Dynamic Graph Algorithms

CSC-531

2026

Classical Problems

Dr. Shahbaz Khan
Department of Computer Science and Engineering,
Indian Institute of Technology Roorkee
shahbaz.khan@cs.iitr.ac.in

Analysis

COMPLEXITY?

Why do we need to analyze Worst Case?

Why do we analyze Best Case?

Types

What is complexity of a Problem?

What is complexity of an Algorithm?

What is complexity of a mathematical function?

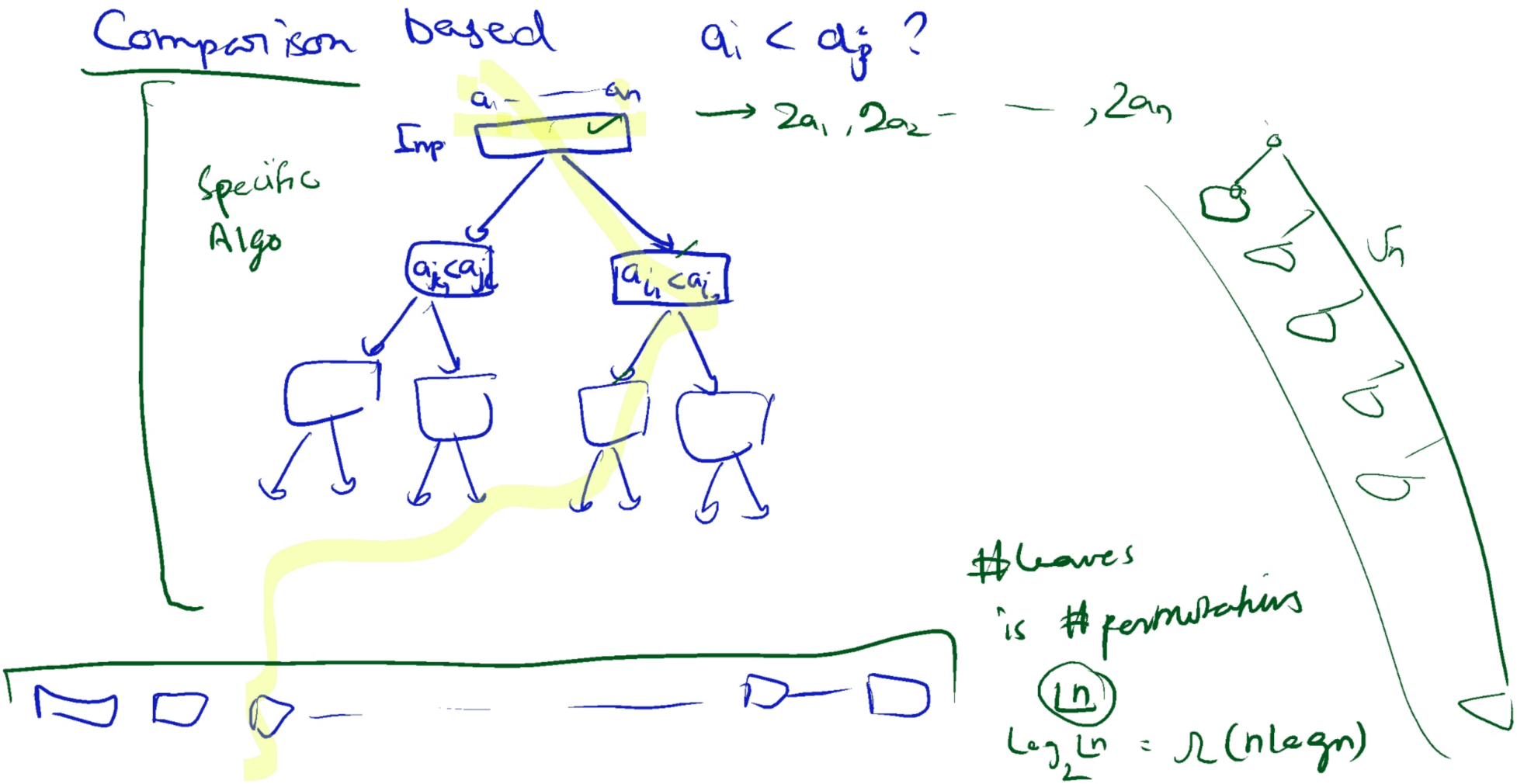
Analysis

- **Problems:** Sorting, Maximum
- **Algorithms:** Quick Sort
- **Functions:** $f(n)=n^3+2n^2+45$

Types
What is complexity of a Problem?
What is complexity of an Algorithm?
What is complexity of a mathematical function?

