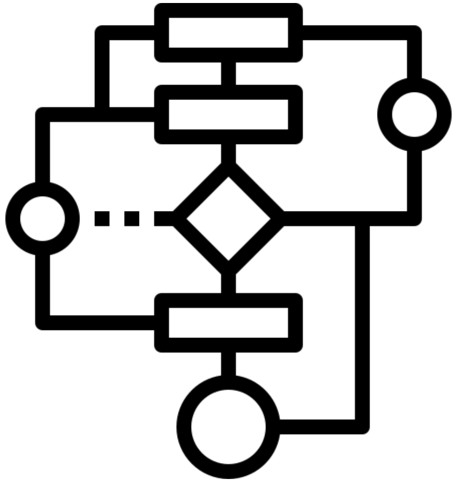


CS 310: Algorithms

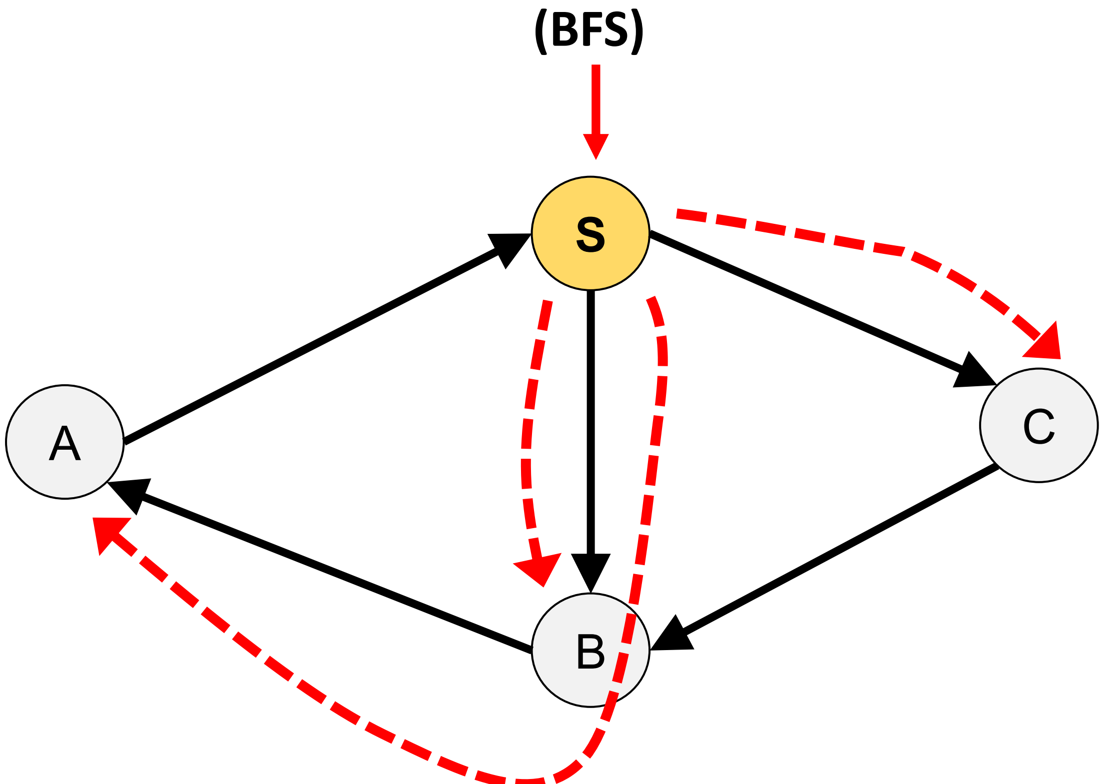
Lecture 12

Instructor: Naveed Anwar Bhatti

Quiz 3 - Solutions

Question 1:

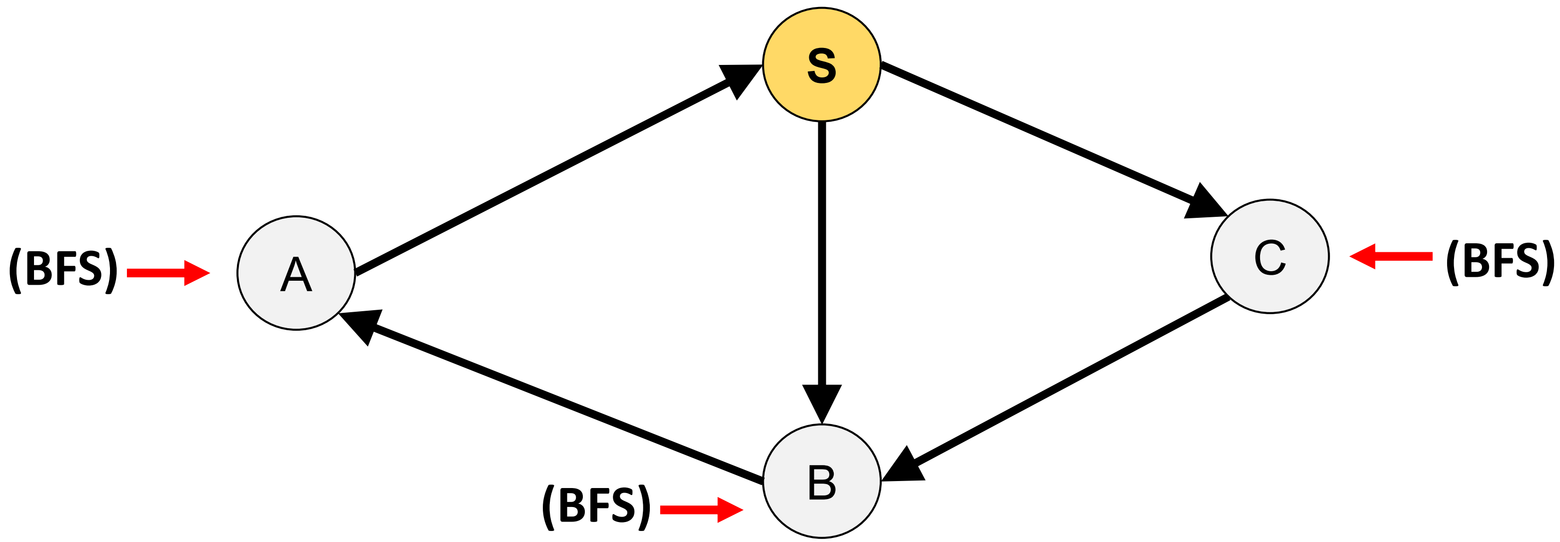
Design an algorithm (pseudo or just steps) to check if a directed graph is strongly connected. Your algorithm **should only use Breadth-First Search (BFS)** and **should not calculate the reverse** (or transpose) of the graph G. After describing your algorithm, state its time complexity.



