Degree of vertex: # edges incident on the vertex

8.11.2022

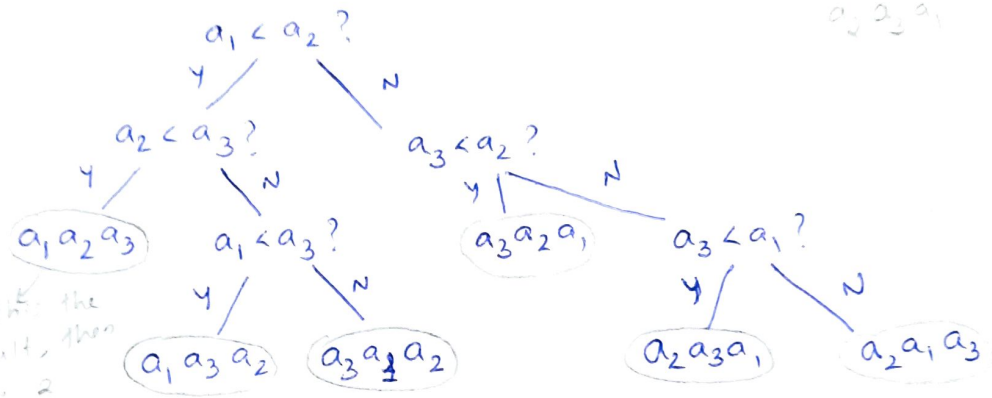
learn counting sort

Theorem 1: merge sort is optimal.

Theorem 2: any deterministic comparison-based sorting algorithm requires atleast $n \log n$ comparisons

} the lower bound of comparison based sorting algo is $n \log n$.

Proof: Consider $a_1, a_2, a_3$ that we need to sort.
We use the decision tree.




There are $n!$ leaf nodes in the tree.

Height of the tree = $h$

Not all leaf nodes are at the lowest level $h$

↳ # of leaf nodes at lowest level $= 2^h$

Then: $2^h \geq n!$

Taking log on both sides: $\log 2^h \geq \log(n!) \Rightarrow h \geq \log(n!)$

Now, we want to find the bound for $\log(n!)$ and prove that:

$$\log(n!) = O(n \log n)$$

$$\log(n!) = \log(n \cdot (n-1) \cdot \ldots \cdot 1)$$
$$\leq \log(n \cdot n \cdot n \cdot \ldots) \quad (n \text{ times})$$
$$\leq \log(n^n)$$
$$\leq n \log n$$

$$\Rightarrow \log(n!) = O(n \log n)$$

Now: $\log 2^h \geq \log(n!)$

$h \geq \log(n!)$

$h \geq n \log n \Rightarrow$ height is max. req to be $n \log n$ which means max $n \log n$ comparisons

$\Rightarrow T(n) = O(n \log n)$

## Graphs:

$\deg(v)$ = # incident edges

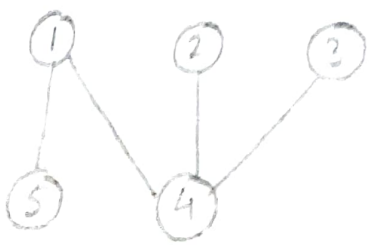
if the graph is directed, then: $\text{indeg}(v)$: # incident edges enter

$\text{Outdeg}(v)$: # incident edges leave

★ In a group of n people, the # people who make odd number of handshakes will be an even number.

↳ Handshake Lemma in Graph Theory

Proof:




$\deg(1) = 2$

$\deg(2) = 1$

$\deg(3) = 1$

$\deg(4) = 3$

$\deg(5) = 1$

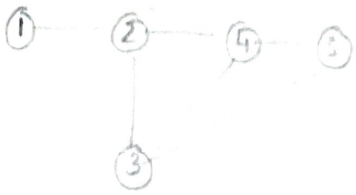
8

$\Rightarrow \left| \sum_{i \in V} \deg(i) = 2E \right|$ (1 edge is counted twice in an undirected graph)

| # people | # handshakes | # edges |
|---|---|---|
| e | e | e |
| o | o | e |
| e | o | e |
| o | o | o ← this will not occur as it violates the above rule |

9.11.2022

## Walk in Graph




- Walk: a sequence of vertices
- path: a walk where the vertices / edges are not repeated.

1 2 4 3 5 ← a path, vertices aren't repeated

- tour: a closed path
- loop: if an edge connects vertex to itself

★ there can be parallel edges b/w 2 vertices

$\{v_1, v_2\}$ —— $\{v_5, v_3, v_6\}$

★ Hypergraph: 1 edge connects multiple vertices

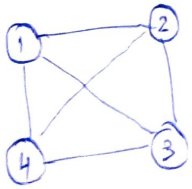
simple graph: with no loops and parallel edges

regular graph: degree of all vertices are same

★ Degree of vertex:

- simple graph, $n = 1$
- complete graph, every vertex is connected to all other vertices

$K_4$:



★ Connected graph: if all vertices are reachable from any vertex, then it is connected.


} not connected

Q. Min # edges required to ensure connectivity?

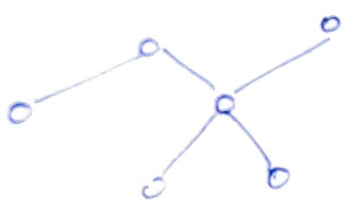
if there are 'n' vertices, n - 1 is the min # edges.


