Graph Data Structure.

$G = (V, E)$

Types:

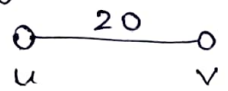
i) Null graph: No edges, isolated vertices.

ii) Trivial graph: single vertex

iii) Connected & Disconnected graph:
   ↳ we can have path blw any two nodes directly or indirectly.

iv) Directed & Undirected: edges have direc^n then directed.

v) Weighted & Unweighted: Assign weights to edges.



vi) Complete graph: direct path blw any two nodes

vii) Regular graph: all nodes have same degree.

viii) Bipartite graph: Divide vertices into two sets V_1 V_2 such that $V_1 \cap V_2 = \emptyset$ and there are no edges within each part.

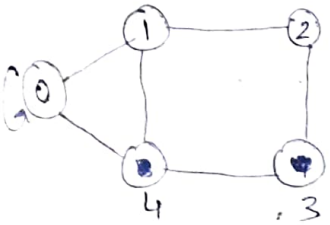
ix) Cyclic graph: Atleast one cycle in it.

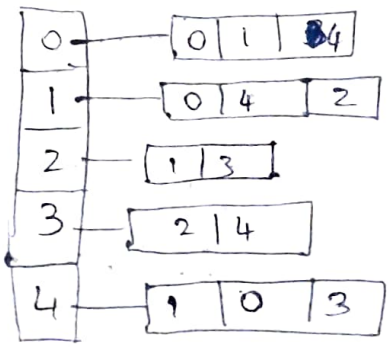
x) Directed Acyclic Graph (DAG)

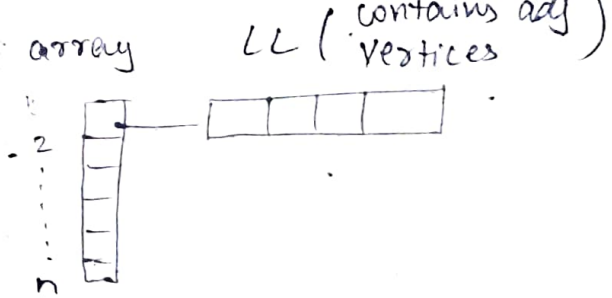
Representation of Graph.

→ array of linked list

(i) Adjacency List

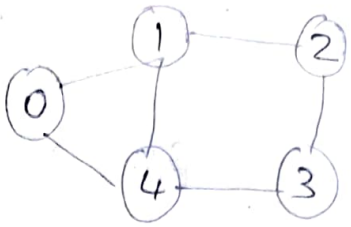






2) Adjacency Matrix

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



if there is a edge b/w vertices ~~(u,v) the~~ u a
then (u,v) = (v,u) = 1

if it was a weighted graph then instead of 0,1 we store 0, w w = weight.

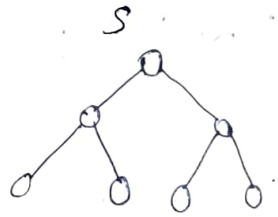
if list is very long, then traversal through the list in Adjacency List becomes costly. Otherwise we prefer List.
Matrix takes up more space.

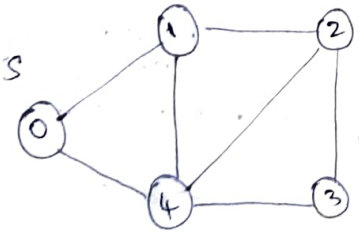
Graph Operations

1) Traversal
2) Spanning Tree (MST)
3) Shortest Path

TRAVERSAL

1) BFS (Breadth first search)
2) DFS (Depth first search)





· BFS uses queue DS and visited array

Q | 0 | step-1 itr-1

V.A | 0 |

step:
→ pop front of queue
→ Then visit its neighbours
→ Then enqueue the neigh in queue

neigh of 0 = 1, 4

Q | 1 | 4 | itr-2

VA | 0 | 1 | 4 |

pop 1 neigh(1) = 2 → add to queue

Q [4|2|] VA [0|1|4|2|] itr-3

pop 4 neigh(4) = 3

Q [2|3|] VA [0|1|4|2|3] itr-4

pop 2 neigh(2) = null

Q [3|] VA [0|1|4|2|3] itr-5

pop 3 neigh(3) = null

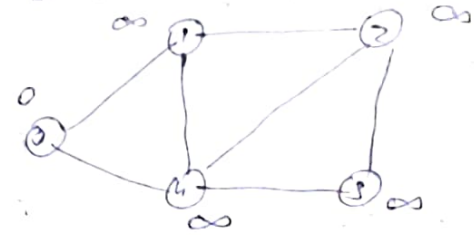
Q [ ] VA [0|1|4|2|3] itr-6 empty queue

Alternate

→ Assign source vertex as dist=0 & others as dist = ∞

→ Assign unvisited vertex colour = white

→ first visit grey then black.



Q [0|] itr-1

pop-0 neigh(0) = 1, 4

v.d = u.d + 1 update dist

going from coming : itr-2

Q [1|4|]

pop-1 neigh(1) = 2

dist(2) = dist(1)+1

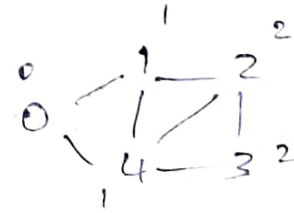
= 1+1 = 2

pop-4

Q [4|2|]

neigh(4) = 3

dist(3) = dist(4)+1 = 1+1 = 2



Q [2|3|] Now all vertices visited. Stop.

Now we get the shortest dist. of each node from 0.

