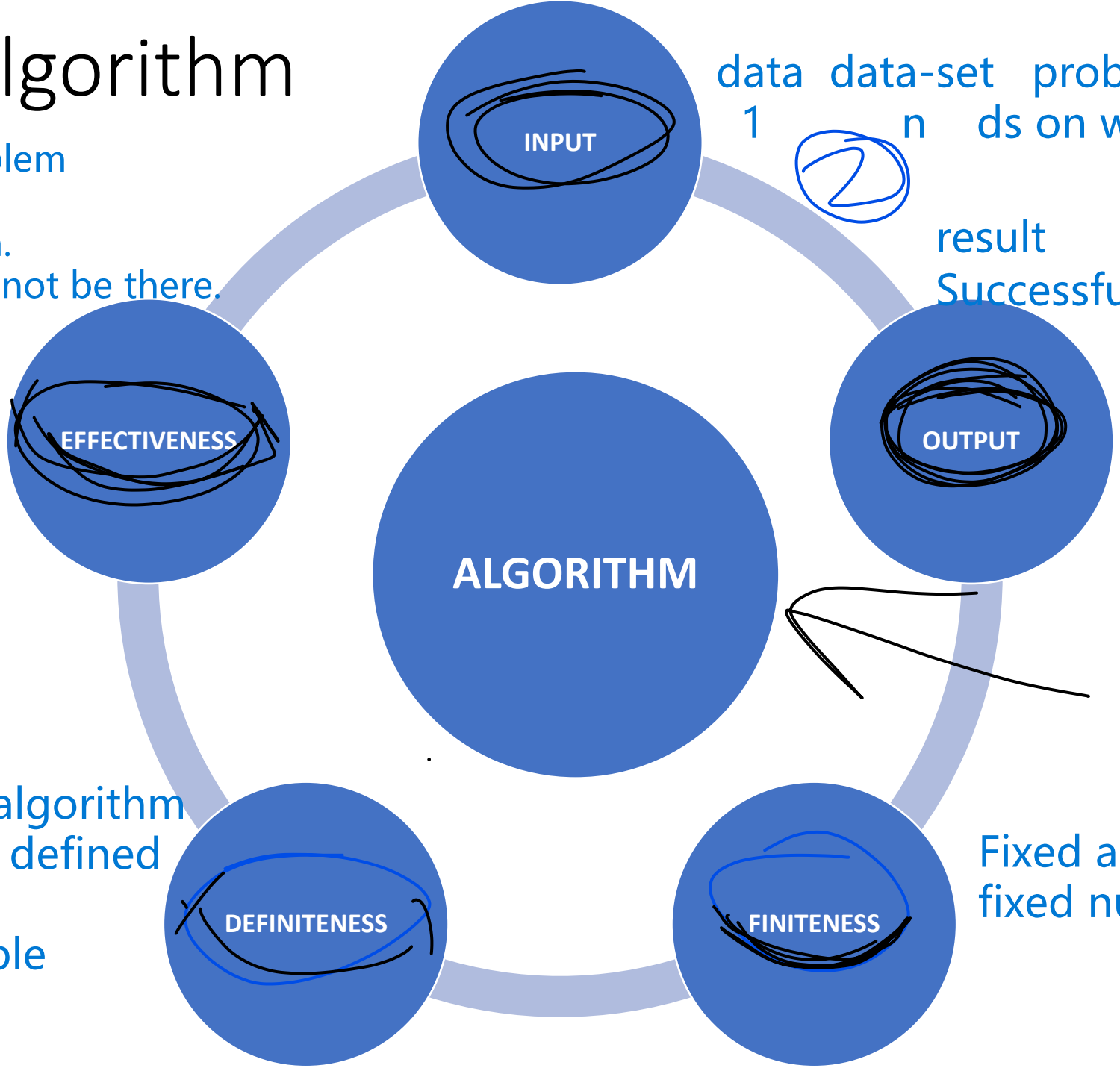
Algorithms

What is Algorithm ?

- Set of finite steps if followed will solve given problem in finite steps and finite time.

Pillars of Algorithm

Each step should
either solve the problem
or should help us
to solve the problem.
Otherwise, it should not be there.



data data-set problem set
1 n ds on which algo will run

result response.
Successful unsuccessful

Each step of the algorithm
should be clearly defined
otherwise,
it won't be possible
to code it.

Fixed amount of time,
fixed number of steps.

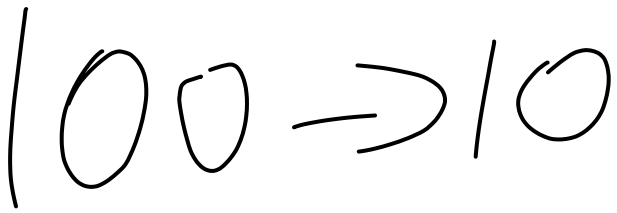
Analysis ?

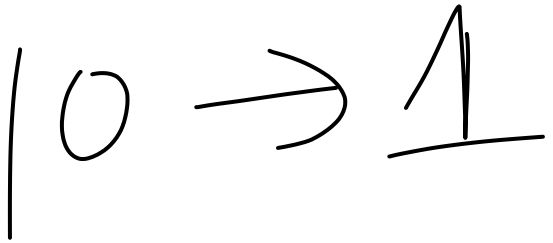
- **Why ?**
  - **So one can predict time and space complexity and then select the one.**

100 ⇒ 10 ⇒ imp ⇒ Best

Analysis ?

- **When ?**
- **Priori**
  - **Done before implementation**
  - **Using pen paper**
- **Posteriori**
  - **Done after implementation**
  - **Using software by professionals**





Analysis ?

- **What we get ?**
  - **Time Complexity** Amount Of time taken for a code to execute over n inputs.

$$T(P) = t_{compile}(P) + t_{execute}(P)$$

  - **Space Complexity** Amount of space needed to store it in some kind of memory.

$$S(P) = S_{identifiers}(P) + S_{instructions}(P)$$

Analysis ?

- **On What?**
  - **Best Case** Minimum or no work involved?
  - **Worst Case** Maximum number of steps for size of n, n^2, n^3, n!
  - **Average Case** For a problem size of n nearly 50% steps are involved n/2.

For fast prediction with okay accuracy, worst-case only.
For most accurate prediction, which would take time, combine best and worst

why worst for analysis: enough number of steps to predict accurately.

Time Complexity

- **Comments –0**
- **Operation----1**
- **Loop----------n+k**

n=10
3n+5=35

```
for(int i =0;i<=n;i++)
    System.out.println(
```

n:3
int i=0: :1
i<=0:0,1,2,3,4 :n+2
i++:0,1,2,3,4 :n+1
S.op():0,1,2,3 :n+1

Total: :3n+5
time in steps
This time is hardware or processor independent.