Comparison based

$a_i < a_j$?



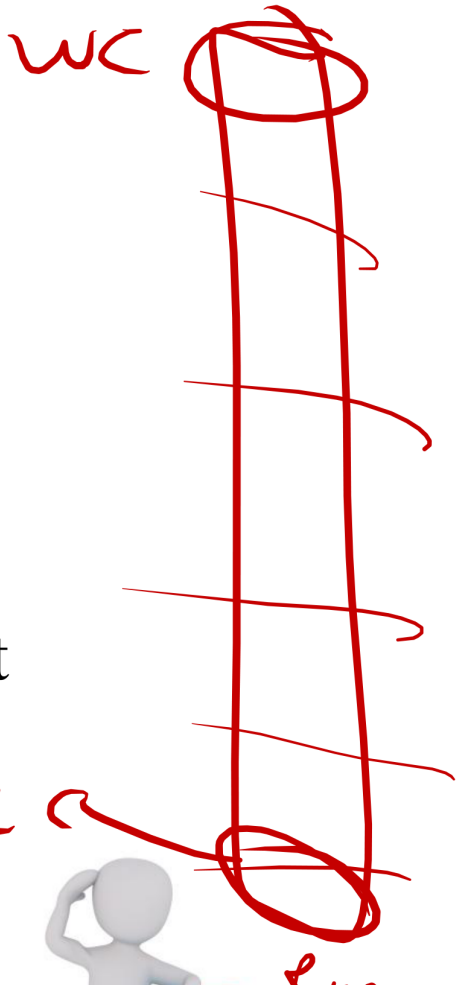
Analysis

- Problems — **P, NP, NP-Hard, EXP**
- Algorithms — **Polynomial, Exponential**
- Functions — **Oh(), Omega(), oh(), omega(), Theta()**

Estimating Algorithms

Algorithm performs differently for different inputs
Possible estimates using Analysis, Examples

- ***Best Case*** is the fastest performance for any input
- ***Average Case*** is average performance for all inputs
- ***Worst Case*** is the slowest performance for any input



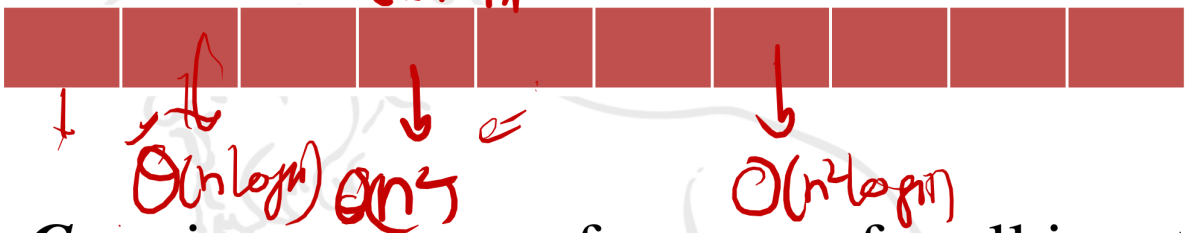
COMPLEXITY?
Why do we need to analyze Worst Case?
Why do we analyze Best Case?