DSA

BFS(G,S)

```
for each vertex u in G.V - {S}
        u.colour = white
        u.d = ∞
        u.π = nil
```

initialisation
S = starting vertex.
white = unvisited

```
S.colour = gray
S.d = 0
S.π = nil
enqueue (Q,S)
```

initialising S and setting up queue.

```
while Q is not empty
    u = dequeue(Q)
    for each v in G.adj[u]
        if v.colour = white
            v.colour = gray
            v.d = u.d + 1
            v.π = u
            Enqueue(Q,v)
    u.colour = black
```

Algorithm
π indicates parent.
it takes O(V+E)
distances measured by BFS are the shortest path dist.

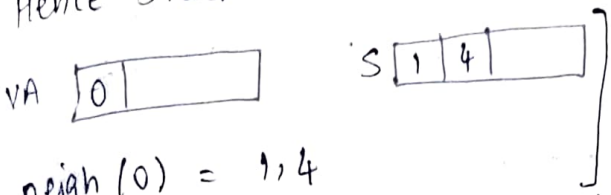
- → web crawlers
- → topological sort
- → GPS navigation
- → shortest dist

} application BFS

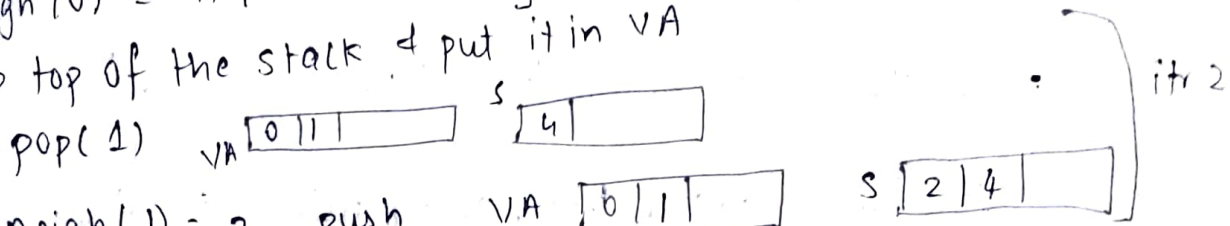
Black colour says we are done with that node.

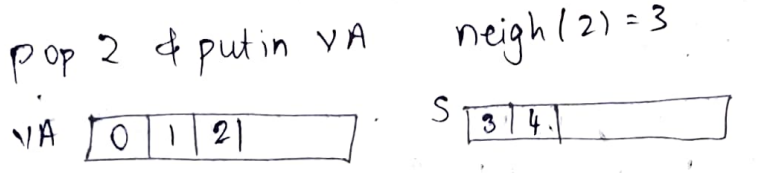
will BFS work on a disconnected graph? NO
modify the algo to restart at disconnected nodes

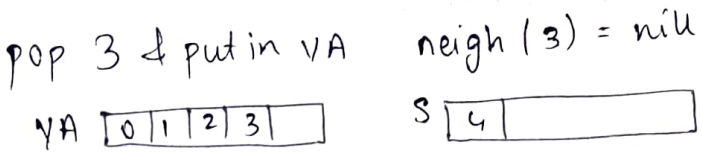
DFS (Depth foot Search)
explore neighbours of neighbours
before exploring siblings
Hence stack is used along with VA

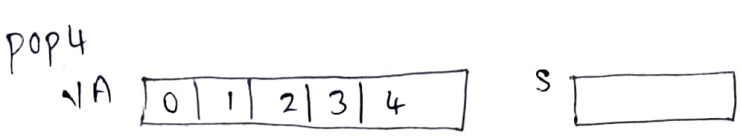


neigh(0) = 1, 4
pop top of the stack & put it in VA





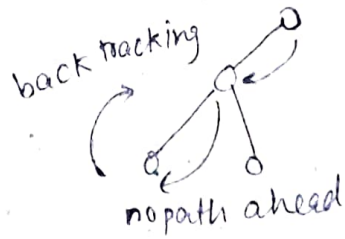




DFS & BFS can give multiple sol^{n}s and both have to be modified for disconnected graph.

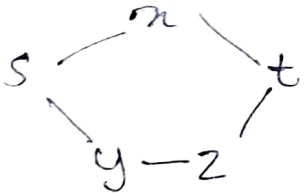
DFS O(V+E)

Applications.
- → Detecting cycles in graph
- → finding a path
- → topological sort
- → check if graph is bipartite
- → backtracking & web crawlers



if vertexs are many, DFS is stuck
if branching is high, BFS is stuck.

BFS ~~garuntees~~ garentees a soln but DFS doesn't.



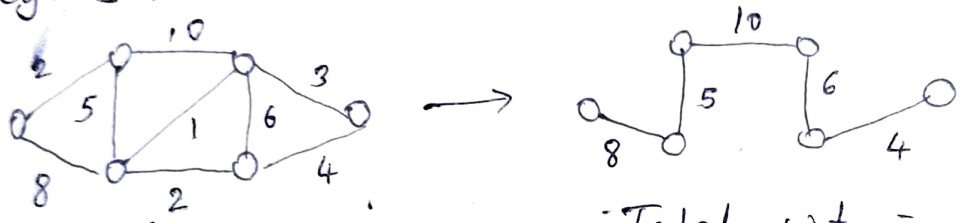
BFS $s \to x \to t$

DFS $s \to y \to z \to t$

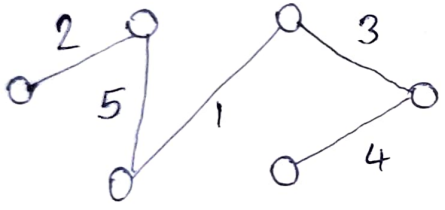
(maynot give shortest path)

Spanning Tree

A tree which contains all the vertices of the original graph but has no cycles in it.