O(n)

Px -> 5 inst/sec -> 35 => 7sec
Py -> 15 inst/sec -> 35 => 2.xsec

Problem

Complexity will always give you the number of steps, which are then compared with time.

- **An algorithm takes 0.5milli seconds for 100 inputs calculate time if input is increased to 500 and if complexities are**
- **Linear** n
- **Quadratic** n^2
- **Cubic** n^3
- **Log based** $\log n$

Linear n 62.5

$(100) \rightarrow n^2 \Rightarrow 10,000$

$(500) \rightarrow n^2 \Rightarrow 2,50,000$

100 steps 0.5 msec

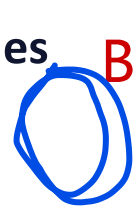
500 ? 2.5ms

Asymptotic Notation

It's a way of recording complexity using math. or mathematical unit to record complexity.

- **Asymptotic Notation is used to describe the running time of an algorithm - how much time an algorithm takes with a given input, n.**

- **Algorithmic Common Runtimes**
- fastest to slowest are:
- constant: Θ(1)
- logarithmic: Θ(log N)
- linear: Θ(N)
- polynomial: Θ(N^2)
- exponential: Θ(2^N)
- factorial: Θ(N!)



Big Oh:

Only targets worst-case
extremely fast, simple
but less accurate(as only Worst seen).

Theta:

targets average of (worst-case+best-case)
slow as it takes time to do
but very accurate.

Omega:

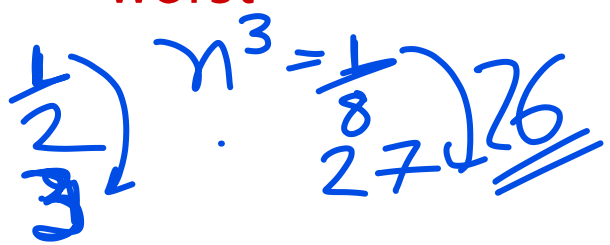
Only targets best-case
extremely simple but hardly used.
(most algorithms behaves similar manner in best case)

Asymptotic Notation

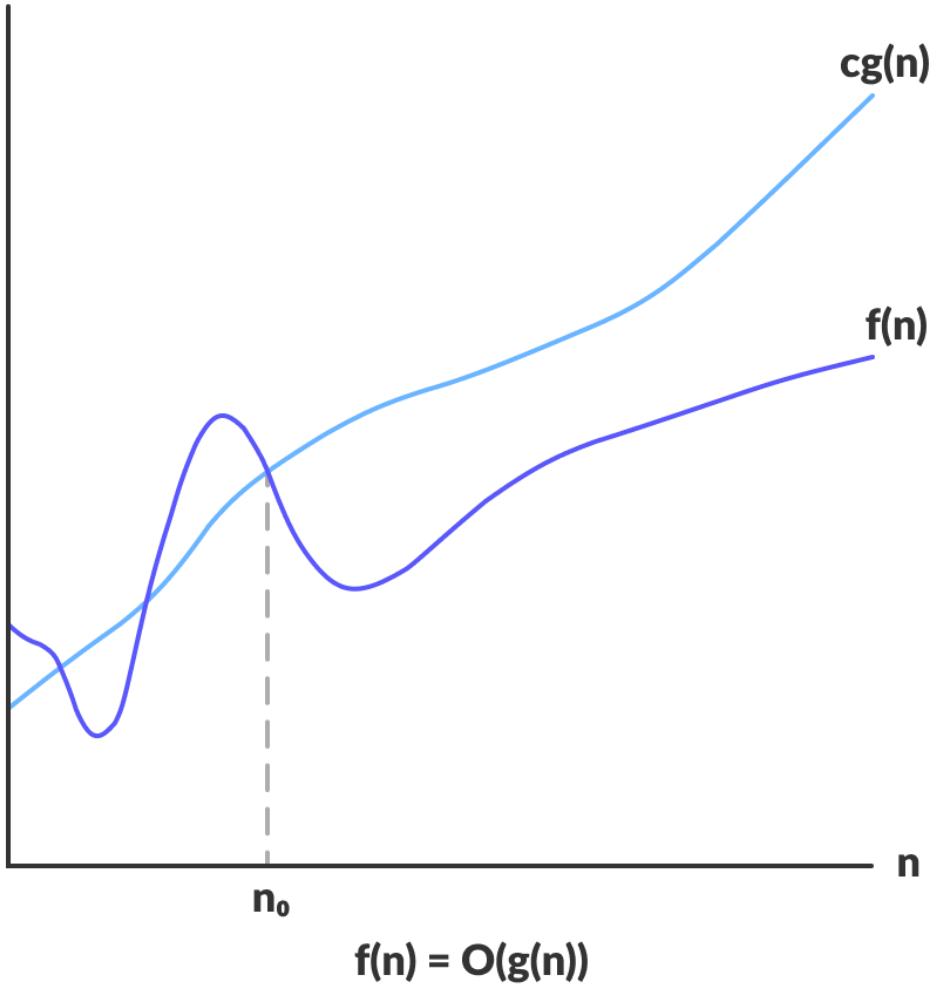
- **Asymptotic Notation is used to describe the running time of an algorithm - how much time an algorithm takes with a given input, n.**
- **Algorithmic Common Runtimes**
- fastest to slowest are:
- constant: Θ(1)
- logarithmic: Θ(log N)
- linear: Θ(N)
- polynomial: Θ(N^2)
- exponential: Θ(2^N)
- factorial: Θ(N!)

f(n):input rate targets number of inputs
g(n):growth rate :targets number of steps to complete alg