Quiz 3 - Solutions

Question 1:

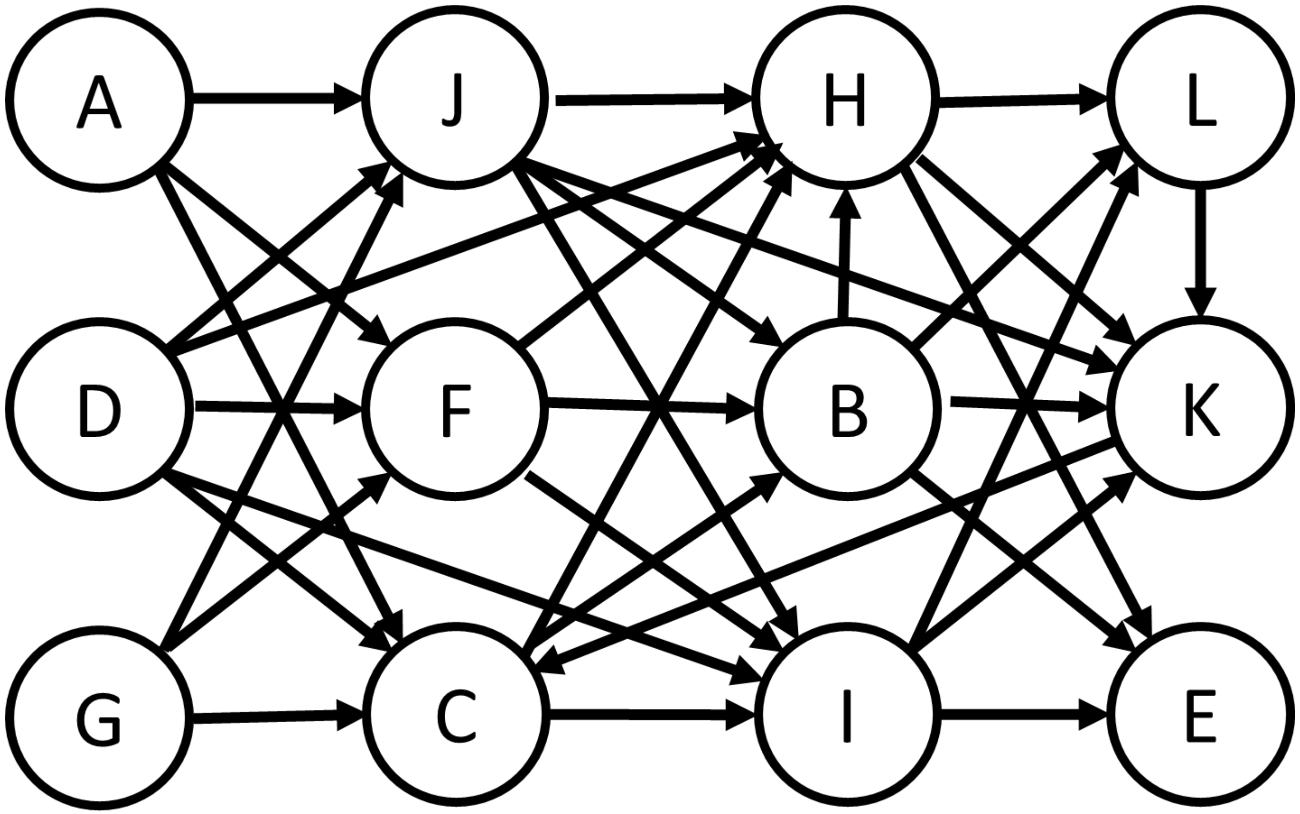
Design an algorithm (pseudo or just steps) to check if a directed graph is strongly connected. Your algorithm **should only use Breadth-First Search (BFS)** and **should not calculate the reverse** (or transpose) of the graph G. After describing your algorithm, state its time complexity.



Quiz 3 - Solutions

Question 2:

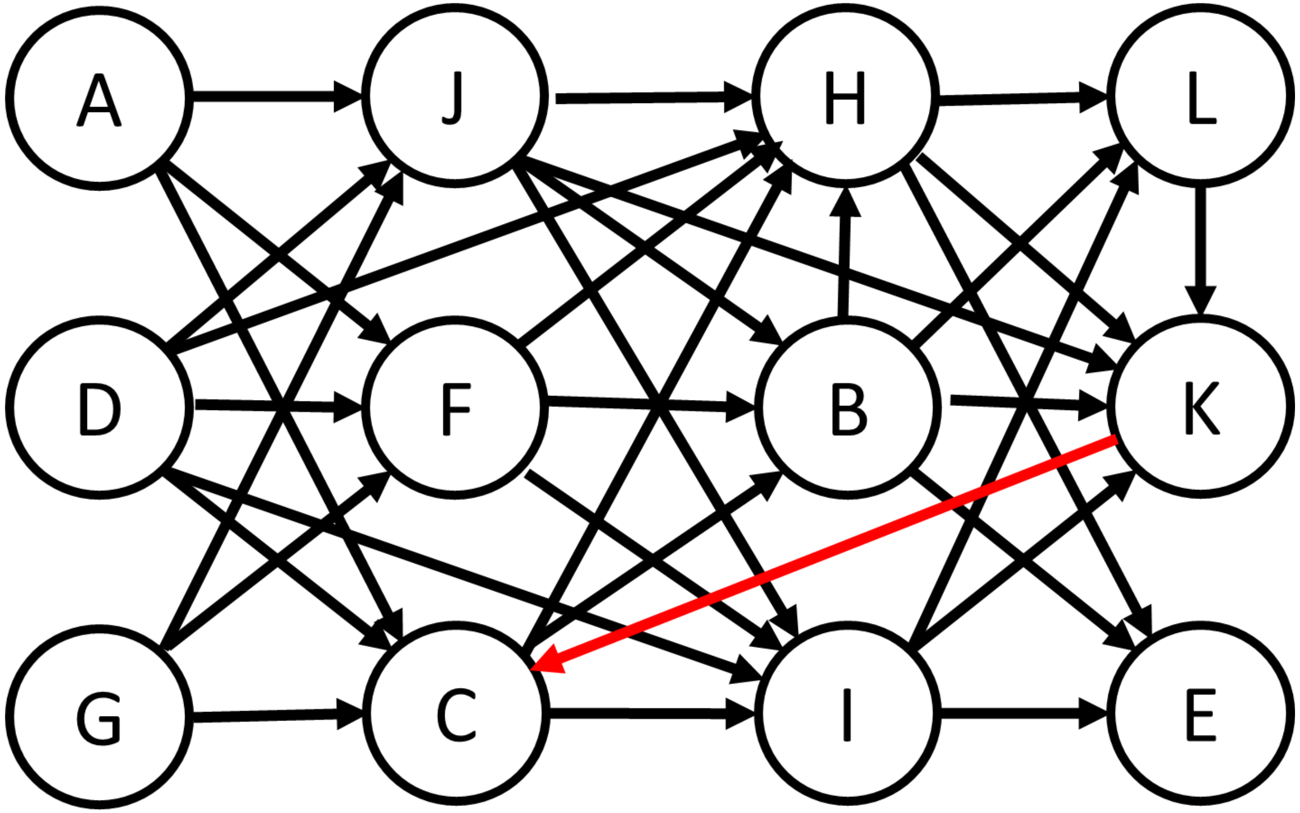
Find topological order.