Total wt = 33 (not the least)



Total wt = 15 (minimum)

↳ Minimum Spanning Tree (MST)

↳ may not be unique

Props:

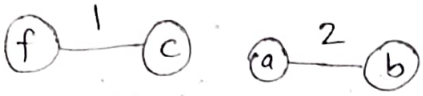
1) edges = vertices − 1 $\quad E = V - 1$
2) Spanning trees are acyclic
3) Connected and contains all vertices.

KRUSKAL's ALGO (greedy approach)

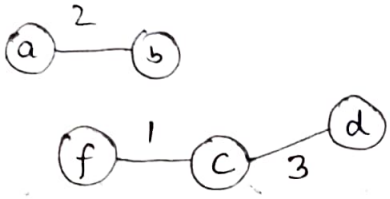
main idea: 1) Sort all edges in increasing order
2) Pick the lowest cost edge and add it to the MST such that it does not create a cycle.

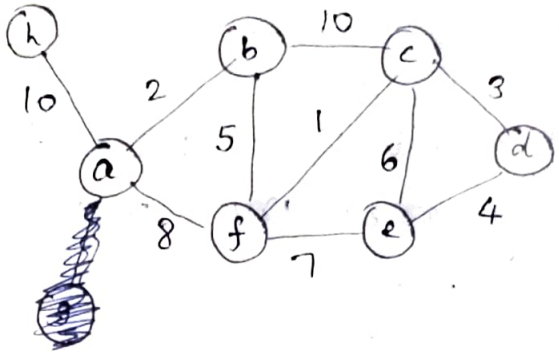
1)

2)



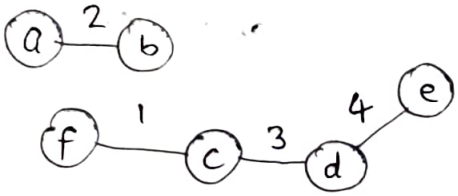
3)



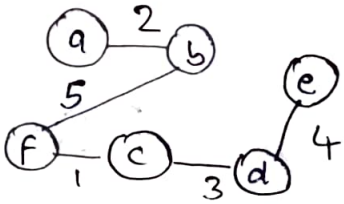


wt	(u, v)
1	f, c
2	a, b
3	c, d
4	e, d
5	b, f
6	c, e
7	f, e
8	a, f
10	b, c
10	a, h

4)

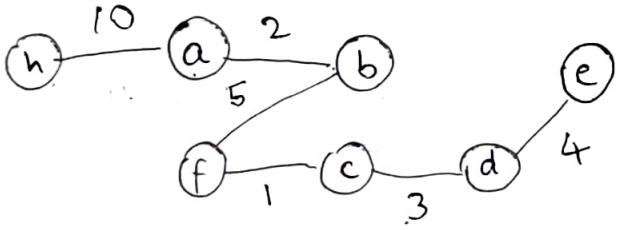


5)



6) discard (c, e) (f, e) (a, f) (b, c) as they form cycle.

7)



Total = 25
(MST)

Sorting: $E \log E$
union find algo: $\log V$
for E edges UFA: $E \log V$

$E \log V$ dominates $E \log E$

$\therefore$ time complexity $O(E \log V)$

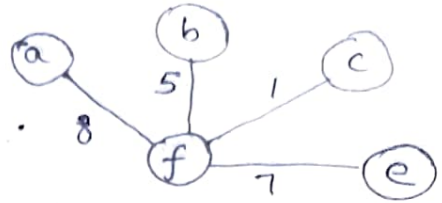
PRIM's ALGO (greedy Algo)

pick any vertice at the start. Now it is a part of MST

Pick f

MST (f)

connections



choose min ie (c)

MST (f) —1— (c)

choose c, d ie 3

MST (f) —1— (c) —3— (d)

choose d, e. ie 4

MST (f) —1— (c) —3— (d) —4— (e)

Choose f, b ie 5

MST (b) —5— (f) —1— (c) —3— (d) —4— (e)

choose a, b ie 2

MST (a) —2— (b) —5— (f) —1— (c) —3— (d) —4— (e)

Now a, h remaining

MST (h) —10— (a) —2— (b) —5— (f) —1— (c) —3— (d) —4— (e)

Total = 25

main idea: Always add the edge which will add a new vertice to the MST and has the least weight. At all times the tree is connected but not cyclic

Time complexity: $\Theta(V)\, T_{\text{Extraction min}} + \Theta(E)\, T_{\text{Decrease-min}}$.

Disjoint set DS

1) Make (x) ↓ creates a set

2) Find (x) ↓ given sets, finds to which sets x belongs to

3) Union (x, y) ↓ creates union operation two elements x and y be to different sets.

Union-find Algo to Detect Cycle in Undirected graph

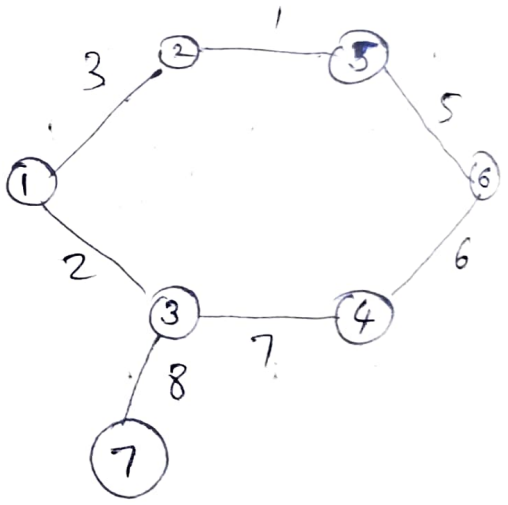
→ Take each vertex in individual set

$S_1\{1\}$ $S_2\{2\}$ .... $S_7\{7\}$

Make-set (x) ↷

→ min wt = 1 (2,5)

wt	edges
1	2, 5
2	1, 3
3	1, 2
5	5, 6
6	6, 4
7	3, 4
8	3, 7



$2 \in S_2$ $5 \in S_5$ using find (2) find (5)