Estimating Algorithms

Algorithm performs differently for different inputs

Possible estimates using Analysis, Examples

- ***Best Case*** is the fastest performance for any input



- ***Average Case*** is average performance for all inputs

- ***Worst Case*** is the slowest performance for any input

Multipop Stack

Push → BC
↘ WC
] $\Theta(1)$

Pop(k) → WC = $\Theta(k)$

$\Theta(n)$

Am. $\Theta(1)$

Best Case

n

Push Pop Push Pop $\Theta(1)$

Am $\Theta(1)$

$\Theta(1)$ $\Theta(n)$

WC

Push → Pop(n)

n-1

2n-2

Average Case → Input

Estimating Algorithms

Algorithm performs differently for different inputs

Possible estimates using Analysis, Examples

- ***Eg. Euler GCD Algo?***

$n = \max(a, b)$

$O(\log n)$

```
gcd(a, b)
{
  if (a > b) return gcd(b, a)
  if (a == 0) return b
  return gcd(b % a, a);
}
```

Example

Fibonacci:

$\Omega(\log n)$

Connectivity Vs Reachability

Connectivity: Undirected Graph G

- *Q(x,y): Are x and y connected?*
  Is there a path between x and y in G
- *Related:* Connected Components

Reachability: Directed Graph G

- *Q(x,y): Is y reachable from x?*
  Is there a path from x to y.
- *Related:* Strongly Connected Components

Reachability is harder than connectivity?

Undirected Graph => Directed Graph

Incremental Reachability

Single Source Reachability:

Given a graph G(V,E) and a source vertex **s**.

Update(x,y): Add edge (x,y) to the graph G.

Query(x): Is the given vertex x ***reachable from*** s in G.

- **Algorithm:**

Incremental Reachability

Single Source Reachability:

Given a graph G(V,E) and a source vertex **s**.

Update(x,y): Add edge (x,y) to the graph G.
Query(x): Is the given vertex x ***reachable from*** s in G.

- **Algorithm:**
  Maintain a R[n] array, 0 if not ***reachable from*** s and 1 if yes

Query(y):

Update(x,y):

Incremental Reachability

Single Source Reachability:

Given a graph G(V,E) and a source vertex **s**.

Update(x,y): Add edge (x,y) to the graph G.
Query(x): Is the given vertex x ***reachable from*** s in G.

- **Algorithm:**
  Maintain a R[n] array, 0 if not ***reachable from*** s and 1 if yes

```
Query(y): Report R[y]

Update(x,y):   Process(x,y)

Process(x,y):  If(R[x]==1 && R[y]==0)
                   R[y]=1;
                   forAll( (y,z) in E)
                         Process(y,z)
```

O(1) Time

Update Time?

Incremental Reachability

Single Source Reachability:

Given a graph G(V,E) and a source vertex **s**.

Update(x,y): Add edge (x,y) to the graph G.
Query(x): Is the given vertex x ***reachable from*** s in G.

- **Algorithm:**
  Maintain a R[n] array, 0 if not ***reachable from*** s and 1 if yes

Query(y): Report R[y]

Update(x,y): Process(x,y)

Process(x,y):
```
If(R[x]==1 && R[y]==0)
    R[y]=1;
    forAll( (y,z) in E)
        Process(y,z)
```

Update(x,y) takes *O(1) time* or R[x]=1, R[y]=0 makes R[y]=1

And call *Update* on neighbours *deg(y)*
Time: $\sum deg(y) = 2m$

Incremental Reachability

Single Source Reachability:

Given a graph G(V,E) and a source vertex **s**.

Update(x,y): Add edge (x,y) to the graph G.
Query(x): Is the given vertex x ***reachable from*** s in G.

- **Algorithm:**
  Maintain a R[n] array, 0 if not ***reachable from*** s and 1 if yes

```
Query(y): Report R[y]

Update(x,y):   Process(x,y)

Process(x,y):  If(R[x]==1 && R[y]==0)
                    R[y]=1;
                    forAll( (y,z) in E)
                          Process(y,z)
```

O(1) Time

O(m) Worst Case Time

Amortized Time???