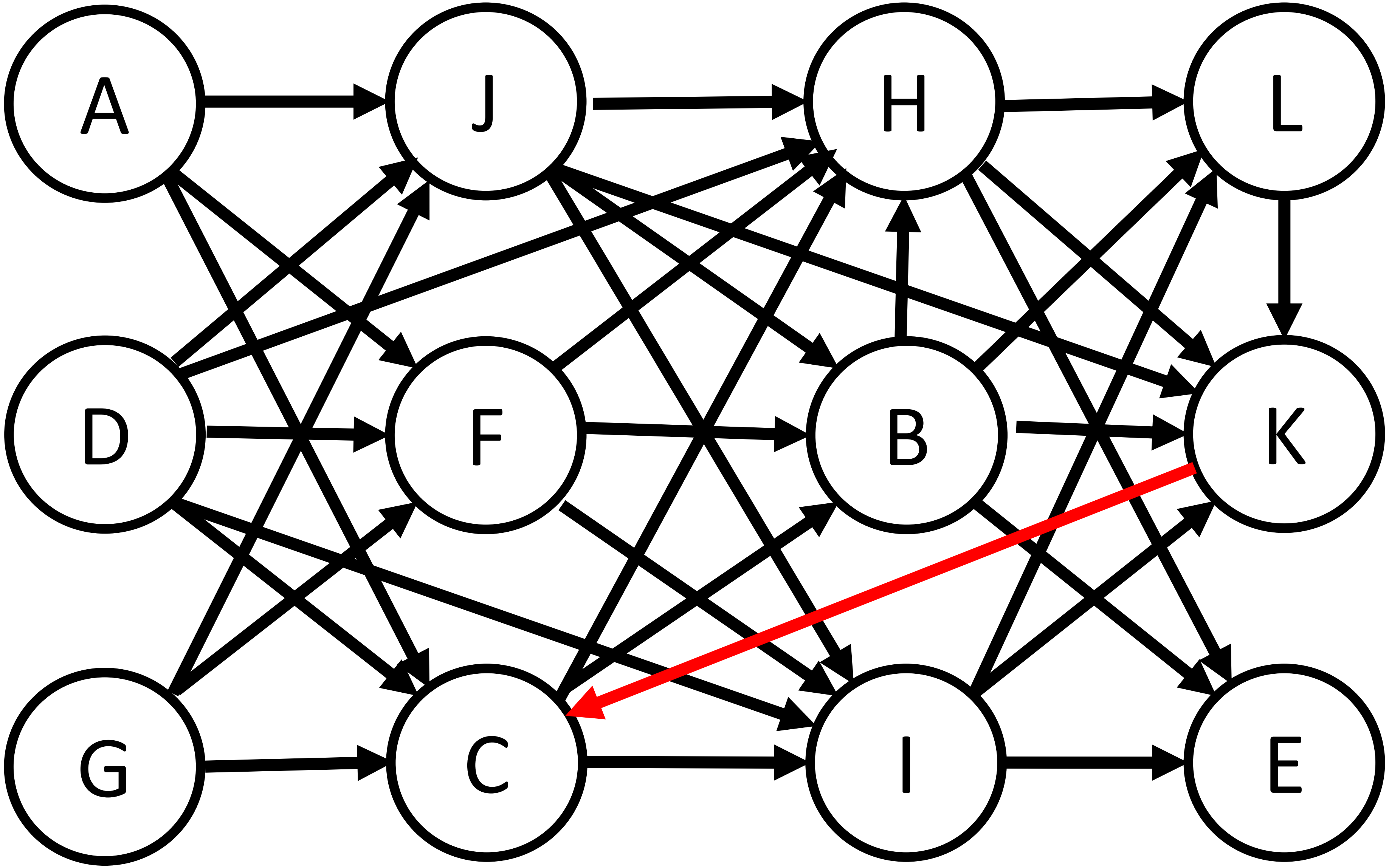
Quiz 3 - Solutions

Question 2:

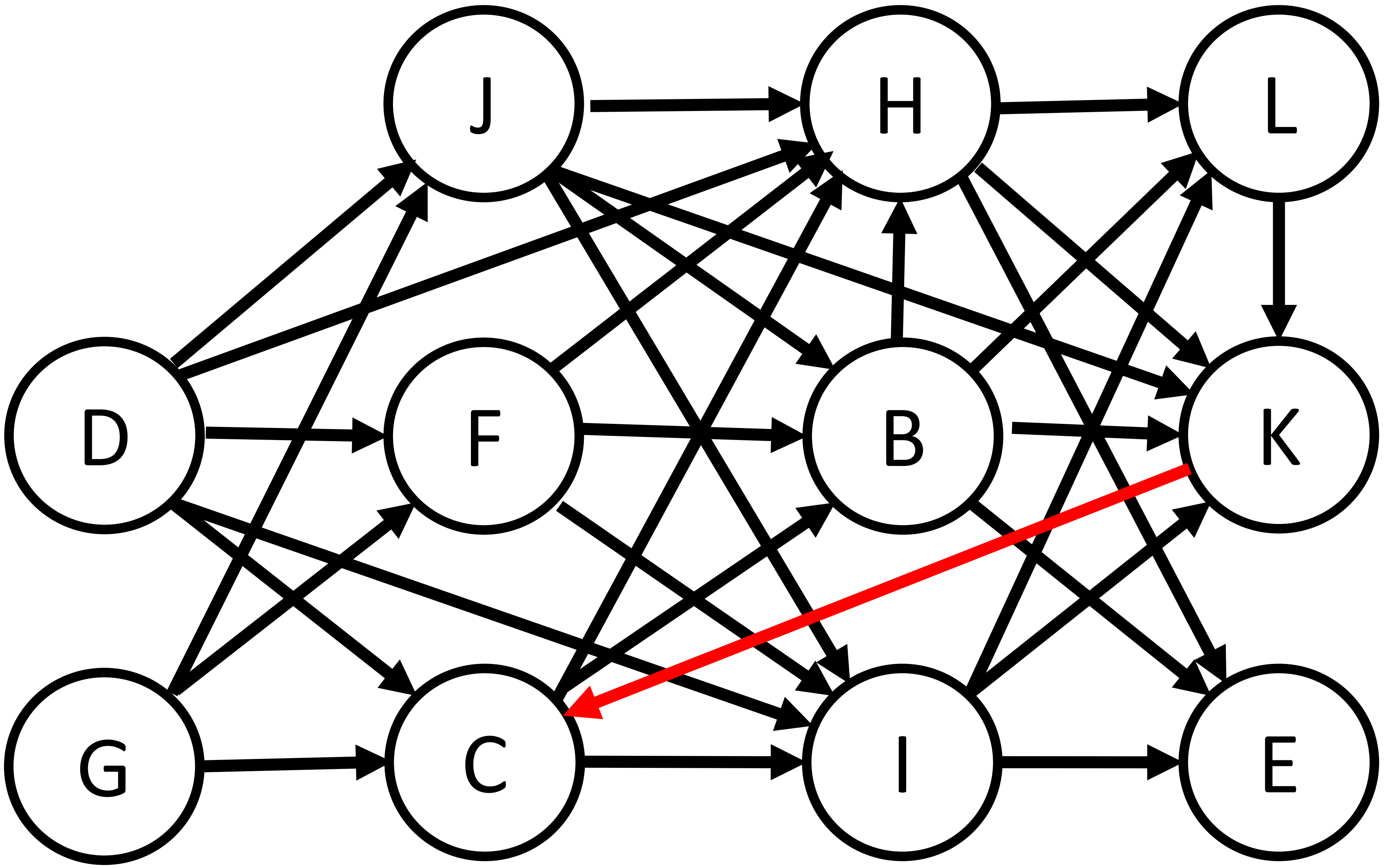
Find topological order.