$S_8 = S_2 \cup S_5 = \{2, 5\}$ using union (x, y)

delete S_2 and S_5

→ next min = 2 (1,3) S_9 = union (S_1, S_3)
1 ∈ S_1 3 ∈ S_3 = {1, 3}
Delete S_1, S_3

→ next wt = 3 (1, 2) 1 ∈ S_9 2 ∈ S_8
S_{10} = S_9 ∪ S_8 = {1, 2, 3, 5} Delete S_9 S_8

→ next wt = 5 (5, 6) 5 ∈ S_{10} 6 ∈ S_6
S_{11} = S_{10} ∪ S_6 = {1, 2, 3, 5, 6} Del (S_{11}, S_6)

→ next wt = 6 (6, 4) 6 ∈ S_{11} 4 ∈ S_4
S_{12} = S_{11} ∪ S_4 = {1, 2, 3, 5, 6, 4} Del (S_{11}, S_4)

→ next = 7 (3, 4) 3 ∈ S_{12} 4 ∈ S_{12}
Since they both belong to same set, skip them

→ next = 8 (3, 7) 3 ∈ S_{12} 7 ∈ S_7
S_{13} = S_{12} ∪ S_7 = {1, 2, 3, 4, 5, 6, 7} Del (S_{12}, S_7)

Why do we use queue using BFS?
→ We want to process the first element that entered the DS. FIFO is required.

BFS Method

Modify Q queue to hold (source, parent) and visited array

→ visite vertex 1

VA | 1 |

Q | (1, -1) |

→ neigh 2, 3 in Q add
visit vertex 2

Q | (3, 1) | (2, 1) | (1, -1) |

VA | 1 | 2 |

→ add neigh of 2 = 5 in Q
visit vertex 3 now

Q | (5, 2) | (3, 1) | (2, 1) | (1, -1) |

VA | 1 | 2 | 3 |

→ neigh of 3 = 4, 7 in Q
visit vertex 5

Q | 4, 3 | 7, 3 | 5, 2 |

VA | 1 | 2 | 3 | 5 |

→ neigh of 5 = 6
visit 7

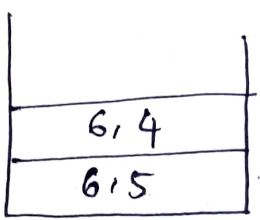
VA | 1 | 2 | 3 | 5 | 7 |

→ neigh of 7 = ∅
Visit 4

VA | 1 | 2 | 3 | 5 | 7 | 4 |

→ neigh of 4 = 6
visit 6

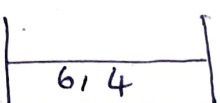
Q



VA | 1 | 2 | 3 | 5 | 7 | 6 |

→ neigh of 6 = ∅
6 from (6,5)
visit 6 from (6,4)
But 6 is already visited. Hence deque (6,4)

Q



Q is empty. STOP.

Searching and Sorting Algorithms

1) Linear Search → $O(n)$ last, not present

2) Binary Search → $O(\log n)$ must be already sorted.

↓ divide and conquer (DNC)

3) Ternary Search (divide array into 3 parts)

4) Fibonacci Search

5) Exponential Search.

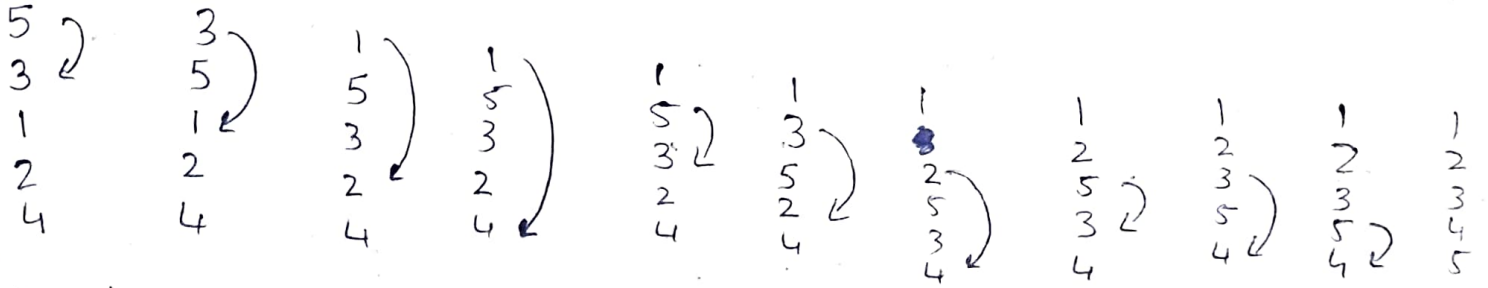
SORTING ALGO

Key ↑ Satellite data →

Roll no	Marks	Name

1) Selection Sort

find the shortest element in the array
put it on the first index
reduce length of array by 1 and repeat.



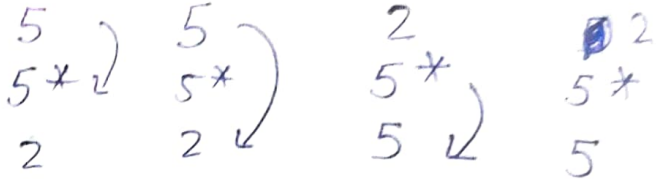
```
for i = 1 to i = n-1
    for j = i+1 to j = n
        if ( a[i] > a[j] )
            swap ( a[i], a[j] )
```

time: $O(n^2)$ worst case
even best case.

space: $O(1)$
↳ extra space during execution.