} 2 nodes
1 edge

## Tree

- Connected Acyclic Graph:

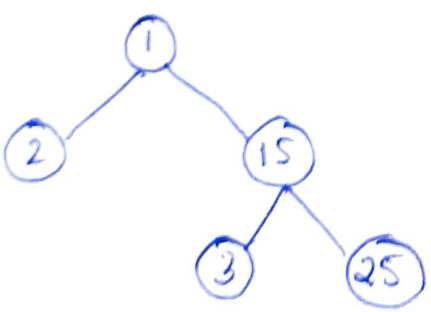
Eg:



## Binary Tree

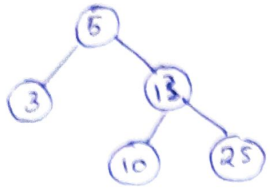
rooted tree

every node has atmost 2 children.




★ children aren't ordered

- **Binary Search Tree**
  - values of left side must be less than the parent and right side must be greater than parent.



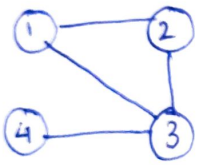

★ children nodes are ordered.

## Representation of Graph

- Adjacency matrix
- Adjacency list

### → Adjacency matrix

4×4 matrix




$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

★ If graph is undirected, then matrix will be symmetric

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 3 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

★ **Path matrix** : where each entry shows us how many paths are from each node with length 2.

$A^3$ path matrix will show how many paths from each node with length 4, 3

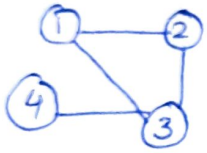
★ Transitive Closure:

$A \cdot A \cdot A \cdots$ upto k

$A^k$, we will get saturated matrix

i.e. $A^k = A^{k-1}$ then its Transitive closure, i.e. saturated

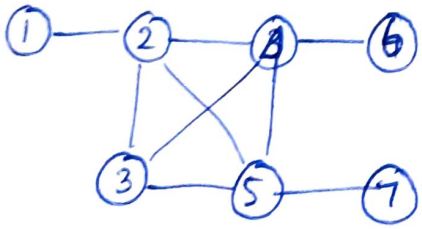
→ Adjacency List




→

1 → 2 → 3
↓
2 → 1 → 3
↓
3 → 1 → 2 → 4
↓
4 → 3

Clique




Subgraph is Complete

use py package.
Networ
g = Graph()
g.addedge(
g.add
for the

10.11.22

## Randomized Quicksort

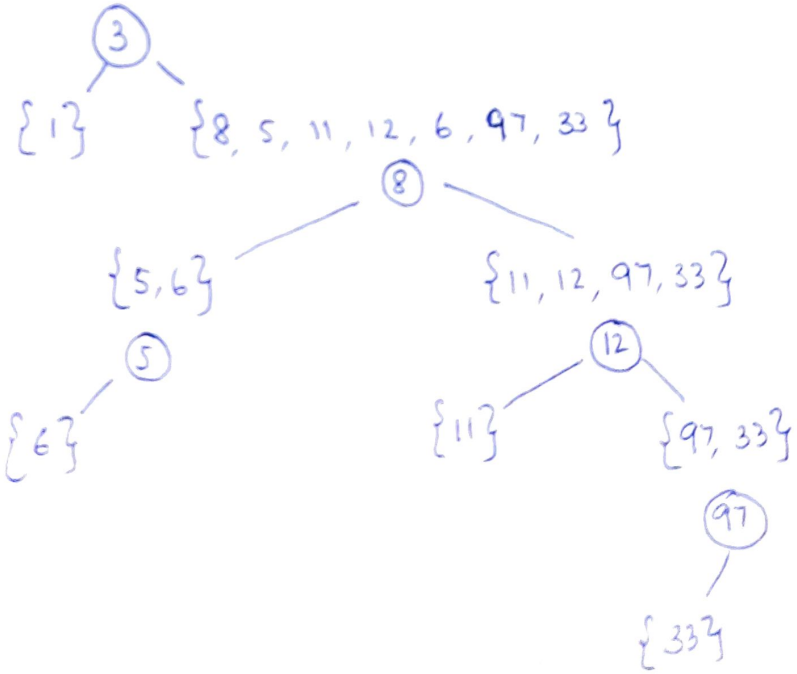
```
QS
  p = partition()
  QS (s, p-1)
  QS (p+1, e)
```

$$T(n) = T(p) + T(n-p) + \theta(n)$$

- randomly select pivot element
- partition the array based on pivot
- Qsort is applied recursively to left array
- Qsort is applied recursively to right array.

1 8 5 3 11 12 6 97 33

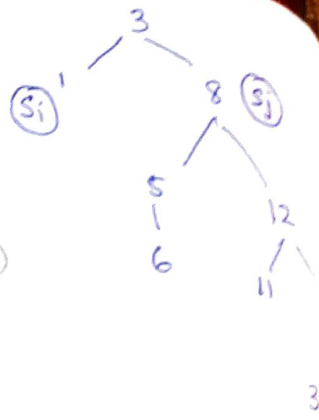
↑
pivot element




* This tree is BST

  i.e. Quicksort basically constructs a BST

Let $S_i$ be the element with rank i
Let $S_j$ be the element with rank j




Now we introduce :

Random variable $x_{ij}$ – Indicator Random Variable {0, 1} (IRV)

$$x_{ij} = \begin{cases} 1, & \text{if } S_i \text{ is compared with } S_j \\ 0, & \text{otherwise} \end{cases}$$