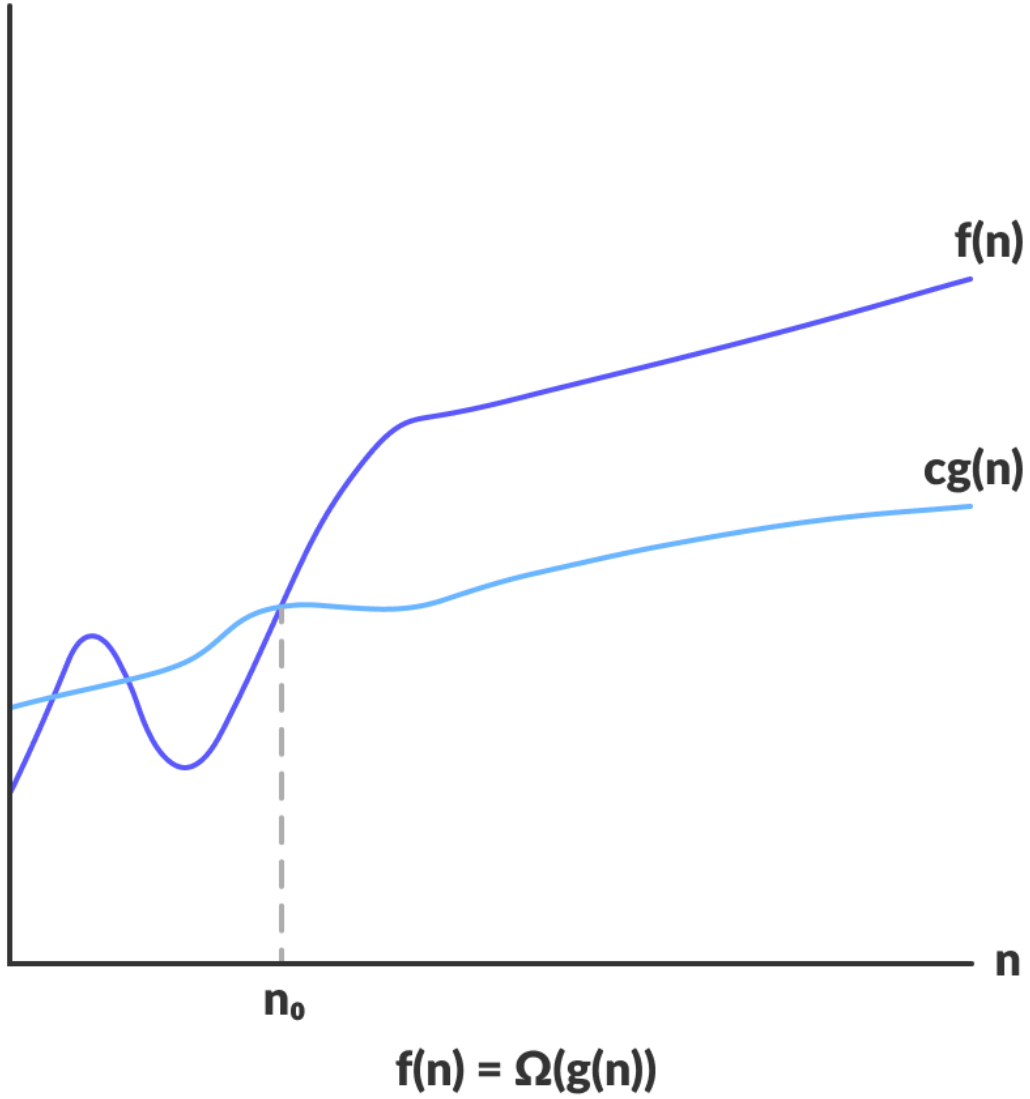
Omega: f(n)>=c.g(n) 100inputs but searching for the 1st
Theta: c1.g(n)<=f(n)<=c2.g(n)
best worst
Big Oh f(n)<=c.g(n)



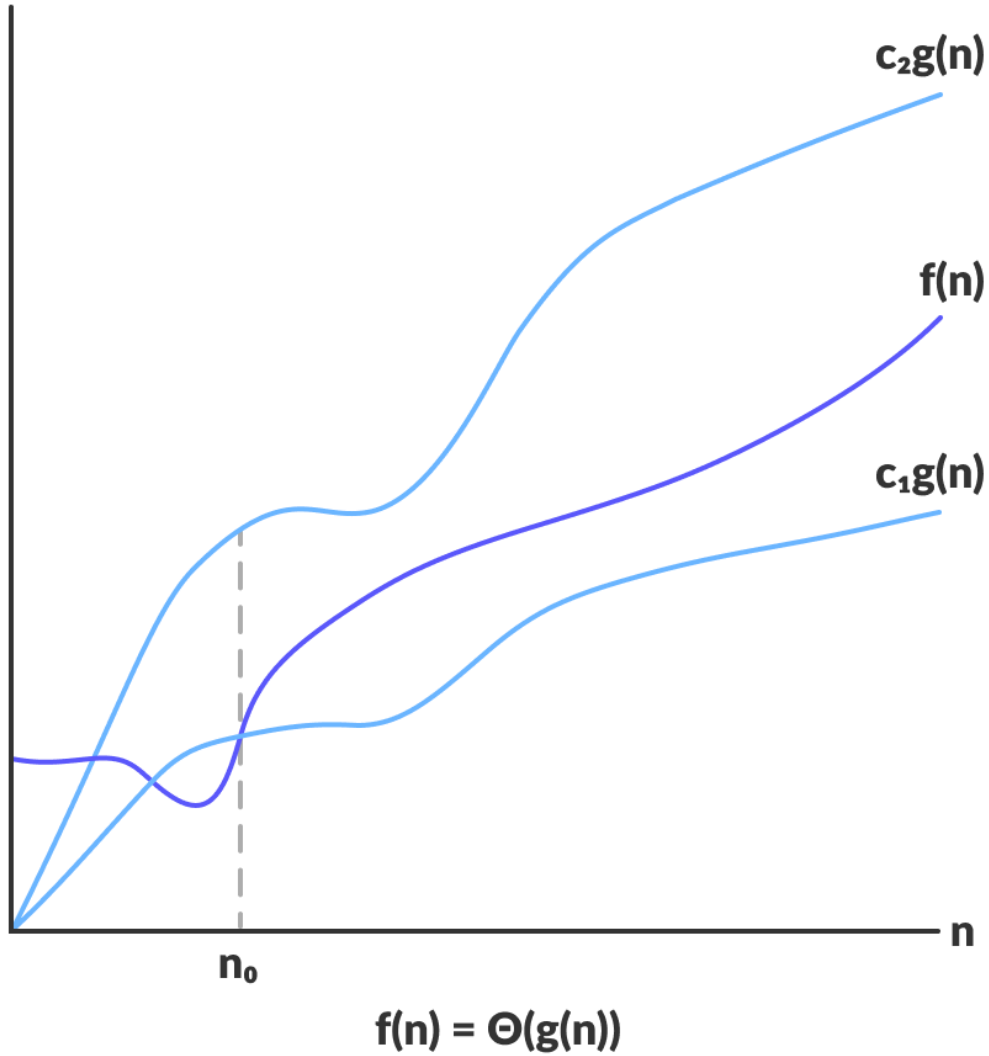
- **Big-O Notation (O-notation)**
- Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm.
-



- **Omega Notation (Ω-notation)**
- Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.