Incremental Reachability

Single Source Reachability:

Given a graph G(V,E) and a source vertex **s**.

Update(x,y): Add edge (x,y) to the graph G.
Query(x): Is the given vertex x ***reachable from*** s in G.

- **Algorithm:**
  Maintain a R[n] array, 0 if not ***reachable from*** s and 1 if yes

Query(y): Report R[y]

Update(x,y): Process(x,y)

Process(x,y):
```
If(R[x]==1 && R[y]==0)
    R[y]=1;
    forAll( (y,z) in E)
        Process(y,z)
```

O(1) Time

O(m) Worst Case Time

Amortized Time:
Total Time= O(m)
Each vertex R[y]=1 once
Amortized time = O(1)

Incremental Reachability

Single Source Reachability:

Given a graph G(V,E) and a source vertex **s**.

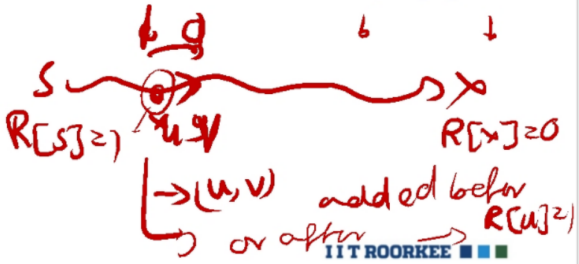
Update(x,y): Add edge (x,y) to the graph G.
Query(x): Is the given vertex x ***reachable from*** s in G.

- **Algorithm:**
  Maintain a R[n] array, 0 if not ***reachable from*** s and 1 if yes

Query(y): Report R[y]
Update(x,y): Process(x,y)

Process(x,y):
```
If(R[x]==1 && R[y]==0)
    R[y]=1;
    forAll( (y,z) in E)
        Process(y,z)
```

Correctness:
=> If **R[x]=1, x** is connected to **s:**
<= If **x** is connected to **s, R[x]=1:**