∴ Total complexity = Sum of all $x_{ij}$

$$\text{Total \#comparisons} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} x_{ij}$$

$$X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} x_{ij}$$

$$E(X) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[x_{ij}]$$

(Expectation value)

— linearity of expectation

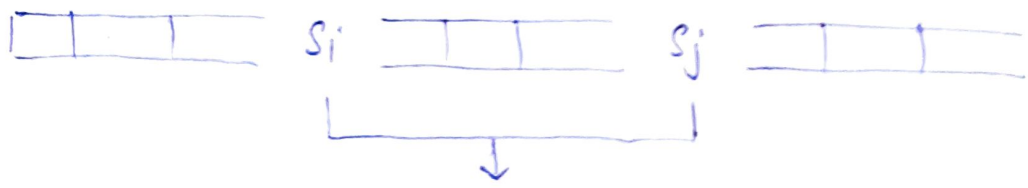
$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} P_{ij}$$

it can be 0 or 1 (IRV)

if comparison is done, then prob = 1, else prob = 0

Let $S_i < S_j$

In array :




- If we take any element in this range, then $S_i$ will go to one subtree and $S_j$ will go to the other subtree.

→ they won't be compared in the future.

- The only possibility of $S_i$ and $S_j$ being compared is if any of $S_i$ and $S_j$ is selected before selecting elements b/w $S_i$ and $S_j$

$P_{ij}$ (probability value) $= \frac{2}{j-i+1}$

$$E(X) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{k+1}$$

$$\leq \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k+1}$$

$$\sum_{k=1}^{n} \frac{2}{k+1} = \frac{2}{2} + \frac{2}{3} + \frac{2}{4} + \cdots \frac{2}{n+1}$$

$$= 2\left[\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} \cdots \frac{1}{n+1}\right]$$

replace replace

$$\leq 2\left[\frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} \cdots\right]$$

1/4 4 times

$$\leq 2\left[1 + 1 + 1 + \cdots\right]$$

log n terms

$$\leq O(\log n)$$

Instead of proving $\leq$, lets prove for $\geq$

$$S \geq 2\left[\frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \cdots\right]$$

instead of 1/3, give 1/2

$$s \geq 2\left[\frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} \cdots\right]$$

$\log n$ times

$$s \geq O(\log n)$$

$$\therefore \quad s = \Omega(\log n) = \log n$$

$$E(x) \leq \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k+1}$$

$$\leq \sum_{i=1}^{n} \log n$$

$$= O(n \log n)$$

* Randomized Quicksort is optimal.

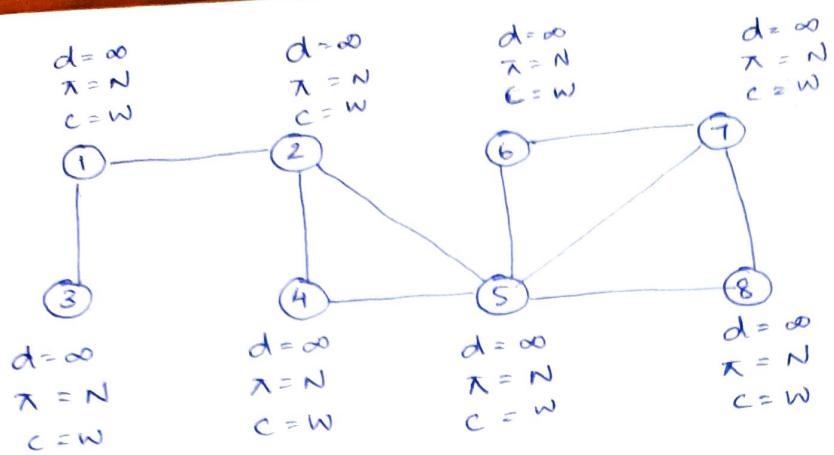
14.11.2022

$\pi$ = parent
d = distance
c = color

BFS

```
BFS(G, s)
    for each vertex u ∈ V-{s}                          } O(V)
        u.color = white, u.d = ∞, u.π = NIL
    s.d = 0, s.π = NIL, s.color = gray
    Q = {}, Q.enqueue(s)
    While Q ≠ {}                                       } O(E)
        u = Q.dequeue
        for v ∈ adj[u]
            if v.color == white
                v.color = gray, v.d = u.d+1, v.π = u
                Q.enqueue(v)
        u.color = black
```

start node = 2




| | 5 | 4 | 1 | 2 |
|---|---|---|---|---|

vertex 2 color is W → change to gray
vertex 1 color is W → change to gray and dist becomes 1, parent = 1

1 → 2 → 3
↓
2 → 1 → 4 → 5
↓
3 → 1
↓
4 → 2 → 5
↓
5 → 2 → 4 → 6 → 7
↓
6 → 5 → 7
↓
7 → 5 → 6 → 8
↓
8 → 5 → 7

① d = 1, π = 2, c = G  and  ② d = ∞, π = N, c = G

④ d = 1, π = 2, c = G

⑤ d = 1, π = 2, c = G

Processing 2 is over, so change color to black.