- **Theta Notation (Θ-notation)**
- Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.
-



- **Big-O Notation (O-notation)**
  - **Rules**
    - **Constant has no value** $O(100n) \Rightarrow O(n)$
    - **In addition use max(worst only)** $O(n)+O(n^2)+O(n^3) \Rightarrow O(n^3)$
    - **In nesting multiply and then use addition rule**

$$O(n) * O(n+3)$$

$$O(n^2)+O(3n) \text{ – Rule 1}$$

$$O(n^2)+O(n) \rightarrow \text{Rule 2}$$

$$O(n^2)$$

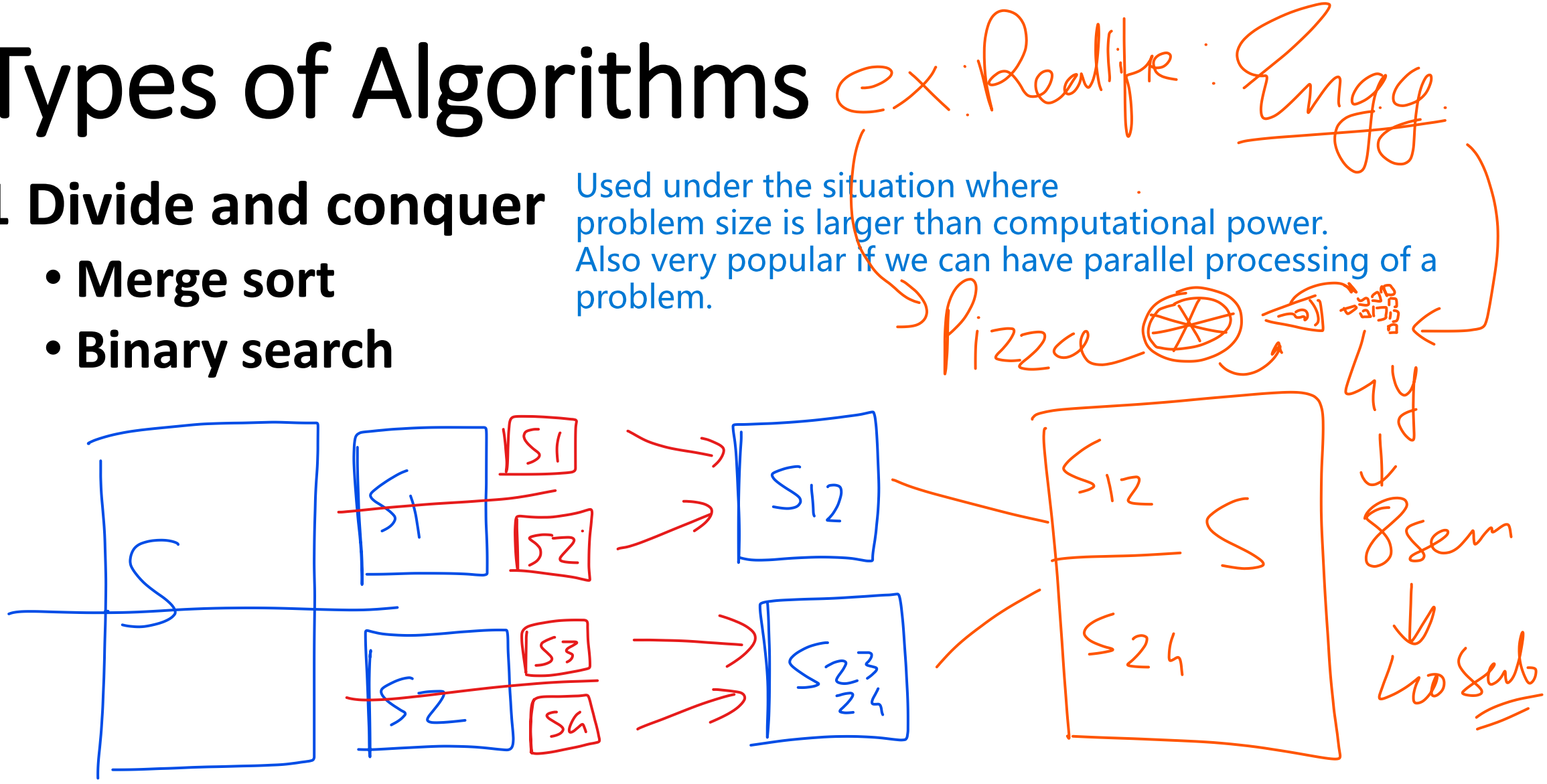
Calculate Complexity using Big OH

Types of Algorithms

1 Divide and conquer

- **Merge sort**
- **Binary search**

Used under the situation where problem size is larger than computational power. Also very popular if we can have parallel processing of a problem.