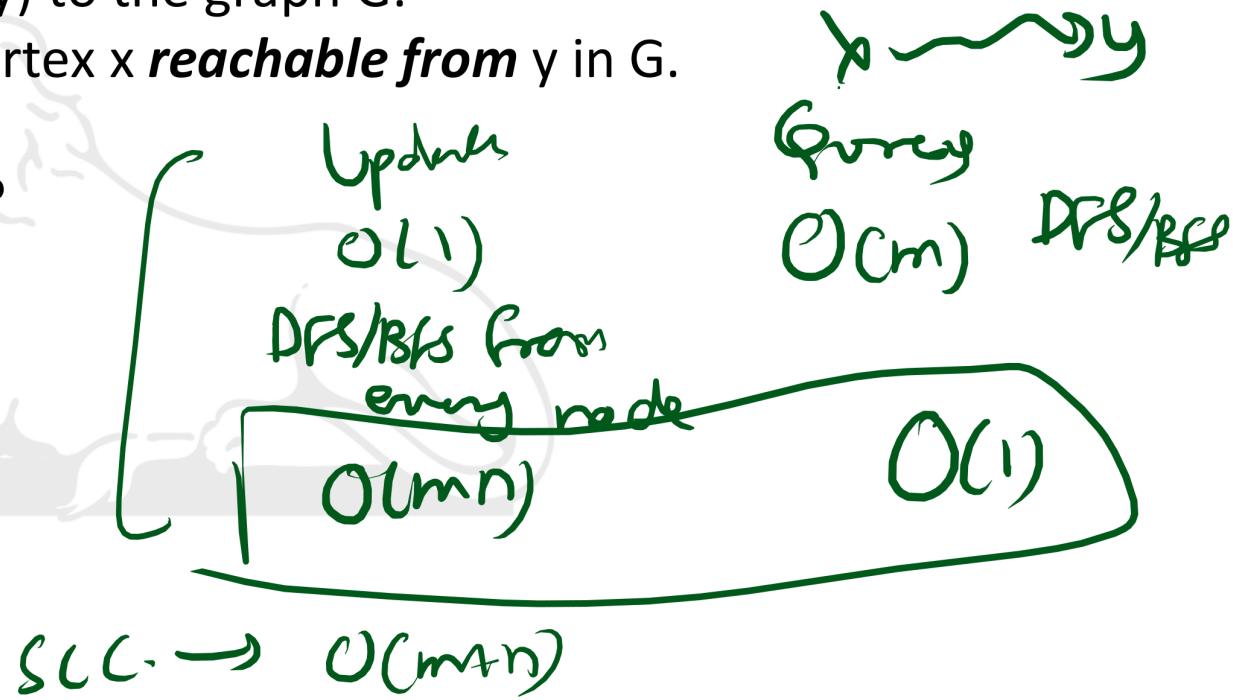
Incremental Reachability

All Pairs Reachability:

Given a graph G(V,E).

Update(x,y): Add edge (x,y) to the graph G.
Query(x,y): Is the given vertex x ***reachable from*** y in G.

- **Algorithm:** **TRIVIAL?**



Incremental Reachability

All Pairs Reachability:

Given a graph G(V,E)

Update(x,y): Add edge (x,y) to the graph G.
Query(x,y): Is the given vertex x ***reachable from*** y in G.

- **Algorithm:** Maintain Single Source Reachability from each vertex

Update: O(n) amortized,
O(mn) worst-case
Query: O(1)

R[x][y] x ⇝ y?

Paper Reference

Italiano 86

Incremental Connectivity

Single Source Connectivity:

Given an undirected graph G(V,E) and a source vertex **s**.

Update(x,y): Add edge (x,y) to the graph G.

Query(x): Is the given vertex x connected to s in G.

Disjoint Set Union???

Query(y): $O(\log^* n)$

Update(x,y): $O(\log^* n)$

Incremental Connectivity

Single Source Connectivity:

Given an undirected graph G(V,E) and a source vertex **s**.

Update(x,y): Add edge (x,y) to the graph G.
Query(x): Is the given vertex x connected to s in G.

- **Algorithm:**
  Maintain a C[n] array, 0 if not connected to s and 1 if yes

Query(y):

Update(x,y):

Incremental Connectivity

All Pairs Connectivity:

Given an undirected graph G(V,E).

Update(x,y): Add edge (x,y) to the graph G.

Query(x,y): Is the given vertex x connected to y in G

Incremental Connectivity

All Pairs Connectivity:

Given an undirected graph G(V,E).

Update(x,y): Add edge (x,y) to the graph G.

Query(x,y): Is the given vertex x connected to y in G

Union Find:	Query	O(log* n)
	Update	O(log* n)

Incremental Connectivity

All Pairs Connectivity:

Given an undirected graph G(V,E).

Update(x,y): Add edge (x,y) to the graph G.

Query(x,y): Is the given vertex x connected to y in G

Union Find:	Query	O(log* n)
	Update	O(log* n)

Trees, Path Compression (Complex Analysis)

Represent Sets By Trees,
Merge Trees Using Ranks, **Gives O(log n)**

Use Path Compression,
Complex Analysis **Gives O(log* n)**

Incremental Connectivity

All Pairs Connectivity:

Given an undirected graph G(V,E).

Update(x,y): Add edge (x,y) to the graph G.

Query(x,y): Is the given vertex x connected to y in G

Union Find:	Query	O(log* n)
	Update	O(log* n)

Trees, Path Compression (Complex Analysis)

SIMPLE ALGORITHM

Use Lists and Arrays to get a simple algorithm

Worst Case Time O(n)

Amortized time O(log n)