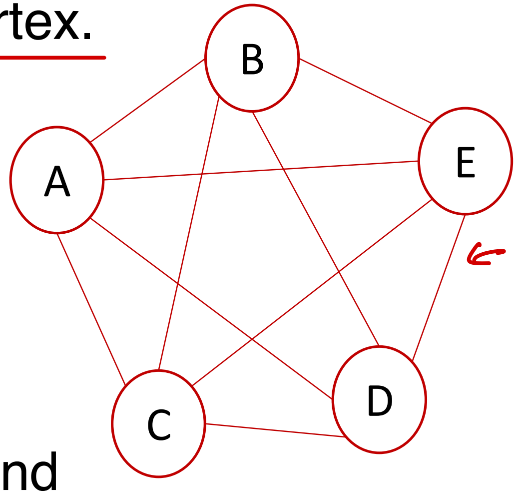


disconnected graph



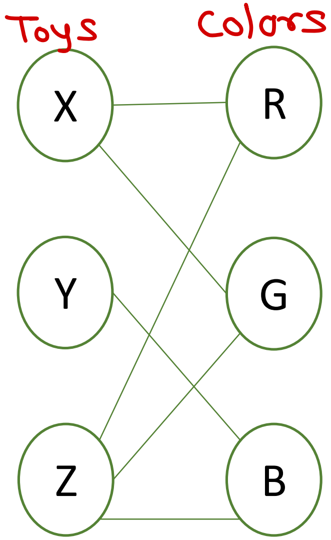
- Complete graph

  direct edge
  - Each vertex of a graph is adjacent to every other vertex.
  - Un-directed graph: Number of edges = n (n-1) / 2
  - Directed graph: Number of edges = n (n-1)



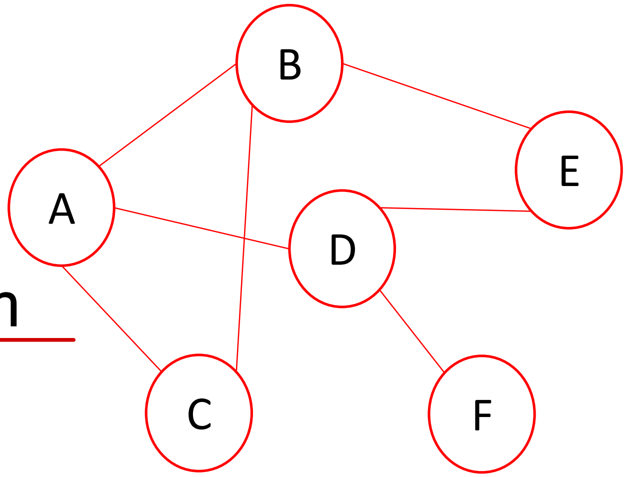
undirected Complete graph

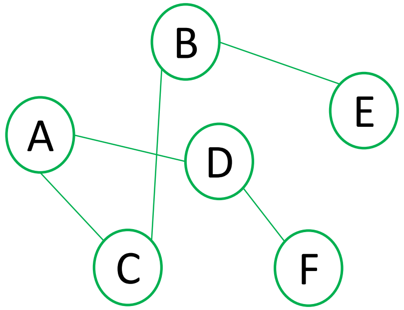
- Bi-partite graph
  - Vertices can be divided in two disjoint sets.
  - Vertices in first set are connected to vertices in second set.
  - Vertices in a set are not directly connected to each other.