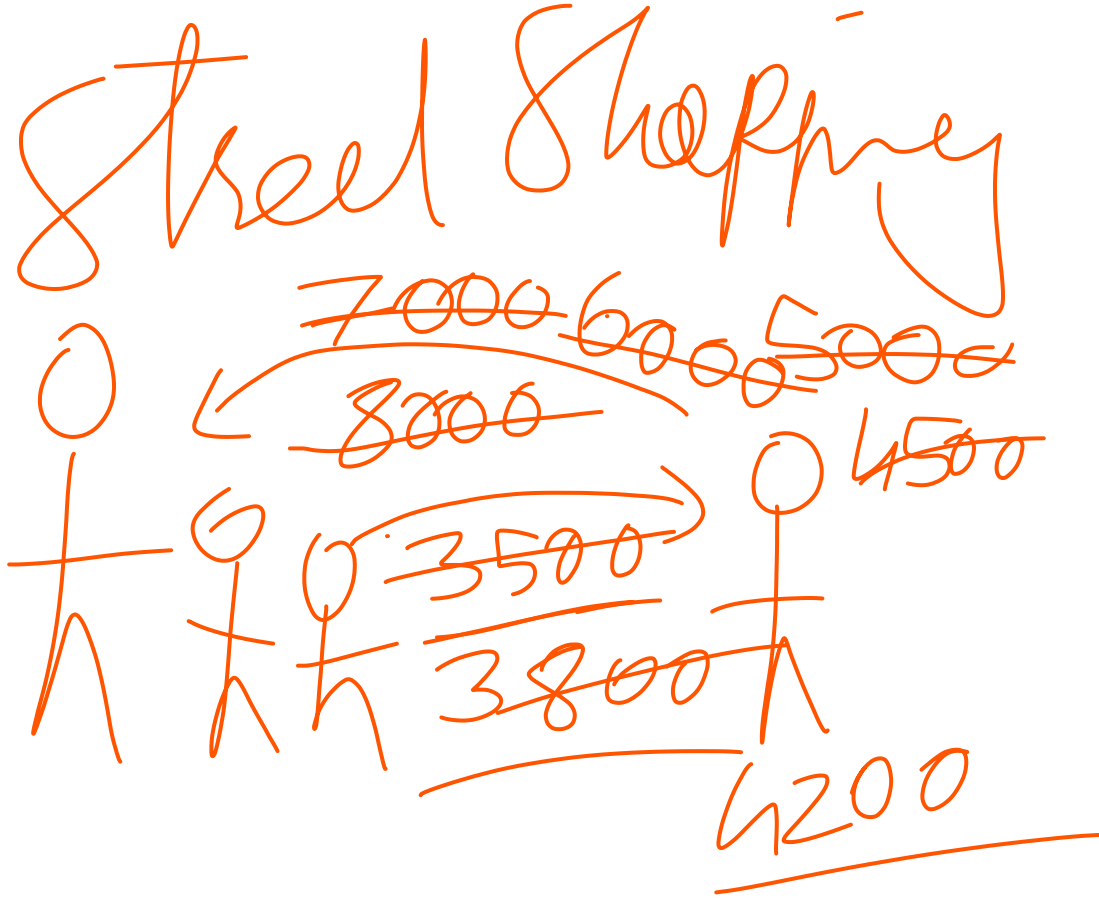


Types of Algorithms

2 Greedy Algorithm

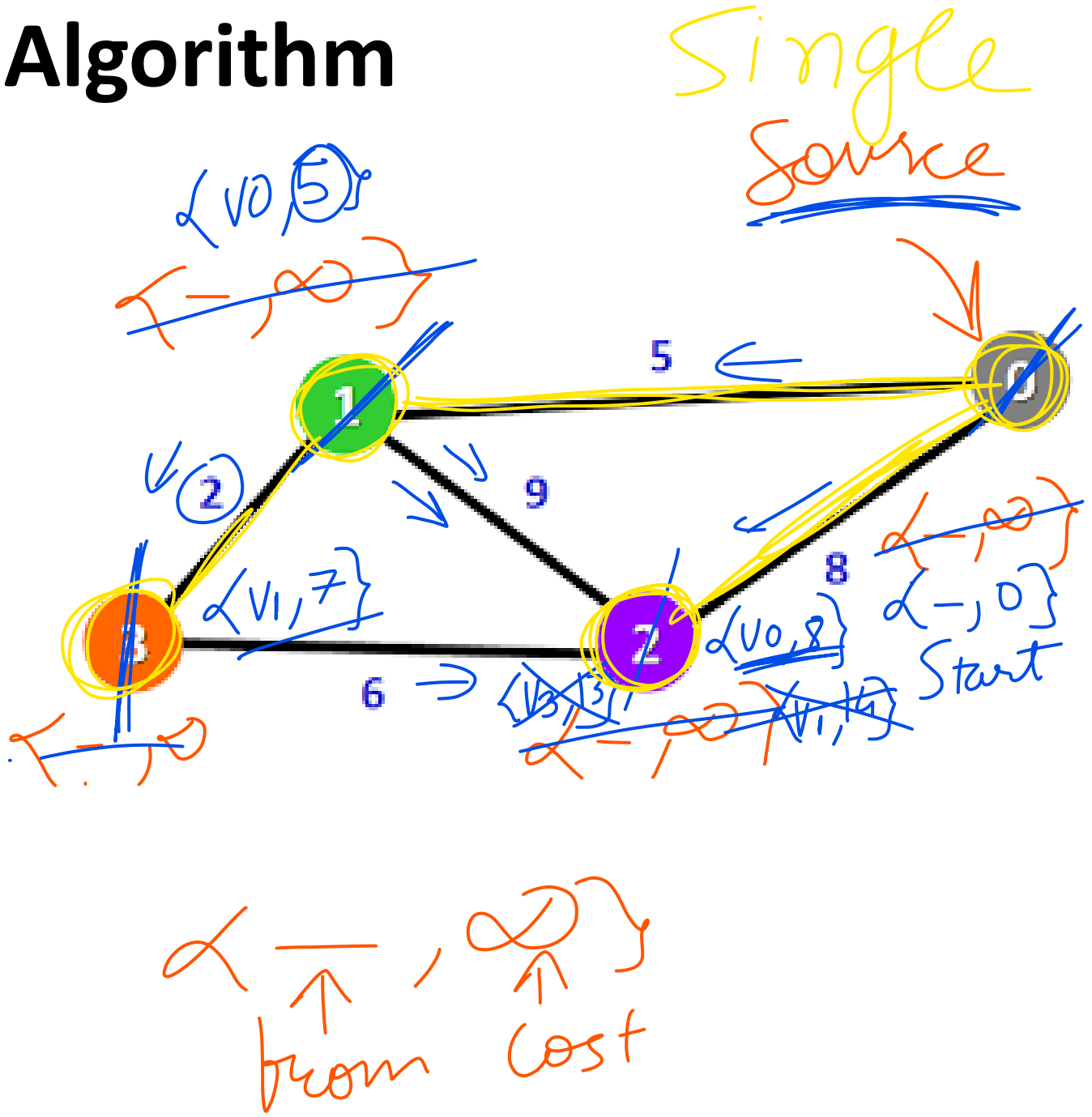
- **Talks About Optimization**
- **Optimization**
  - **Minimization-Cost**
  - **Maximization-Profit**