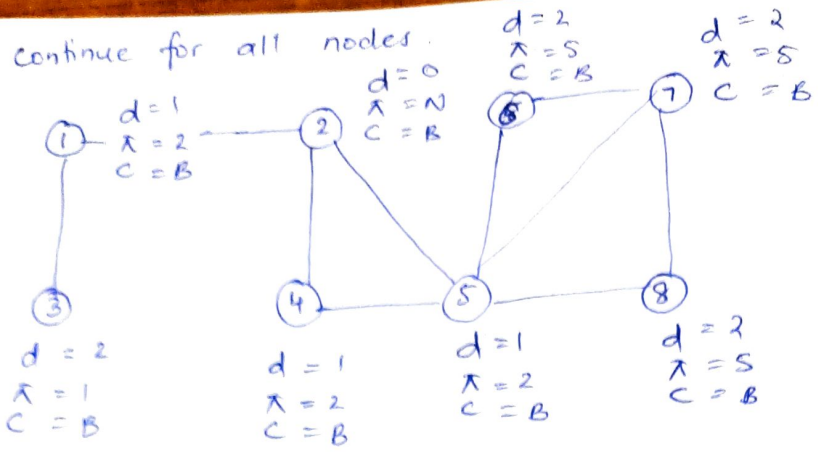
② → d = ∞, π = N, c = black (B)

| | 3 | 5 | 4 | 1 | 2 |
|---|---|---|---|---|---|

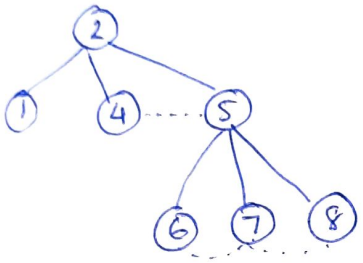
Now: processing 1

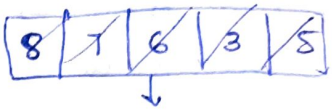
③ d = 1, π = 1, c = G } processing is over so: ① d = 1, π = 2, c = G B

continue for all nodes



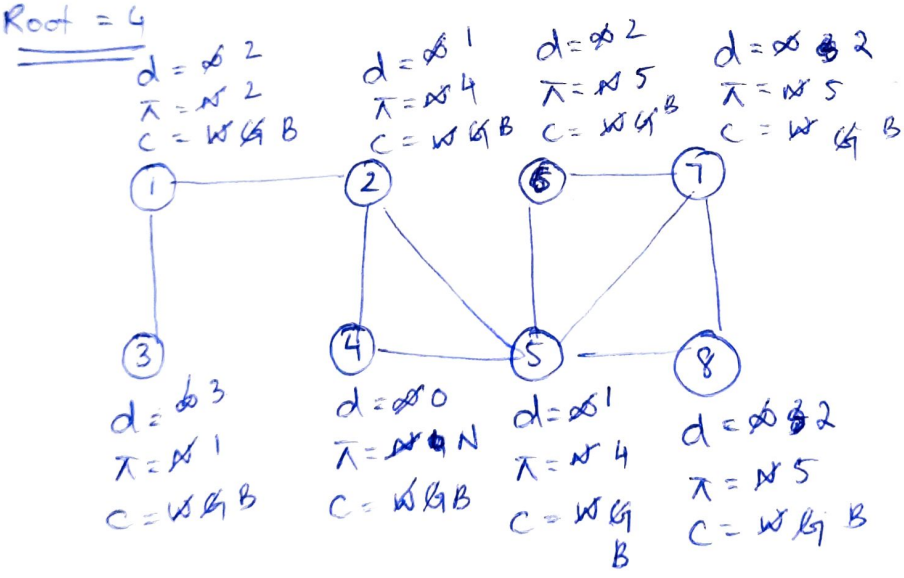

BFS tree rooted at 2:

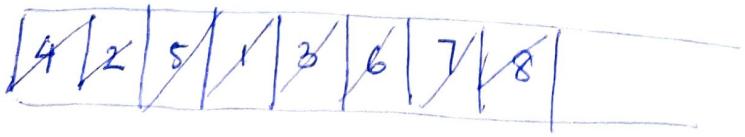


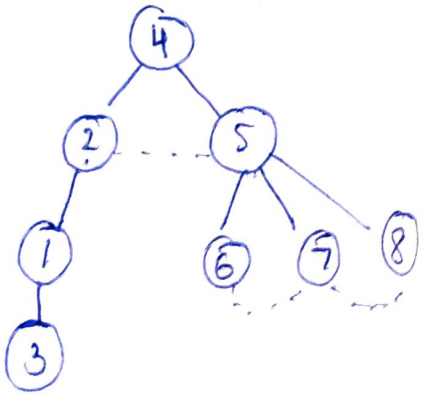

take 6
look any adj vertex and check if

Root = 4




tree



## Complexity

$O(V+E)$ if V is bigger, then $O(V)$
if E is bigger, then $O(E)$

★ **Lemma A**

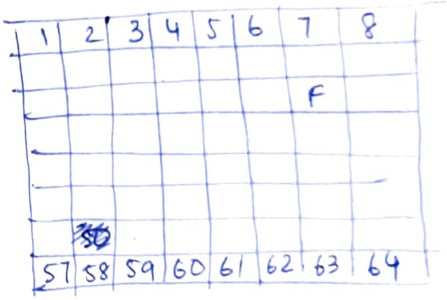
Let $\delta(a,b)$ be the shortest distance from ~~a~~ a to b.
if $(u,v) \in E$, then $\delta(u,v) = \delta(s,u) + 1$ or vice versa

When we draw BFS tree, all non tree edges will connecting siblings, parents

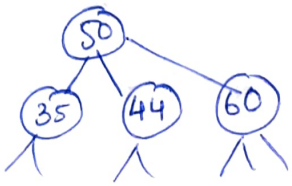
16.11.22

Q Given the initial and final positions, determine the # moves for a Knight to reach the final pos.