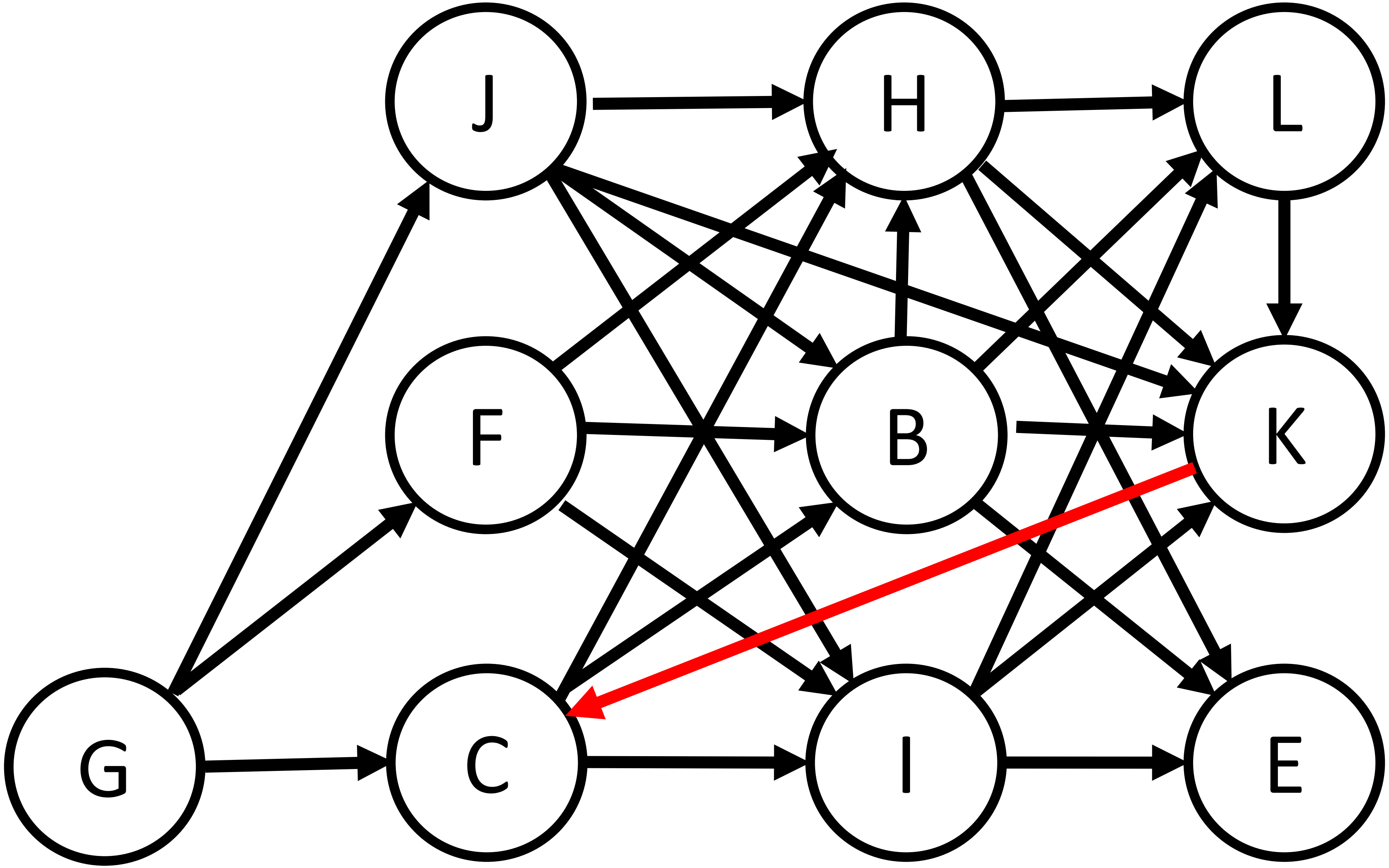
Quiz 3 - Solutions



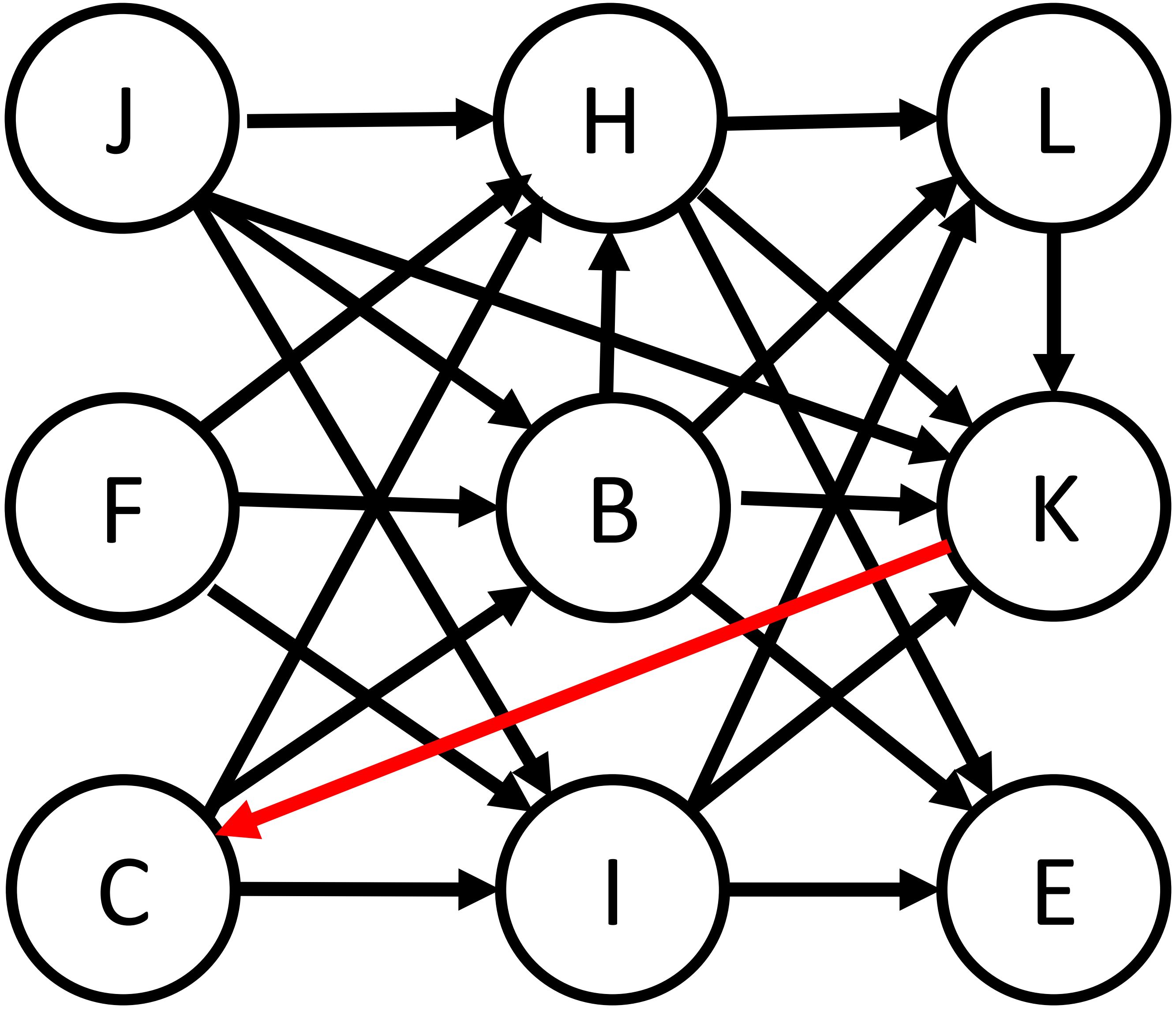
Quiz 3 - Solutions



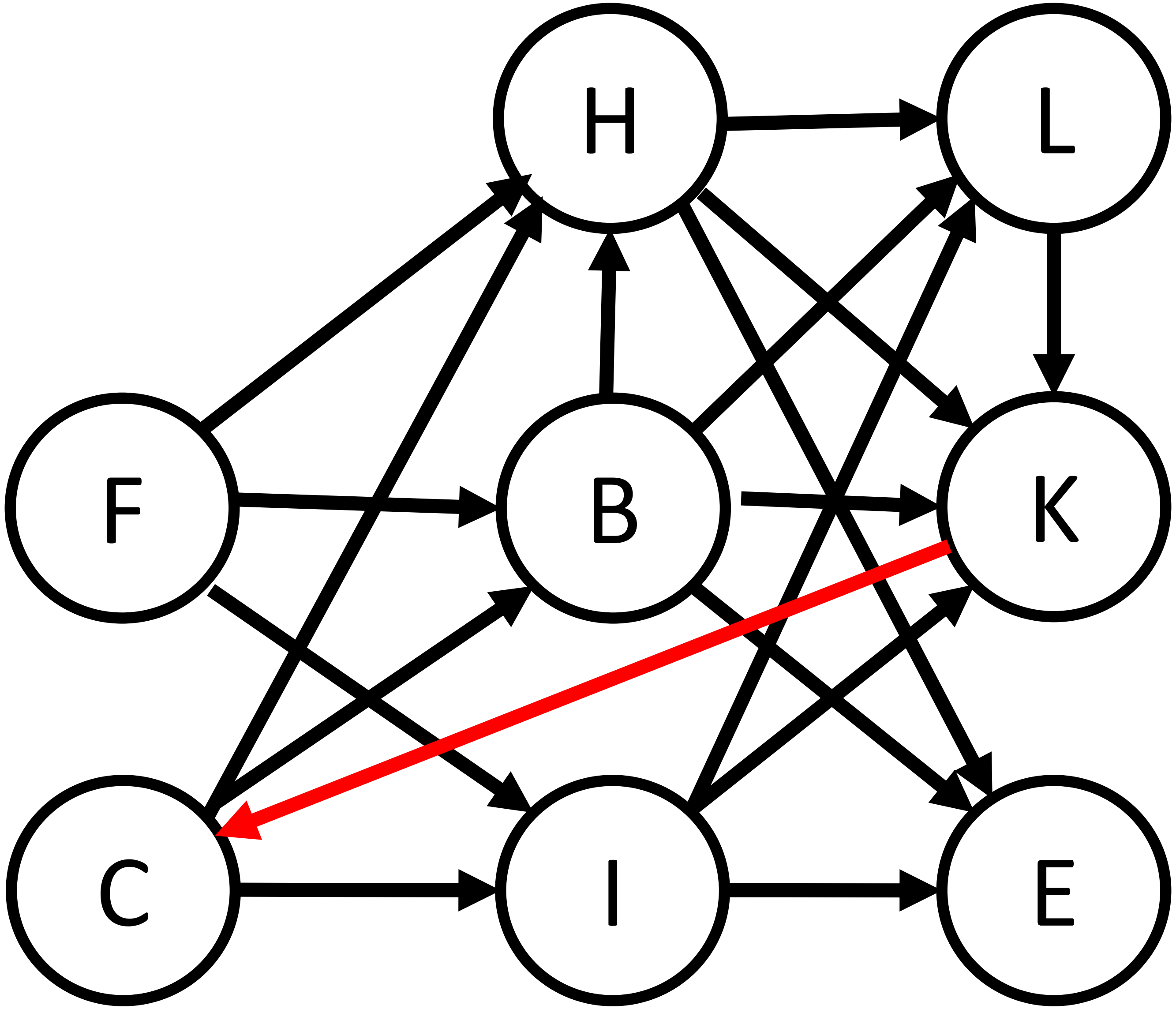
Quiz 3 - Solutions



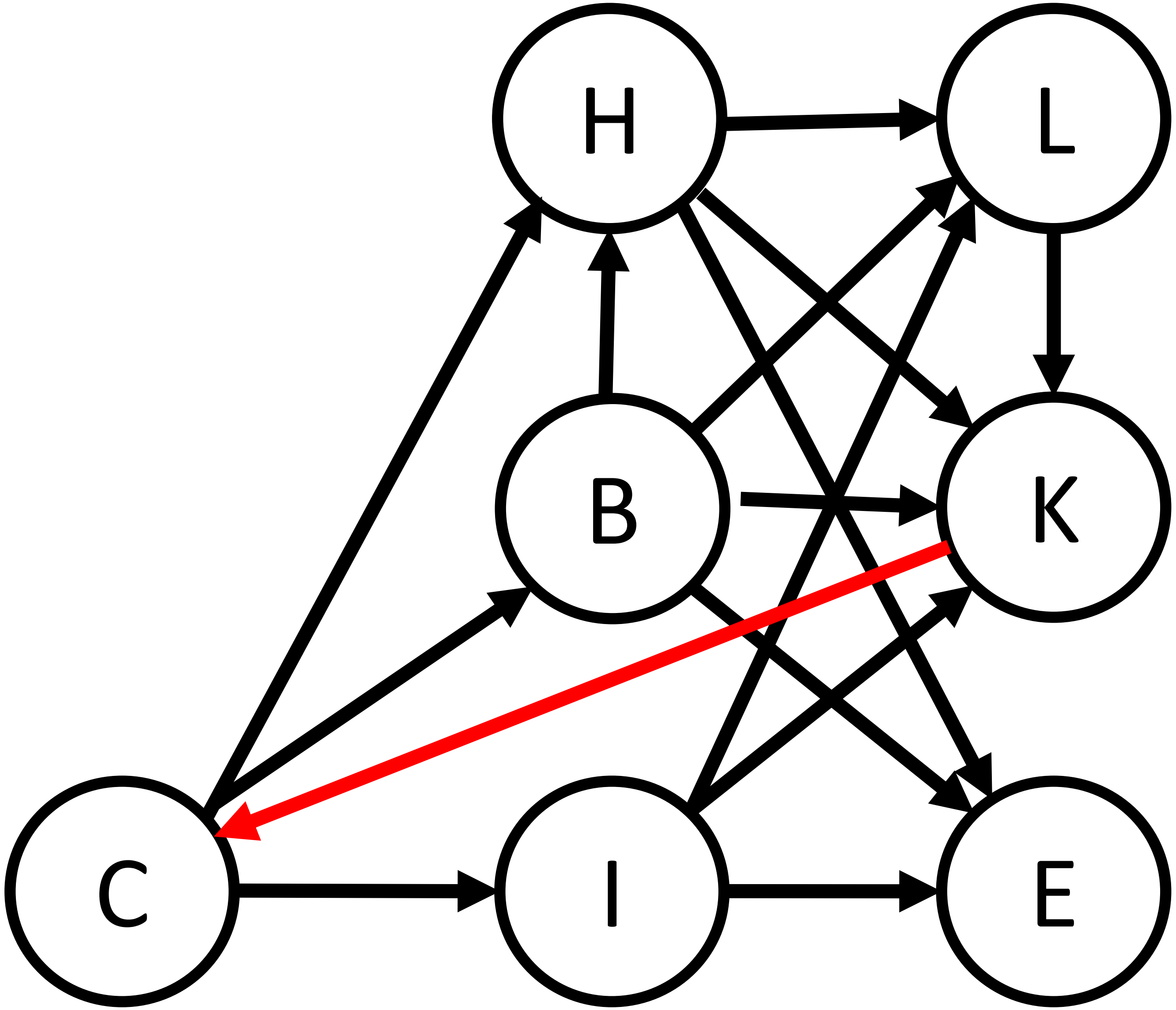
Quiz 3 - Solutions



Quiz 3 - Solutions



Quiz 3 - Solutions



Quiz 3 - Solutions

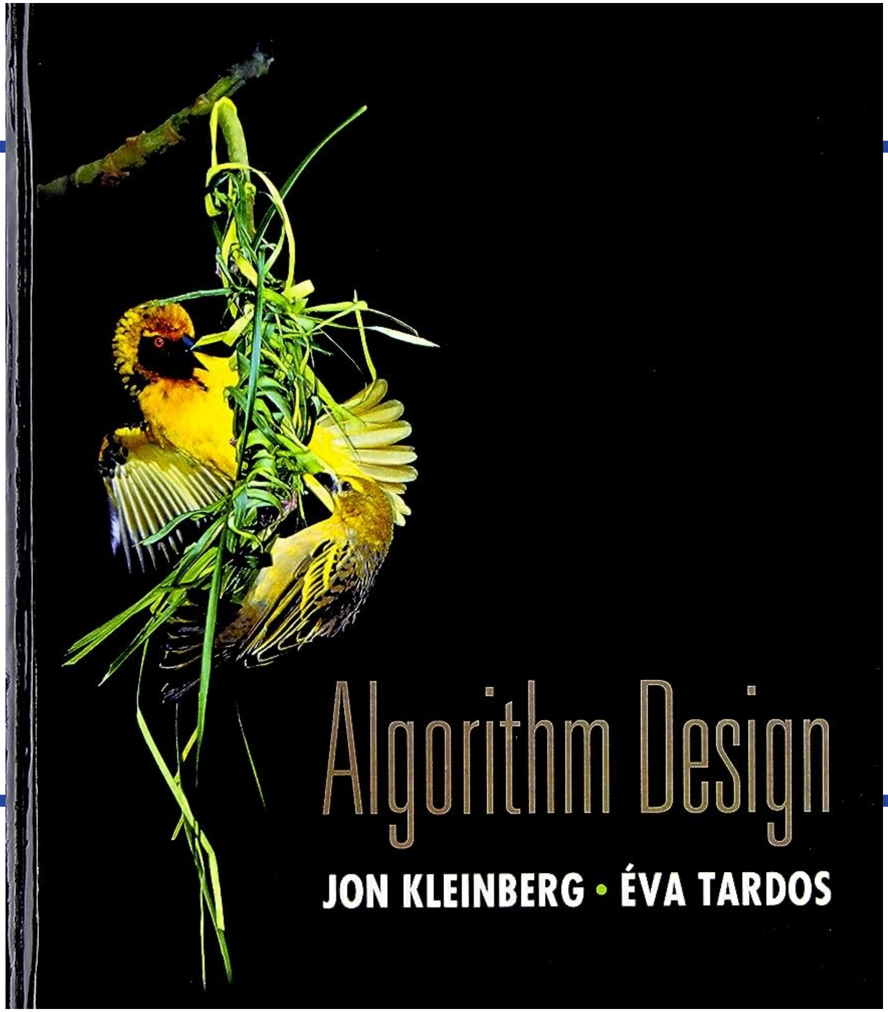
Question 3:

Proof through contradiction that Directed Acyclic Graph (DAG) cannot be a strongly connected graph.