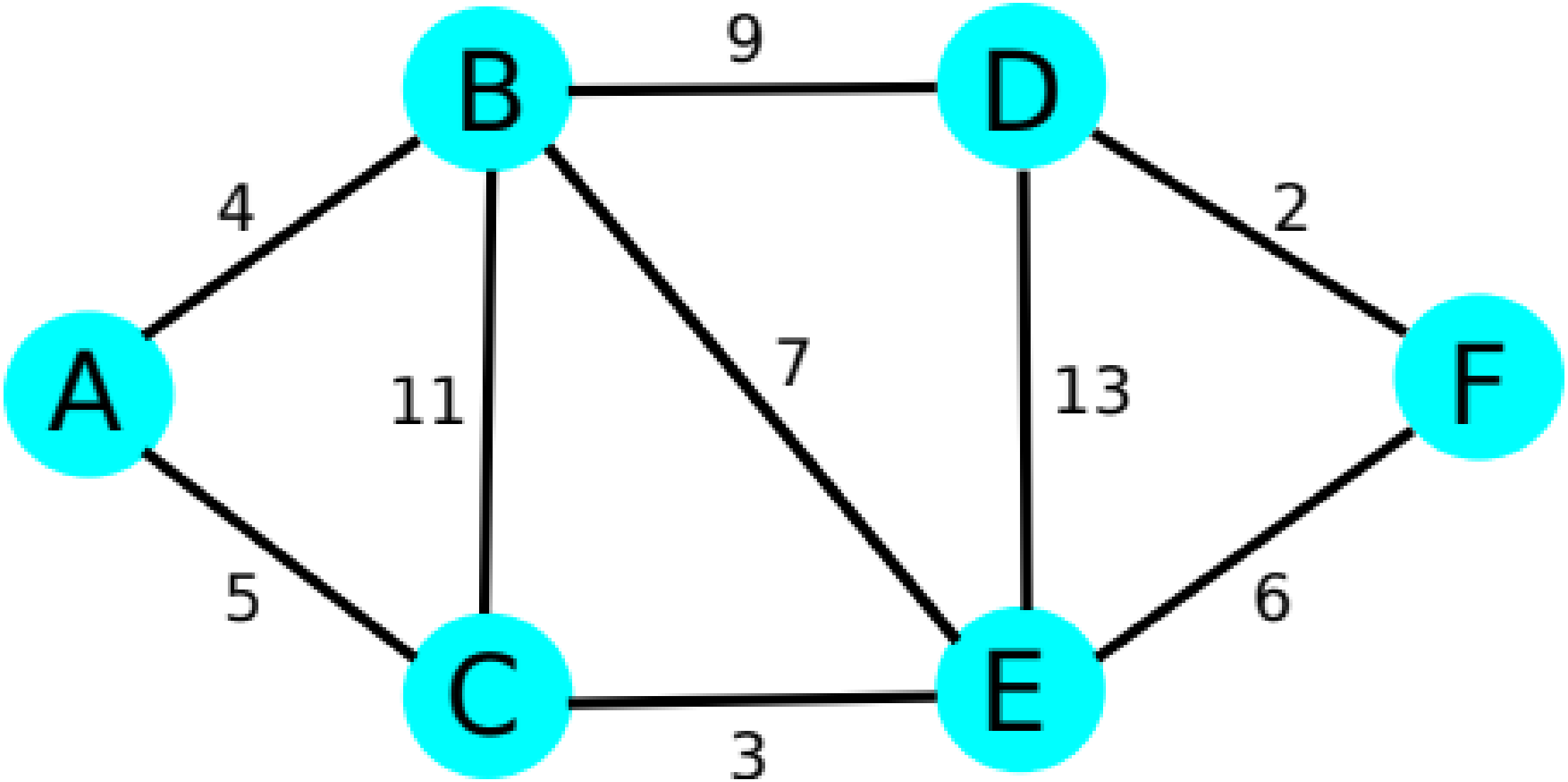
Pay
Take

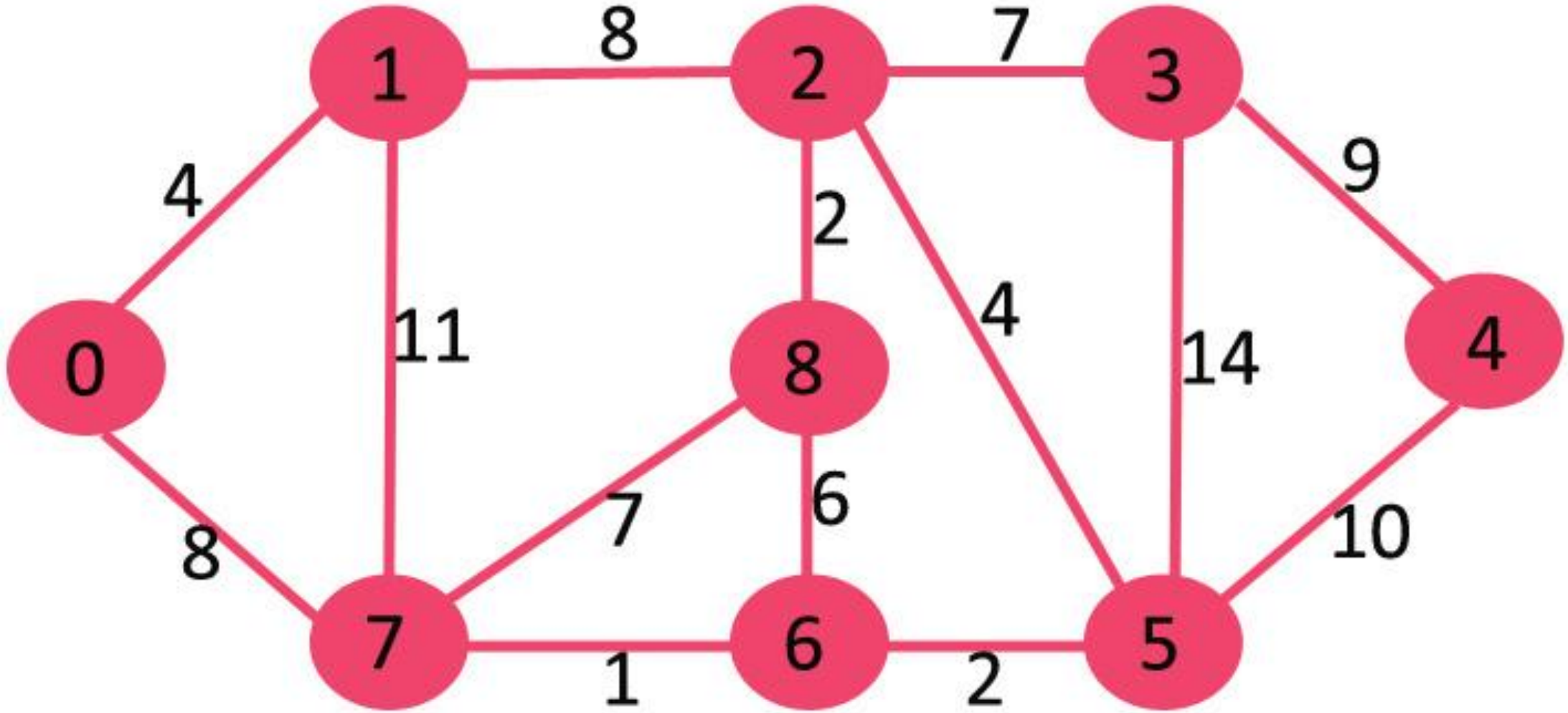


Dijkstra's Shortest Path Algorithm

- Assign a distance value to all vertices in the input graph. Initialize all distance values as **INFINITE**. Assign the distance value as 0 for the source vertex so that it is picked first.
- While **sol** doesn't include all vertices
  - Pick a vertex **u** which is not there in **sol** and has a minimum distance value.
  - Include u to **sol**.
  - Then update distance value of all adjacent vertices of u.
    - To update the distance values, iterate through all adjacent vertices.
    - For every adjacent vertex v, if the sum of the distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.