Consider the board as a graph, and nodes from the initial pos to the other pos through the move of a knight.

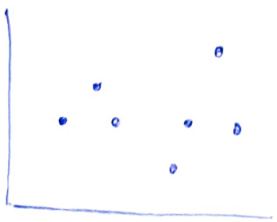



Eg:




Use BFS to find # moves.

Q.



→ find min turns/ lines to connect all the poi

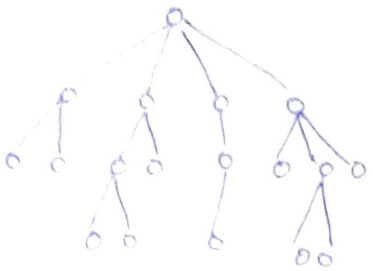
Other possible Q:

→ rectilinear covering problem (np-hard in almost all $R^n$)
→ minimize turns + TSP

BFS
NP-hard proof

★ Diameter of a graph: the ~~long~~ longest path of the graph
[the longest shortest distance]

Q Find the diameter of the following tree:



- use BFS on any node and find the farthest nodes
- perform BFS again, but only on the farther nodes to obtain the farthest nodes from these nodes.
  ↳ through this step, we will find the diameter.

DFS

```
DFS (G)
    for each u ∈ G.V
        u.color = white
        u.π = NIL
    time = 0
    for each u ∈ G.V
        if u.color = white
            DFS_VISIT (G,u)

DFS_VISIT (G, u)
    time = time + 1
    u.d = time
    u.color = gray
    for each v ∈ adj[u]
        if v.color == white
            v.π = u
            DFS_VISIT (G,u)
    u.color = black
    time = time + 1
    u.f = time.
```

π = parent

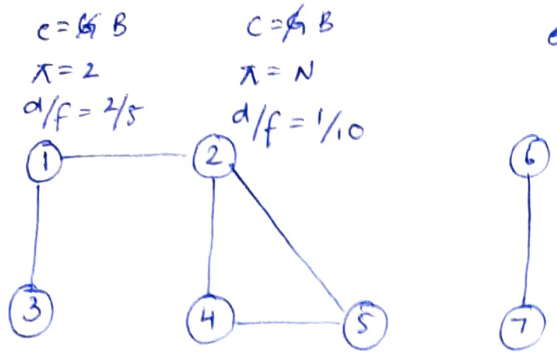
d = discovery time (when we 1st visit node)

f = finish time (when we finish visiting all neighbours)

Color: white (hasn't been visited)

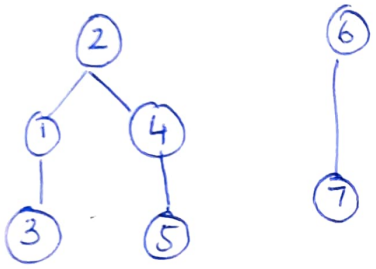
gray (visited for ...)

black (finished visiting)




1 → 3 → 2
↓
2 → 1 → 4 → 5
↓
3 → 1
↓
4 → 2 → 5
↓
5 → 2 → 5
↓
6 → 7
↓
7 → 6

## DFS trees



A group/collection of trees = forest

★ Parenthesis property of DFS: DFS will create a perfectly balanced parenthesis tree.

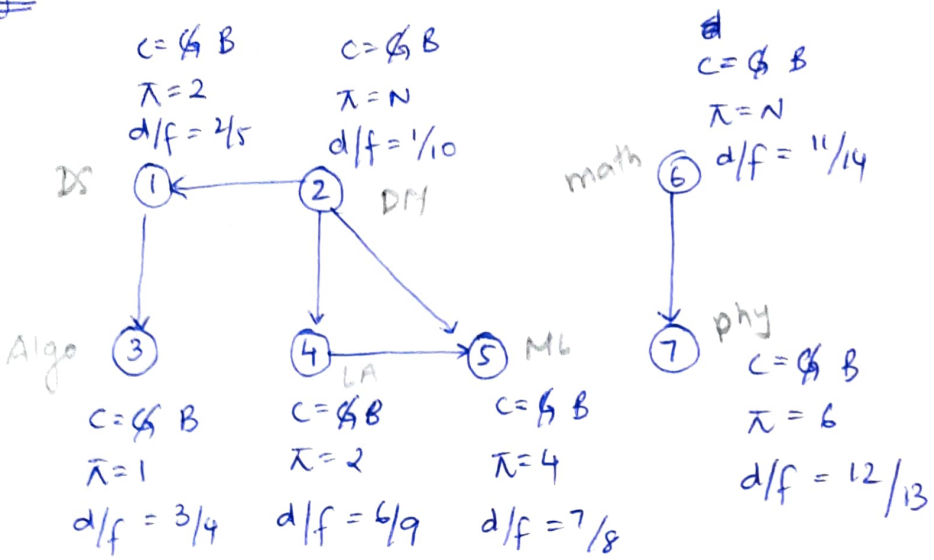
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ( | ( | ( | ) | ) | ( | ( | ) | ) | ) | ( | ( | ) | ) |
| 2 | 1 | 3 | 3 | 1 | 4 | 5 | 5 | 4 | 2 | 6 | 7 | 7 | 6 |

↓ has been discovered

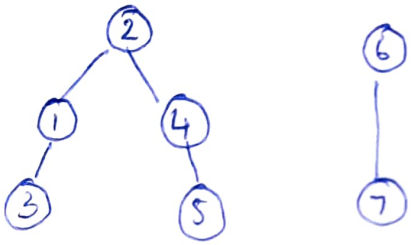
↓ has been finished

DFS can be used to find connected components of a graph. In the case of directed graphs, it can be used to find strongly connected components.