- Assume for the sake of contradiction that there exists a DAG which is also strongly connected.
- Let's take any two vertices, **u** and **v**. Since the graph is assumed to be strongly connected, there exists a path from **u** to **v** and also from **v** to **u**.
- Since there exists a path from **u** to **v** and from **v** to **u**, it implies that we can start at vertex **u**, travel to vertex **v**, and then come back to vertex **u**, forming a directed cycle.
- This contradicts our initial assumption that the graph is a DAG (as DAGs don't have directed cycles).
- Therefore, our initial assumption is wrong.

Administrivia

Midterm Exam on Monday, October 23rd



Chapter 5:
Divide and Conquer

Lower Bound on Sorting Algorithms

ALGORITHMS

RETURN OF THE ~~JEDI~~

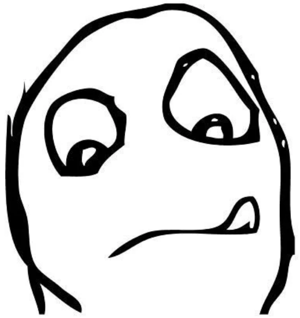
O, Ω, and Θ

Scenario # 1

Ali – (LUMS)

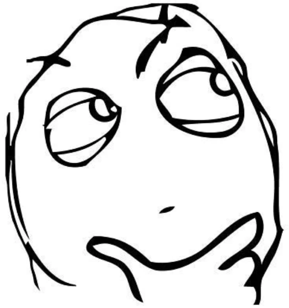
You, implement a sorting algorithm with worst-case runtime **O(n log log n)** by next week.