A
B
C
D
E
F
4
9
2
11
7
13
5
6
3

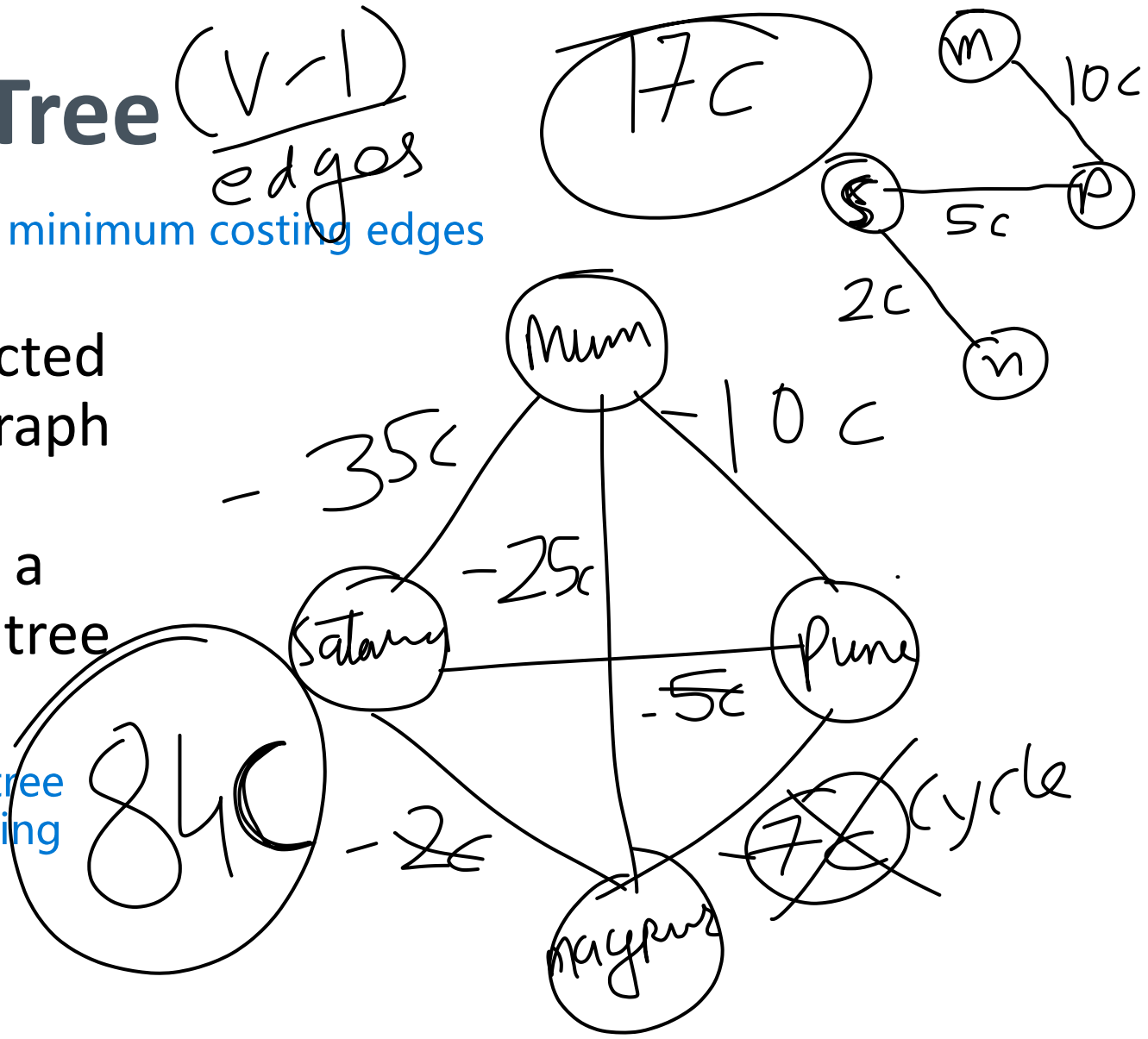

0
1
2
3
4
5
6
7
8
4
8
11
8
2
7
4
9
14
7
6
10
1
2

Minimum Spanning Tree

In this input is a directed graph,
output is a cyclic graph which is a tree with minimum costing edges which doesn't form any cycle.

- Given an undirected and connected graph , a spanning tree of the graph is a tree that spans (that is, it includes every vertex of) and is a subgraph of (every edge in the tree belongs to)with minimumcost.

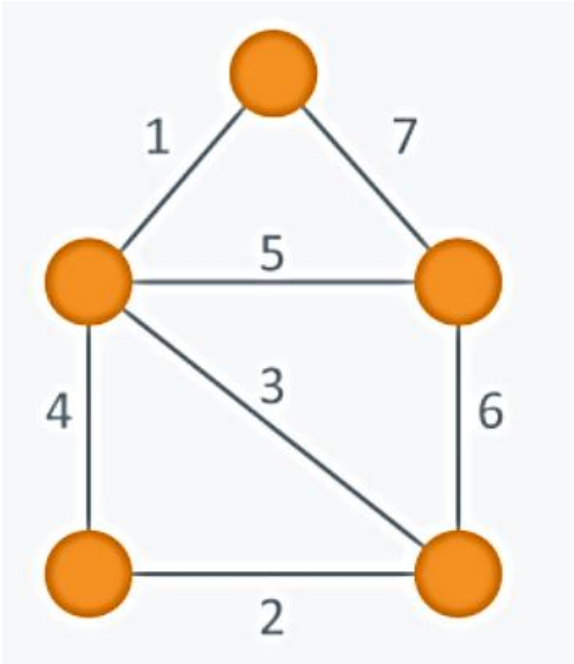
The whole purpose of minimum spanning tree is to minimize connectivity cost by connecting every node to each other without cycles.



It is very commonly used in
road construction, piping, wiring,
power distribution, water distribution,
or any application where one needs to connect multiple without cycle.

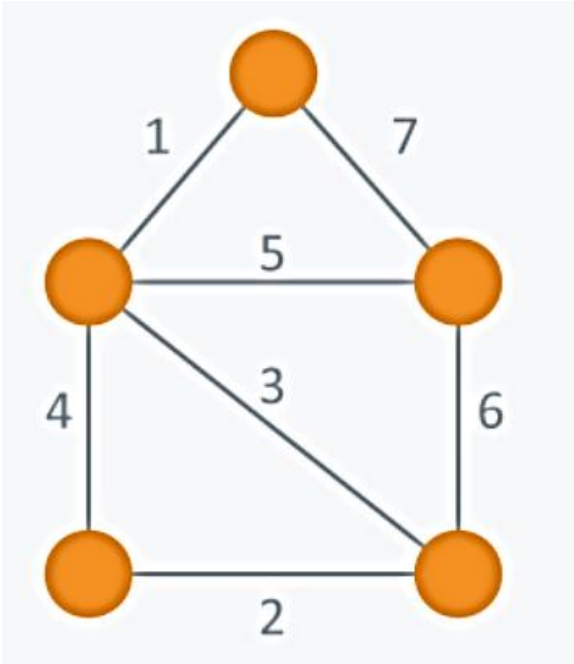
Kruskal’s Algorithm

- **Algorithm Steps:**
- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which doesn't form a cycle , edges which connect only disconnected components.



Prim's Algorithm

- **Algorithm Steps:**
- Start from given source add it to sol tree.
- While all vertices are not in sol tree do
  - Select the cheapest edge between vertex of sol tree and non-sol vertex iff not forming cycle.