Eg:




DFS trees:




★ Consider that each node represents a subject. The direction of the edge rep that one sub is a prerequisite for another sub

Considering the ↓ order of finishing times, we get:

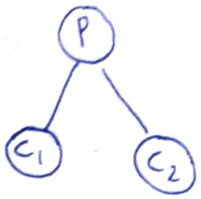
math, phy, DM, LA, ML, DS, Algo.

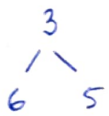
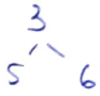
⇒ Ordering / sorting the vertices of a graph in a particular way is called topological sort.

pa. 11. 2022

## Binary Heap

- A spl type of tree data structure.
- 2 types – min heap ($P < C_1, C_2$)
  – max heap ($P > C_1, C_2$)




Eg:  is a min heap or 

 or  ⇒ max heap

### Operations:

- insert() $\log n$
- delete()
- extract-min() $\log n$
- build() $n$

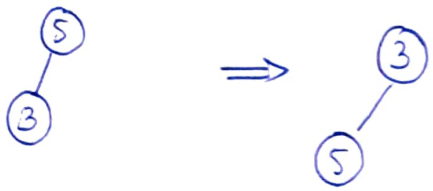
} min heap

in min heap ... add from L to R

Eg: ~~5~~ 5 3 2 7 9 6 8 1 4 [for insertion]

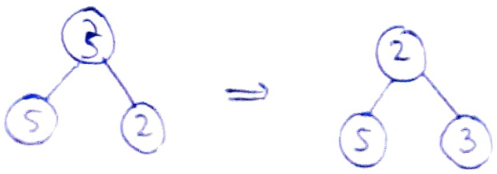
- Initially heap is empty. Add the 1st element

- Add 3.

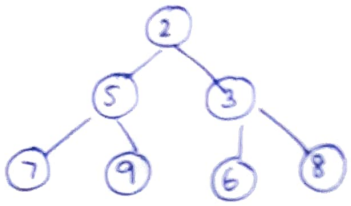


(swapped to satisfy min property)

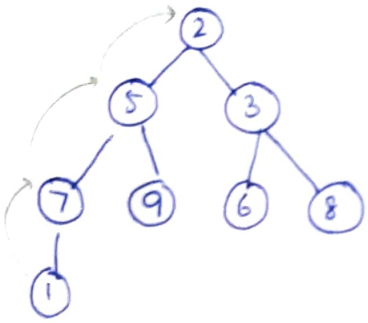
- Add 2:



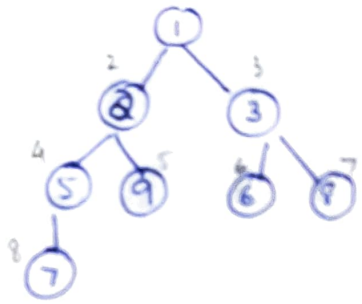
- Add 7, 9, 6, 8




- Add 1



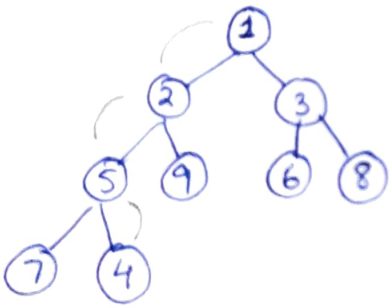

⇒




representing in the form of an array

| 1 | 2 | 3 | 5 | 9 | 6 | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | |

- Add 4




⇒

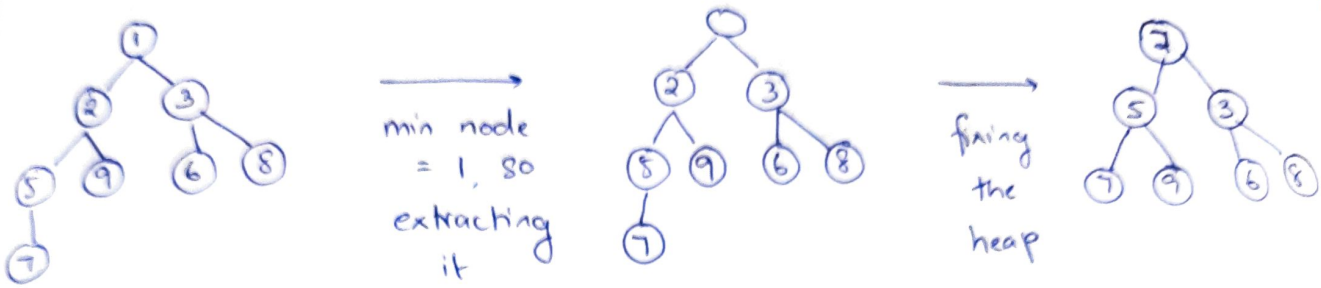
Complexity = $O(n \log n)$

we do swapping along the height of tree

height = $\log n$

extract_min()

The min node is the root. Then perform swapping (fixing property) to fix the heap.