Okay Boss, I will try to do that ~



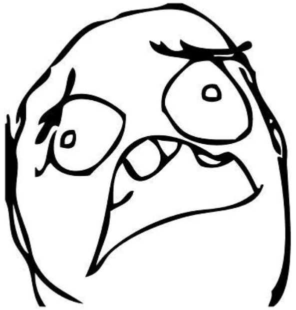
Scenario # 1

Google

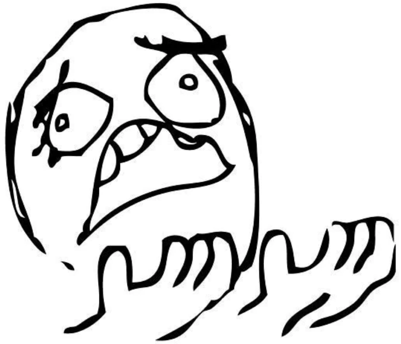


Ali try it for one month… can't do it

Boss,
I can't do it



Thankyou,
You are FIRED



Scenario # 2

Jaffer – (LUMS)

You, implement a sorting algorithm with worst-case runtime **O(n log log n)** by next week.

No, Boss. **O(n log log n)** is below the **lower bound** on sorting algorithm complexity , I can't do it, **nobody** can do it!

Lower bounds on sorting algorithms

Know your limit

we always try to make algorithms faster, but if there is a limit that you cannot exceed, you want to know

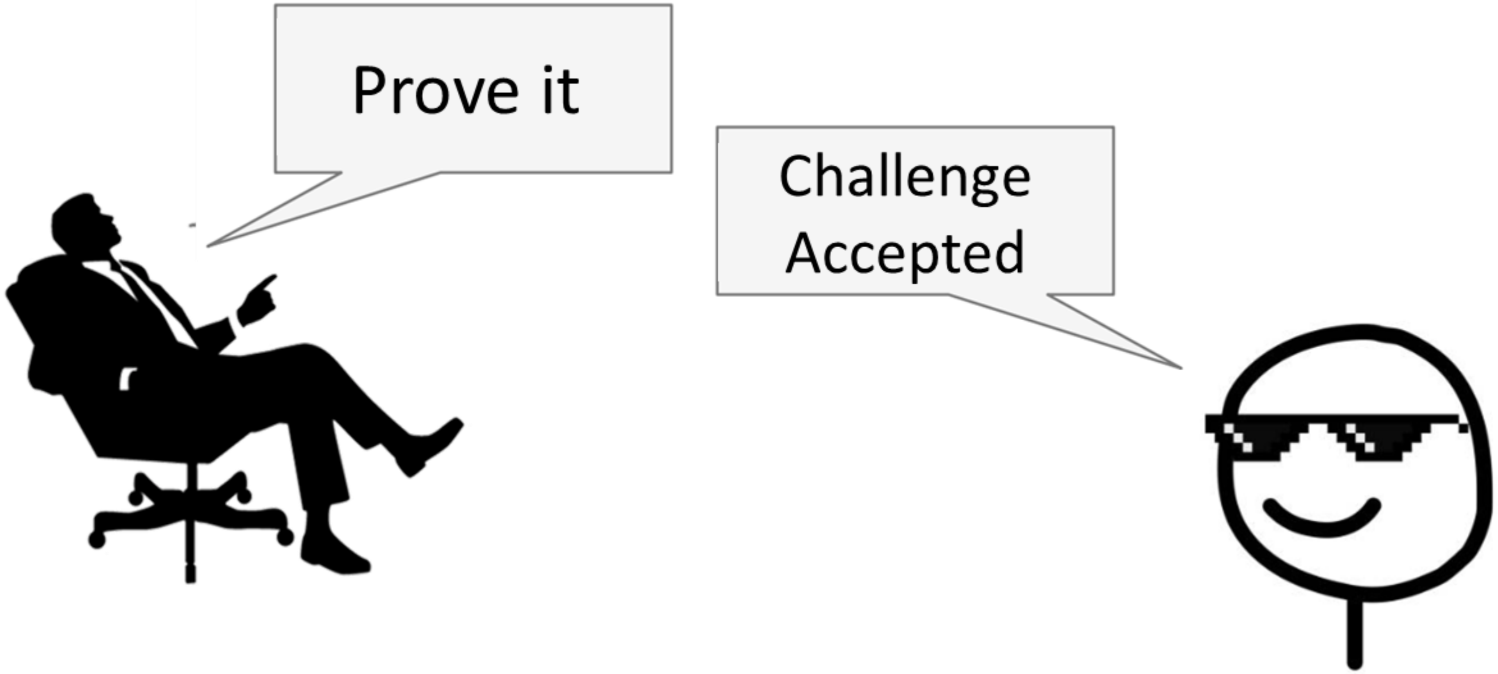
Is **O(n log n)** the best we can do?

Actually, yes, because the lower bound on sorting algorithms is **Ω(n log n)**, i.e., a sorting algorithm needs **at least** **cn log n** time to finish in worst-case.