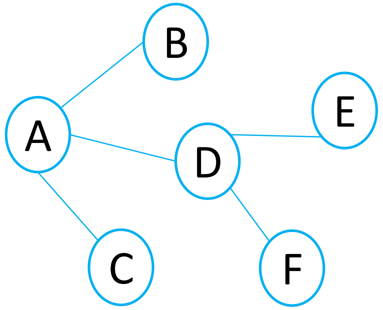


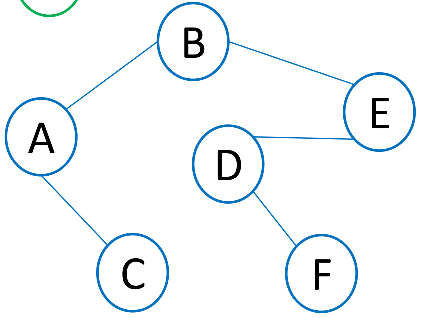
Spanning Tree

- Tree is a graph without cycles.
- Spanning tree is connected sub-graph of the given graph that contains all the vertices and sub-set of edges (V-1).
- Spanning tree can be created by removing few edges from the graph which are causing cycles to form.
- One graph can have multiple different spanning trees.
- In weighted graph, spanning tree can be made who has minimum weight (sum of weights of edges). Such spanning tree is called as Minimum Spanning Tree. (MST)
- Spanning tree can be made by various algorithms.
  - BFS Spanning tree
  - DFS Spanning tree
  - Prim's MST
  - Kruskal's MST



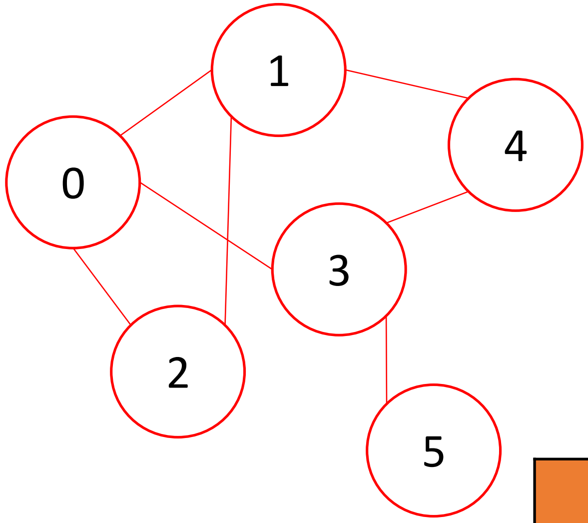






Graph Implementation – Adjacency Matrix

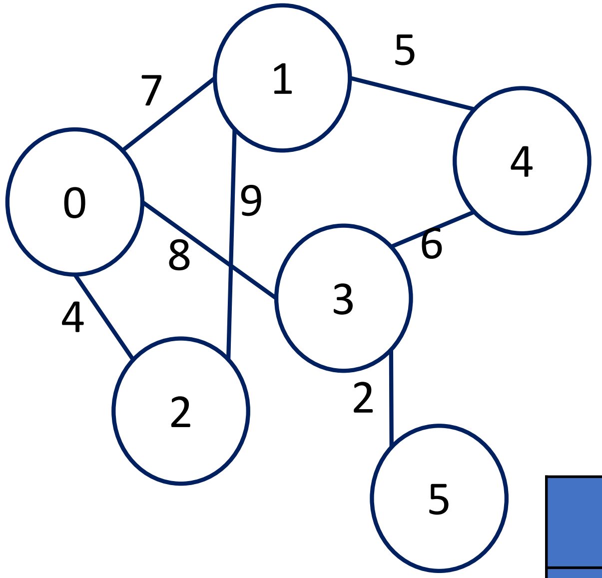
- If graph have V vertices, a V x V matrix can be formed to store edges of the graph.
- Each matrix element represent presence or absence of the edge between vertices.
- For *non-weighted graph*, 1 indicate edge and 0 indicate no edge.
- For un-directed graph, adjacency matrix is always symmetric across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



	0	1	2	3	4	5
0	0	1	1	1	0	0
1	1	0	1	0	1	0
2	1	1	0	0	0	0
3	1	0	0	0	1	1
4	0	1	0	1	0	0
5	0	0	0	1	0	0

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 *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 $O(V^2)$.



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

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>