Complexity = $O(\log n)$

★ parent = $i$

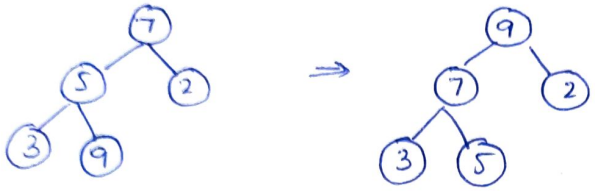
left child = $2i$

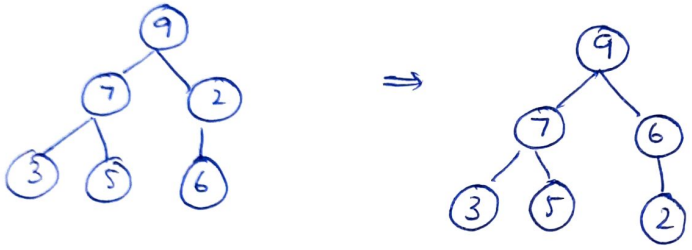
right child = $2i+1$

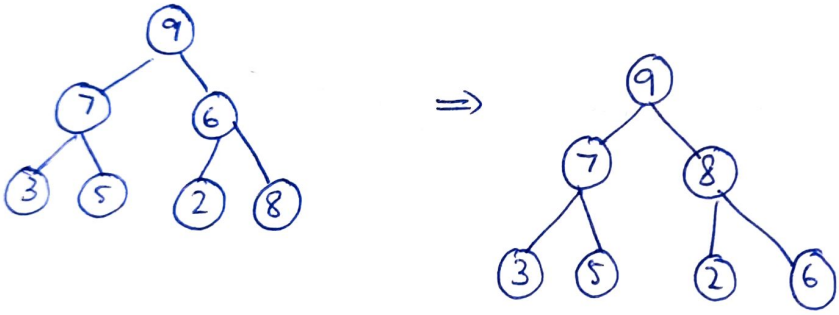
given a child $i$, its parent is at position $\lfloor \frac{i}{2} \rfloor$ of the array.

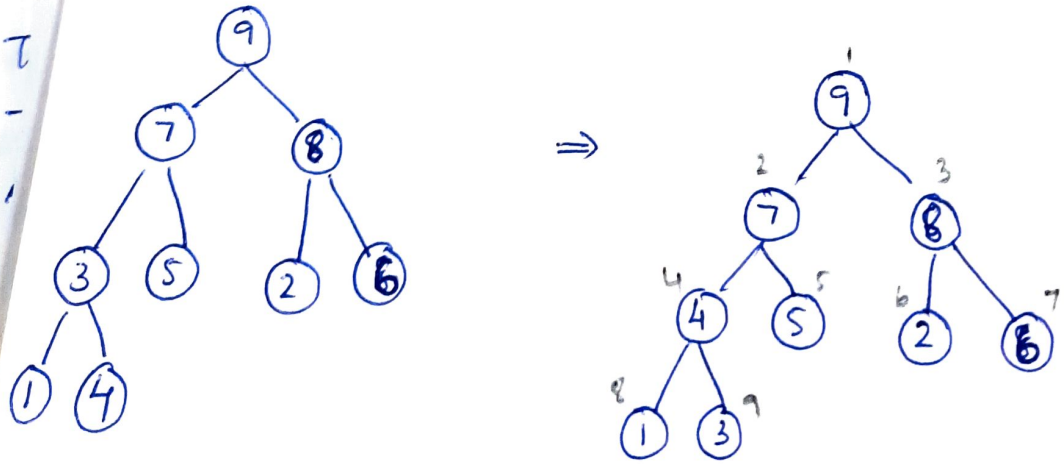
## Max heap

5 3 2 7 9 6 8 1 4









Array:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| 9 | 7 | 8 | 4 | 5 | 2 | 6 | 1 | 3 |

## Heap sort

```
for i = n to 1
    x = extract-max()
    H[i] = x
```

Complexity = $O(n \log n)$

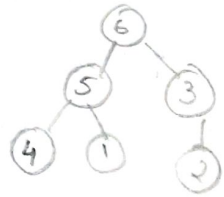
↳ optimal

H = | 9 | 7 | 8 | 4 | 5 | 2 | 6 | 1 | 3 |

when i = 9, 8 7 6 4 5 2 3 1 | 9

i = 8, 7 5 6 4 1 2 3 | 8 9

i = 7, 6 5 3 4 1 2 | 7 8 9 →




x = 7

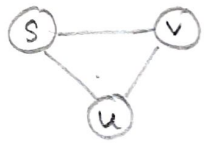
⋮

Dijkstra (G, s) → solves the problem of single source shortest path

```
for i = 1 to n
    d[i] = ∞
d[s] = 0
Q = G.V (based on d)        // const min heap
S = { }
While Q ≠ { } :
    u = Extract_min (Q)
    for each (u,v) ∈ E
        Relax ((u,v))
    S = S ∪ u
```

Constraint: graph G shouldn't have weights, i.e. $w > 0$

```
Relax ((u,v))
if d[v] > d[u] + c(u,v)
    d[v] = d[u] + c(u,v)
    v.π = u
```



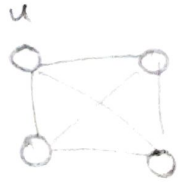

if $d[v] > d[u] + c$ from S then we update and state that v's parent is now u.