Types of Algorithms

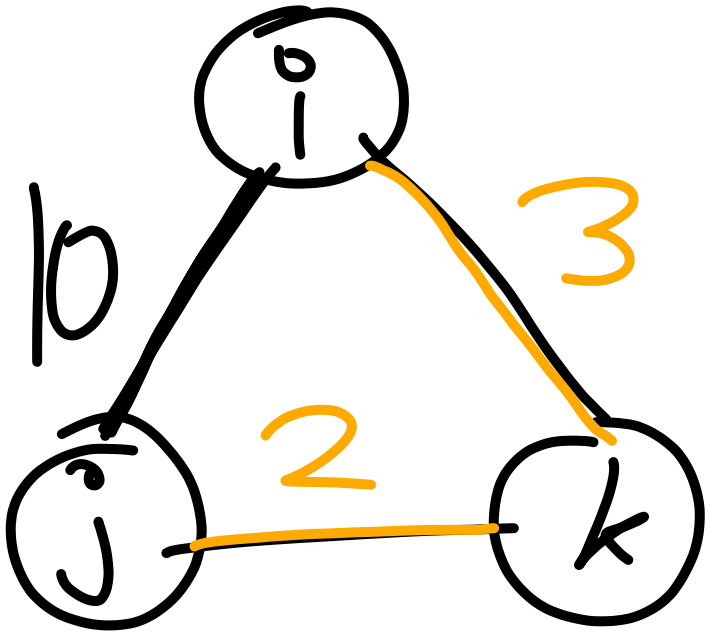
3 Dynamic Programming

- **Break problem in states ,evaluate each state with formula recursively, select best in each step so finally get optimized answer.**
- **Example: All source shortest path(Floyd warshal's)**

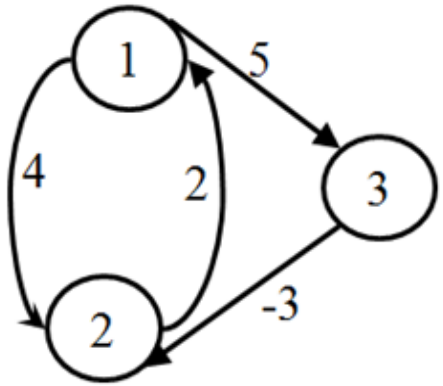
```
Begin
   for k := 0 to n, do
      for i := 0 to n, do
         for j := 0 to n, do
            if cost[i,k] + cost[k,j] < cost[i,j], then
               cost[i,j] := cost[i,k] + cost[k,j]
            done
         done
      done
      display the current cost matrix
End
```

3rd 1 2

All Source Shortest Path



Can be a cheaper indirect path between I to J going v
third vertex K, which is better in terms of cost.



$W = D^0 =$

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

$P =$

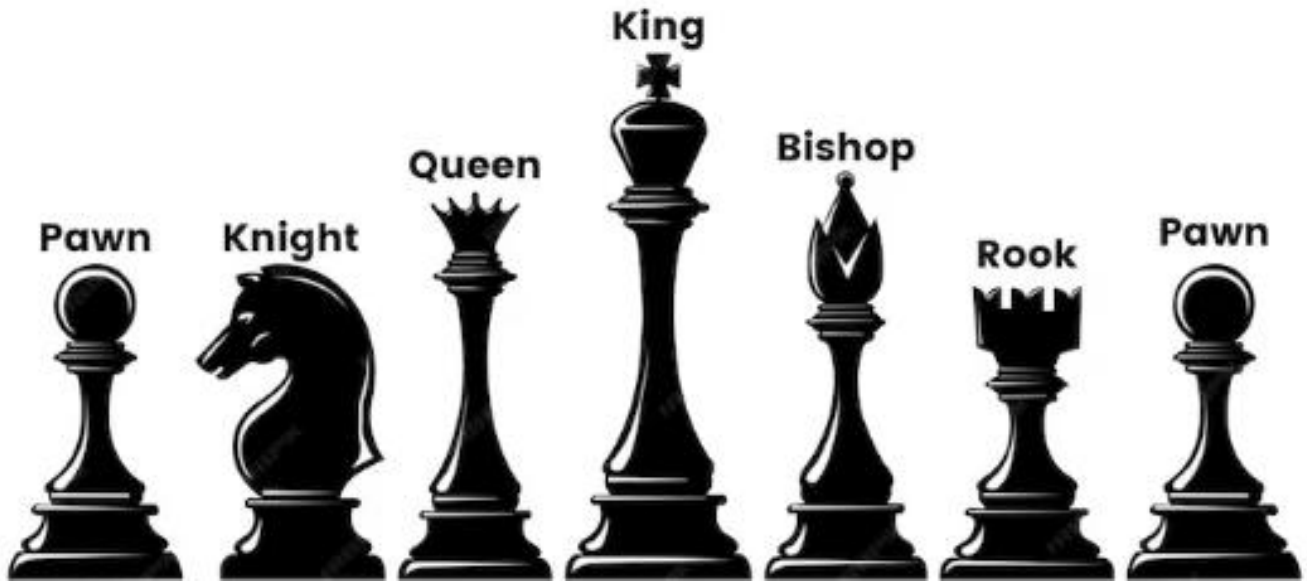
	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Types of Algorithms

4 Back Tracking

- **Only solution needed . (May/May not be optimal)**
- **N-queens problem**

Ś



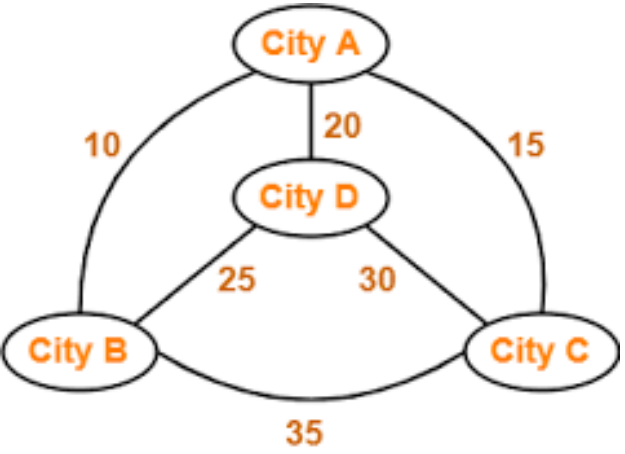
Types of Algorithms

5 Branch n Bound

- All solutions
- State Space tree(decision tree)
- Example: Traveling salesman

option

evaluation of option



Travelling Salesman Problem

Travelling Salesman is an algorithm which starts from a particular city, completes a cycle, and comes back to the same city with the minimum cost of trav

A

AB(10)

AC(15)

AD(20)

ABC (45)

ABD (35)

ACB (50)

ACD (45)

ADB (45)

ADC (50)

ABCD

ABDC

ACBD

ACDB

ADBC

ADCB

ABCDA 95

ABDCA 80

ACBDA 95

ACDBA 80

ADBCA 95

ADCBA 95

Types of Algorithms

- **6 STOCHASTIC ALGORITHM**
  - **Stochastic refers to a variable process where the outcome involves some randomness and has some uncertainty. It is a mathematical term and is closely related to "*randomness*" and "*probabilistic*" and can be contrasted to the idea of "*deterministic*."**
  - **May produce better results**