Stable: if two identical entries are there, then their positions won't be interchanged.

∴ Selection sort is unstable.

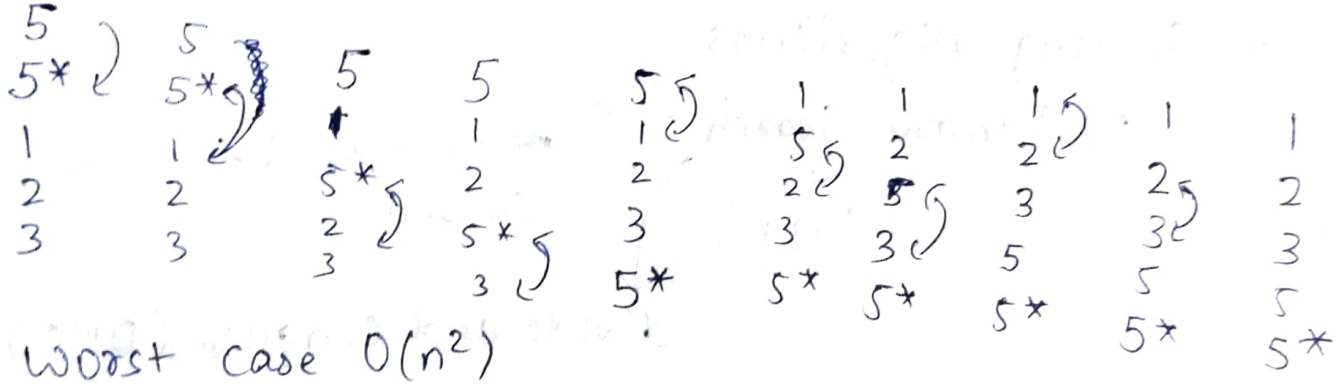


in-place: takes constant amount of extra space.

∴ Selection sort is a in-place alogrithms.

swaps = $O(n^2)$ worst case, $O(1)$ best case

2) Bubble Sort



worst case $O(n^2)$

best case $O(n)$

Stable Algo

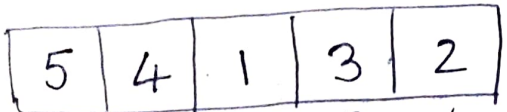
in-place Algo.

swaps = $O(n^2)$ worst, $O(1)$ best.

DSA

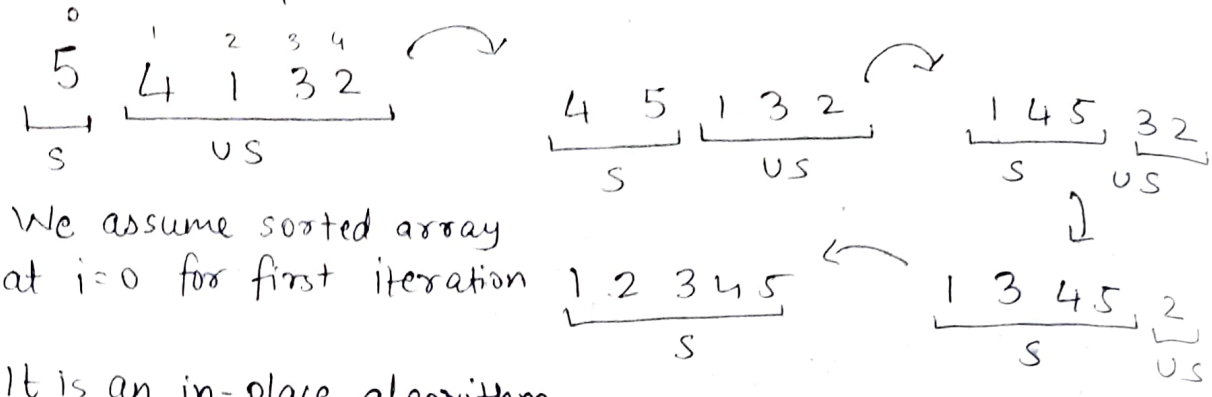
in selection sort, we can find the min element in the whole subarray before swapping it. This will reduce the number of swaps in selection sort.. Hence O(n) for swaps.

we can take all elements of the array and put it in the Linked List in an sorted order. This way, we will always have a sorted DS. This is called insertion sort.



at any time we will have a sorted subarray & unsorted subarray.

Start with i=1 and compare it with all elements before it. And the put it ~~at~~ at its appropriate position.



We assume sorted array at i=0 for first iteration

It is an in-place algorithm

We can use bitwise operations to shift the array elements in a faster way.

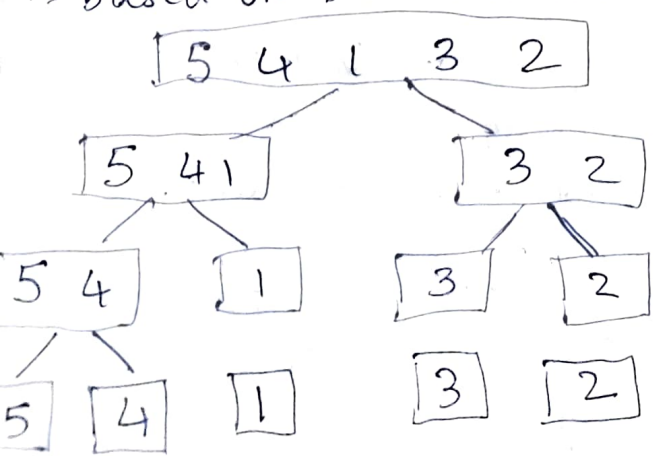
```
insertion sort (A, n)
  for i = 2 to n
     key = A[i]
     j = i-1
     while ( j>0 && A[j] > key)
          a[j+1] = a[j]
          j--
     a[j+1] = key
```

time: worst $O(n^2)$
best $O(n)$
space $O(1)$
in-place → yes
stable → yes

If we feel our input is close to being already sorted, then it will fall into O(n) case. Also if our data is small then we will use insertion Sort.