★ in the worst case, v will relax x times, where x is the # possible paths to V.

★ how many paths in a complete graph?

$A^1, A^2, A^3, \ldots A^t$

$1 + 2 + 2 + \ldots$




$\ldots 2(n-2)(n-1)$

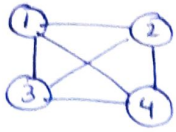
lab

heap

heapsort

Dijkstra

BST binary tree




$(n-1) \leftarrow n(E) \rightarrow {}^{n}C_2$

close to or is $(n-1)$ — Sparse graph

close to or is ${}^{n}C_2$ — dense graph

* When we reduce the number of edges, the distance/ length of the shortest path increases.

For a dense graph G, when we ↓ # edges, we want to construct a new ~~graph~~ such that:

$d^*(u,v) \leq k \cdot d(u,v)$ — Multiplicative graph spanner

$d^*(u,v) \leq d(u,v) + c$ — Additive graph spanner.

eg: $10 \leq 2 \times 5$

21.11.2022

## Algo Design Methodology

- Brute Force: try all possibilities
- Divide and conquer: divide the problem into subproblems, find sol to the subproblems and combine the solutions.

Eg: quicksort, mergesort

matrix multiplication [originally it takes $O(n^3)$]

↳ under divide and conquer:

- Strassen's algo: $O(n^{2.81})$
- Winogauds: $O(n^{2.14})$
- theoretically proved, but no algo exists: $O(n^2)$

- Greedy method

## Greedy Method:

- Suitable for optimization problems (but not all of them)
- to be used, the following conditions must be satisfied.
  - optimal substructure
  - greedy choice.

in dijkstra, relaxation is a greedy choice

→ **Greedy choice**

We make a choice based on local maxima or local minima in the hope that we reach global maxima / minima.

→ **Optimal substructure:**

The optimal solution of a problem contains the optimal solution of its subproblems also.

Eg: Amount = 24 Rs

coins available : 1 2 5

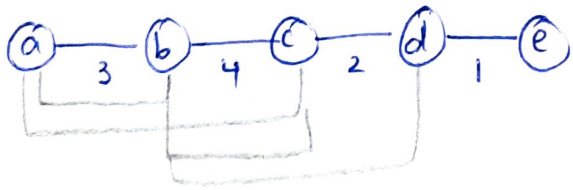
choosing 4 5Rs coins, we are left with 4 Rs

↓

we need to do the same thing for this amount

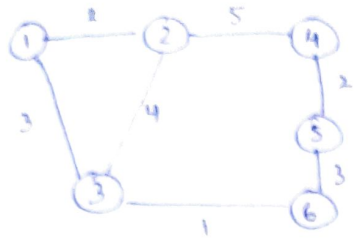
→ its a subprob.

Eg:




Finding the optimal path b/w (a) and (e) is basically finding the optimal path b/w (a)-(b), (b)-(c), (a)-(c), etc...

## Minimum Spanning Tree (MST)




Aim to minimize the total edge cost while finding a tree that connects all the nodes.

→ Kruskal's Algo [select n-1 edges to ensure connectivity with min weight]

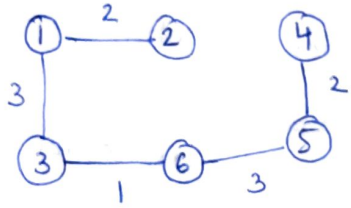
1. Sort edges in the asc. order of their weights
2. Select the edge with least possible weight and add to MST if it doesn't create cycle.

**Step 1:**

| e | w | |
|---|---|---|
| (3,6) | 1 | ✓ |
| (1,2) | 2 | ✓ |
| (4,5) | 2 | ✓ |
| (1,3) | 3 | ✓ |
| (5,6) | 3 | ✓ |
| (2,3) | 4 | } |
| (2,4) | 5 | } |

nodes 2, 3, 4 belong to the same component so we cannot consider these edges. Otherwise, a cycle will be formed.

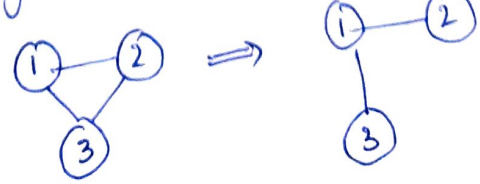
**Step 2:**




Total weight = 2 + 3 + 1 + 3 + 2
= 11

★ Greedy choice: choose edge with min weight

Optimal substructure:

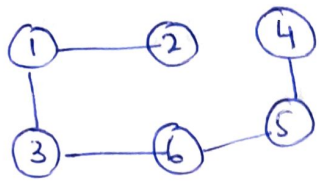



is a subproblem.

MakeSet() : selects a node that represents a subgraph.

| Nodes | rep(r) | Adding (3,6) & (1,2) r | Adding 4,5 (r) | Adding 1,3 (r) | Adding (r) & |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 1 | 1 | 1 | 1 |
| 3 | 3 | 3 | 3 | 1 | 1 |
| 4 | 4 | 4 | 4 | 4 | 1 |
| 5 | 5 | 5 | 4 | 4 | 1 |
| 6 | 6 | 3 | 3 | 1 | 1 |

Note: connect 2 nodes only if their rep nodes (r) are different.




★ Are MST and tree in single source shortest path the same?

No, as their objectives are different.

MST wants to minimize total cost while SSSP wants to find the shortest path from source to all other nodes.