MERGE SORT

-> based on DNC method and used for large data elements

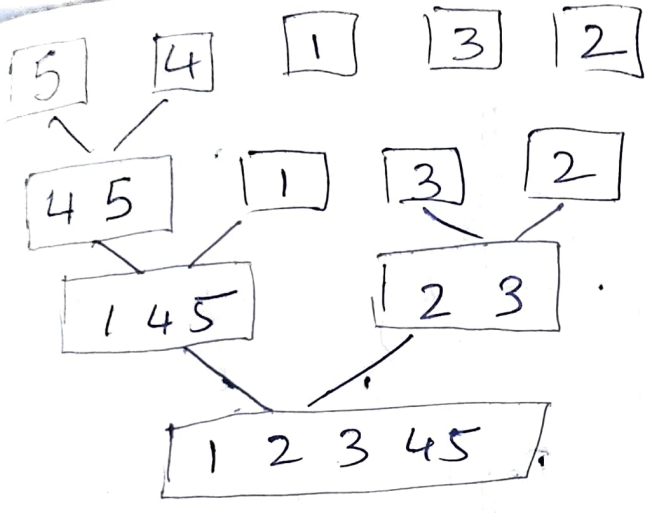


Divide step.

divide until all are single element arrays.

Combine operation is next.

Combine in sorted order.



combine steps.

At each combine step, combine two sorted arrays keeping them again in a sorted order.

time: best / worst / avg $O(n \log n)$

space: $O(n)$

in-place: No

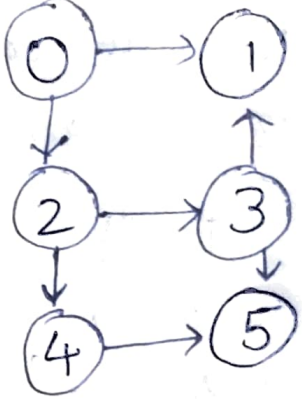
stable: Yes

in practice merge sort is better than heap sort although their time complexities are same in all cases.

DFS

(arrival, departure)

Starting at 0

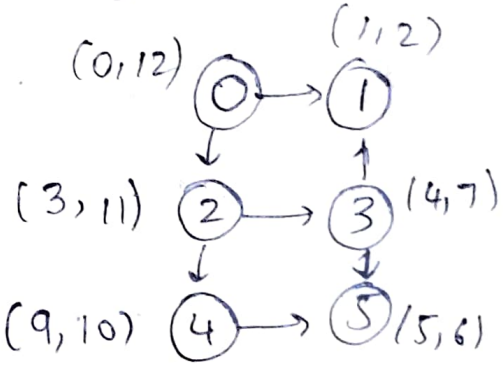


$0 \to 1$ $\quad 1 \to 0$

$0 \to 2$ $\quad 2 \to 3$ $\quad 3 \to 5$ $\quad 5 \to 3$ $\quad 3 \to 2$.

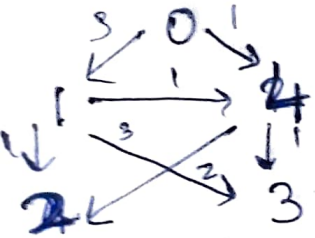
$2 \to 4$ $\quad 4 \to 2$ $\quad 2 \to 0$

$6 \to 7$ $\quad 7 \to 6$



Q) Check whether a graph is bipartite using DFS?

Q) Lowest cost path from $1 \to 3$

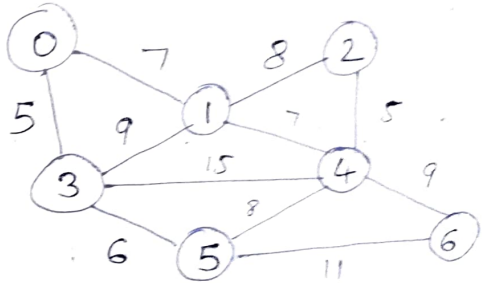


$1 \to 3$ gives 3

$1 \to 4 \to 3$ gives $1+1 = 2$

2) Kruskal's

Q) Selection {4, 1, 3, 9, 7}



QUICK SORT

* comparison based algo
* All comparison based algo have lower bound O(n log n
* DNC algo
* A element in put to its correct position in a single p

P 5 4 1 3 2 4* r

pivot = 4*

i = p-1
j = p

pivot
→ first element
→ last element
→ random
→ median of three

5 4 1 3 2 4*
i j

5 4 1 3 2 4*
i j

5 > 4 j → ++

5 4 1 3 2 4*
i j

4 > 4* j++

1 4 5 3 2 4*
i j

4* > 1 i++ and swap 5 4
j++

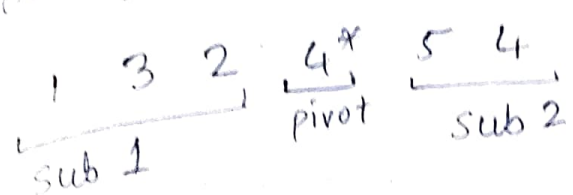
1 3 5 4 2 4*
i j

3 < 4* i++ swap(i,j)
j++

1 3 2 4 5 4*
i j

2 < 4* i++ swap(i,j)
j++

once j reaches pivot swap (i, j)



- final

```
Partition (A, p, r)
i = p-1 , j = p
while ( j < r)
    if.( A[r] < pivot )
        i++
        swap(. A[i], A[j])
    j++
i++
swap ( pivot, a[i])
return i
```

```
Quick Sort (A, p, r)
    if (p == r)
        return
    i = Partition (A, p, r)
    Q.S (A, p, i-1)
    QS ( A, p, i+1)
```

Time: O(n log n) best
O(n²) worst

Space: O(1) for non-stable
O(n) O(log n) for stable

in-place : True

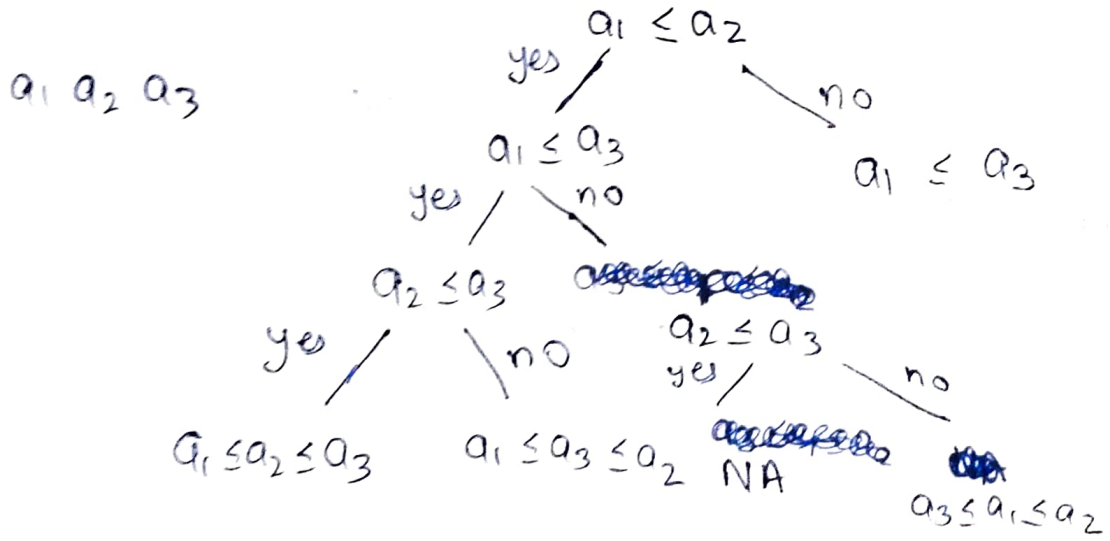
Stability : No

is heap sort stable?

DSA

	Time			Space	Stable	in-place
	Best	Avg	Worst	space		
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	No	Yes
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Yes	Yes
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Yes	Yes
Quick	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(1)$	No	Yes
Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	Yes	No.
Heap	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$	No	Yes

lower bound for ~~sel~~ comparison based algo = $O(n\log n)$



It gives a binary tree with all possible permutations of a_1, a_2, a_3

No. of leaves = 2^h (in our case $n!$)

$2^h \geq n!$ (excluding invalid leaves)

$h \geq \log_2 n!$

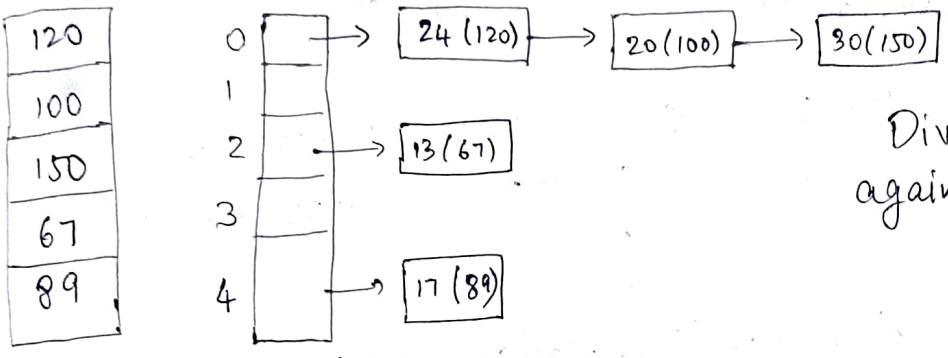
$n! \approx c\, n^n$ sterling approximation

$h \geq \log_2 (c\, n^n)$

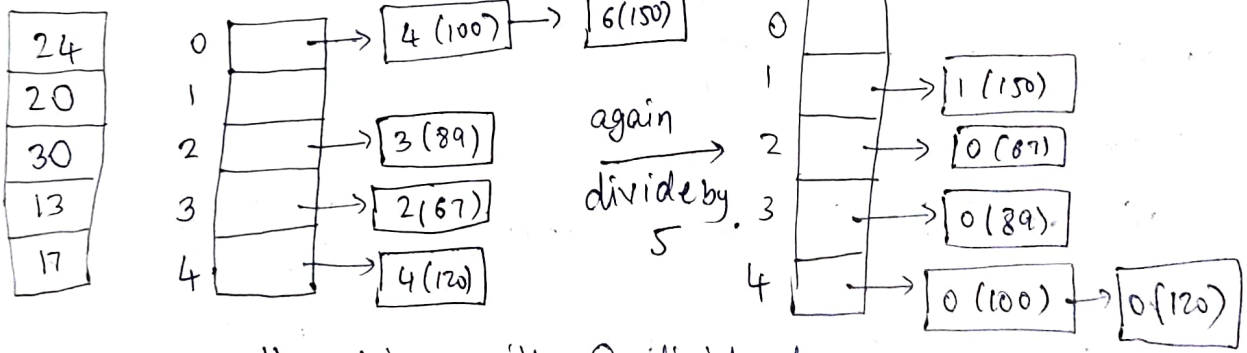
$h \geq n \log n$

∴ any comparison based algo won't go below $n \log n$ height

Non comparison based algo

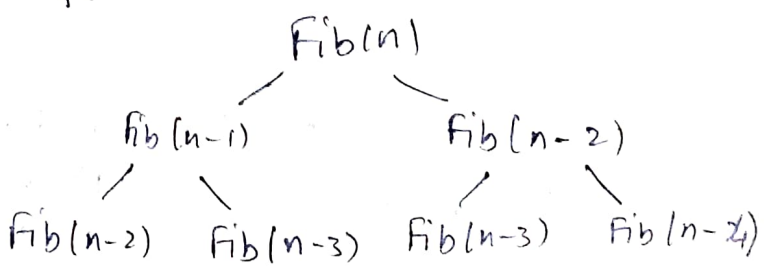


Divide all nos again by 5 ↓ no. of entries



remove all entries with 0 dividend till there are no entries left.

```
Fib (n):
  if n == 0
     return 0
  if n == 1
     return 1
  else
     Fib (n-1) + Fib (n-2)
```



$T(n) = T(n-1) + T(n-2) \cdots$

$= 2^n \quad \therefore O(2^n)$

We can see fib(n-3) is used many times, so instead of calculating it again & again, we store its value for future use : DP → solving optimisation problems

In DP there is an overlap in subproblems, but in DNC subproblems dont overlap.

DP avoids by caching the intermediate results. DNC does more work than neccessary.

```
a[0] = 0.   a[1] = 1
for( i=2; i<=n; i++)
    a[i] = a[i-1] + a[i-2]
```

bottom-up approach
O(n) time
O(n) space.

we can use O(1) by

```
a = 0   b = 1
c = a+b
a = b
b = c
```

3 variable approach.

approaches: bottom-up / Tabulation
Top-down / Memorization.

```
dp[n] = {-1, -1 .... -1}
Fib(n)
    if dp[n] != -1   return dp[n]
    else
        dp[n] = Fib(n-1) + Fib(n-2).
        return dp[n]
```

include base cases of n==0, n==1 or make dp[n] = {0, 1, -1, -1...}

S1 = SUNDAY
S2 = SATURDAY

operations: a) insert
b) delete
c) substitute
All op's cost

min ops^r to convert S1 → S2

ans = 3
convert R→N
delete A & T from SATURDAY

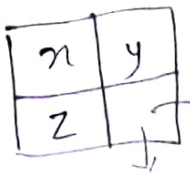
Longest common subsequence

Min Edit Dist (Lavenhestin Dist)

→ used by MS word for spelling correction

```
minEdit ( S1 , S2 , P, Q )
    if S1[P] == S2[Q]
        return minEdit ( S1, S2, P-1, Q-1)
    else
        return 1 + min ( minEdit ( S1, S2, P, Q-1)
                         minEdit ( S1, S2, P-1, Q )
                         minEdit ( S1, S2, P-1, Q-1) )
```

	ϕ	S	A	T	U	R	D	A	Y
ϕ	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
U	2	1	1	2	2	3	4	5	6
N	3	2	2						
D	4						3	4	5
A	5						4	3	4
Y	6						5	4	3



→ min (x, y, z) + 1

if S1[P] != S2[P]

else min (x, y, z)

MA

DSA Tut

Q) Find largest m such that every simple connected graph with n vertices & n edges contains atleast m diff spanning trees

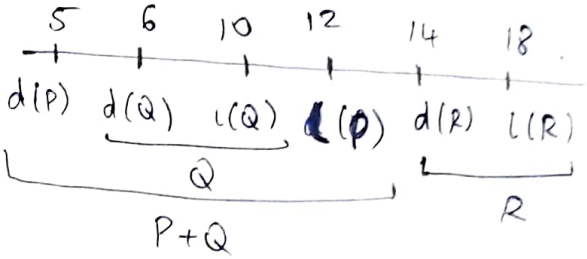
ans = 3

Q) Consider DFS of an undirected graph with 3 vertices P; Q, R. Let d(u) be the time the node is visited first and l(u) be the time the node is visited last. How many connected components are there? When

d(P) = 5 d(Q) = 6 d(R) = 14

l(P) = 12 l(Q) = 10 l(R) = 18

2 components

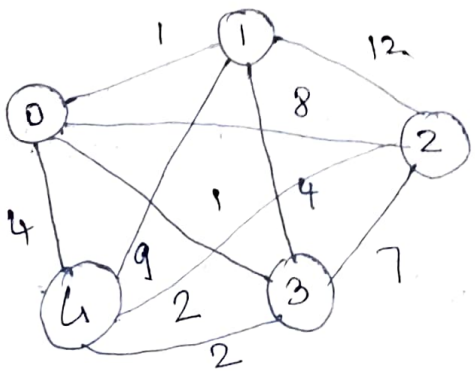


Q) Consider weight matrices

Let 0 be the leaf node

Find min possible weigh of MST?

$$W = \begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 1 & 8 & 1 & 4 \\ 1 & 1 & 0 & 12 & 4 & 9 \\ 2 & 8 & 12 & 0 & 7 & 2 \\ 3 & 1 & 4 & 7 & 0 & 2 \\ 4 & 4 & 9 & 2 & 2 & 0 \end{array}$$



Since 0 is the leaf node, it needs to be processed last therefore take this matrix and find its MST and then add 0 vertex with the min weight

Q) Find min possible weight of path P from 1 to 2 such that # edges in P = 3

Q) There is a list of no.s. Some of the entries are ∞. The list is sorted. Find where the ∞'s start.

Q) In Quick Sort, if $(n/4)$th smallest element is selected as pivot using $O(n)$ time. What is WC TC

Q) There is an array of n no.s, n is even. Find min & max of these n no.s. How many comparisons are needed.

Q) Find min WCTC to find ceiling of a no. in an sorted array