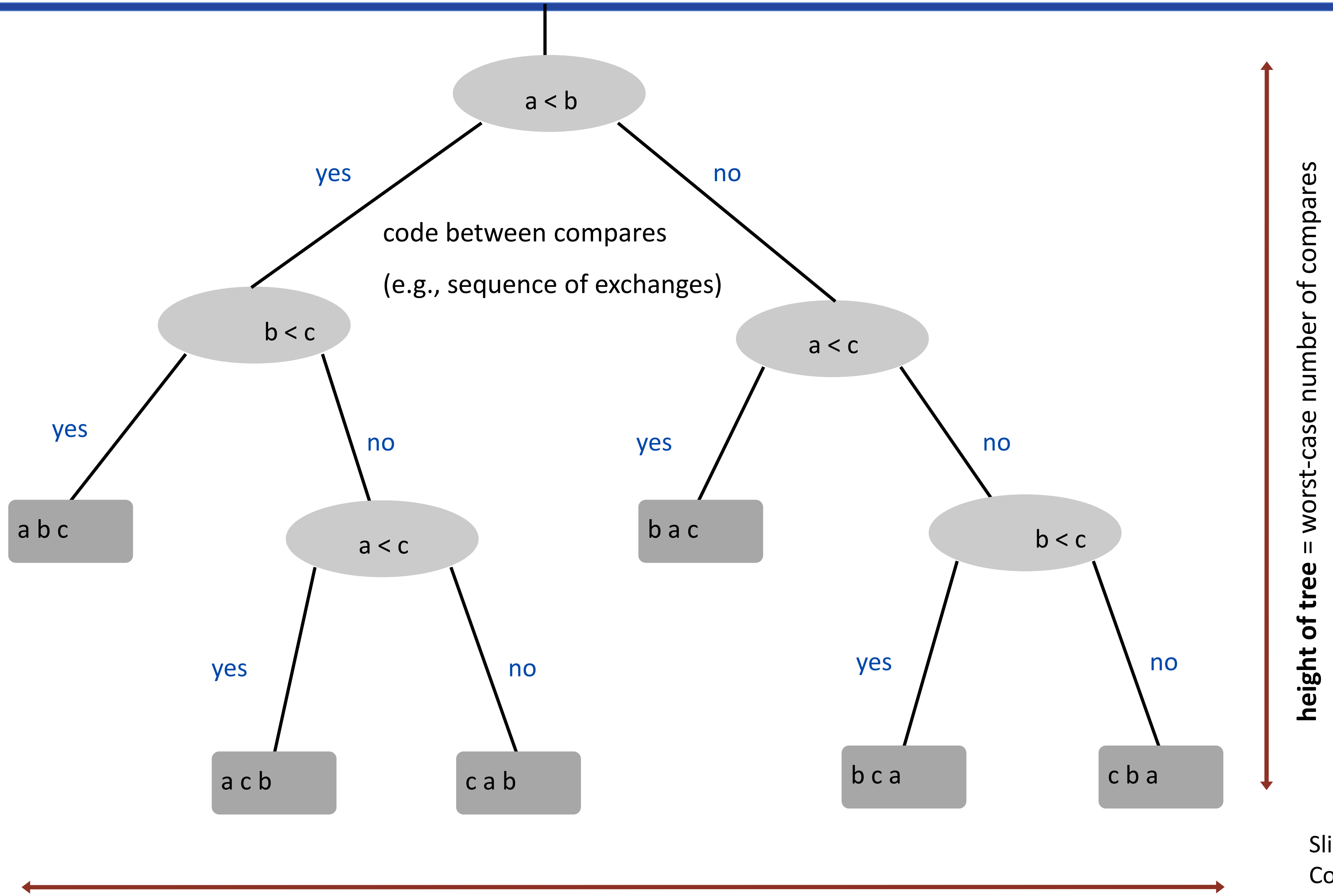
Lower bounds on sorting algorithms

actually, more precisely ...

The lower bound **n log n** applies to only all **comparison based** sorting algorithms



Comparison tree (for 3 distinct keys a, b, and c)



each reachable leaf corresponds to one (and only one) ordering;
exactly one reachable leaf for each possible ordering

Slide credit: Kevin Wayne.

Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

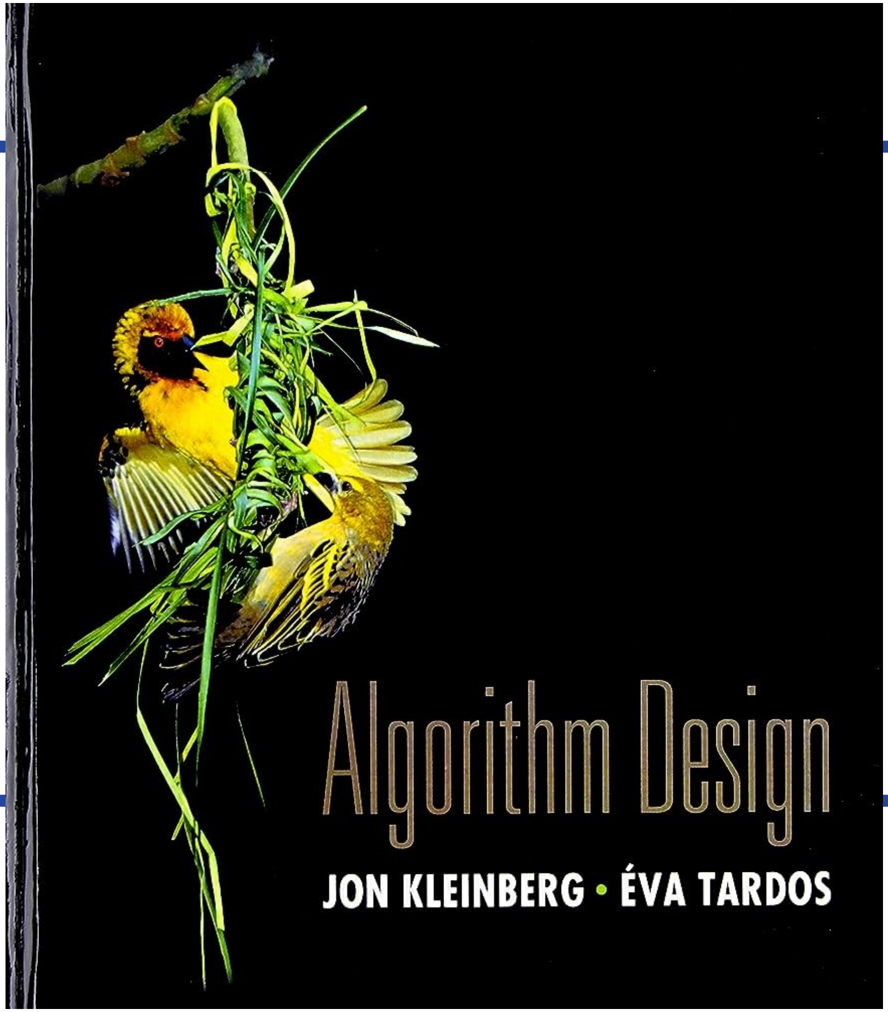
- Assume array consists of n distinct values a_1 through a_n.
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has ≤ 2^h leaves.
- $n!$ different orderings $\Rightarrow$ $n!$ reachable leaves.

$$2^h \geq \text{\# reachable leaves} = n!$$

$$\Rightarrow \quad h \geq \log_2(n!)$$

$$\geq n \log_2 n - \ln(e) \quad \blacksquare$$

Stirling's formula



Chapter 5:
Divide and Conquer

Master Theorem

- Use of Master Theorem
- Proof of Master Theorem (Next Lecture)

What is Master Theorem?

Theorem

If $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ (for constants **a**>0, **b**> 1)

Then let $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a}\, lg\, n\right)$
3. If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$, for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Master Method – Example 1

- Using Master Theorem, find the asymptotic bounds of:

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

Master Theorem

$T(n) = aT(n/b) + f(n)$

1. If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a}\, lg\, n\right)$
3. If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$, for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Master Method – Example 2

- Using Master Theorem, find the asymptotic bounds of:

$$T(n) = T\left(\frac{2n}{3}\right) + 1$$

Master Theorem

$T(n) = aT(n/b) + f(n)$

1. If $f(n) = O\left(n^{\log_b a-\epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a}\, lg\, n\right)$
3. If $f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$, for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Master Method – Example 3

- Using Master Theorem, find the asymptotic bounds of:

$$T(n) = 3T\left(\frac{n}{4}\right) + nlgn$$

Master Theorem

$T(n) = aT(n/b) + f(n)$

1. If $f(n) = O\left(n^{\log_b a-\epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a}\, lg\, n\right)$
3. If $f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$, for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Master Method – Example 4

- Using Master Theorem, find the asymptotic bounds of:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

Master Theorem

$T(n) = aT(n/b) + f(n)$

1. If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a}\, lg\, n\right)$
3. If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$, for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Master Method – Example 5

- Using Master Theorem, find the asymptotic bounds of:

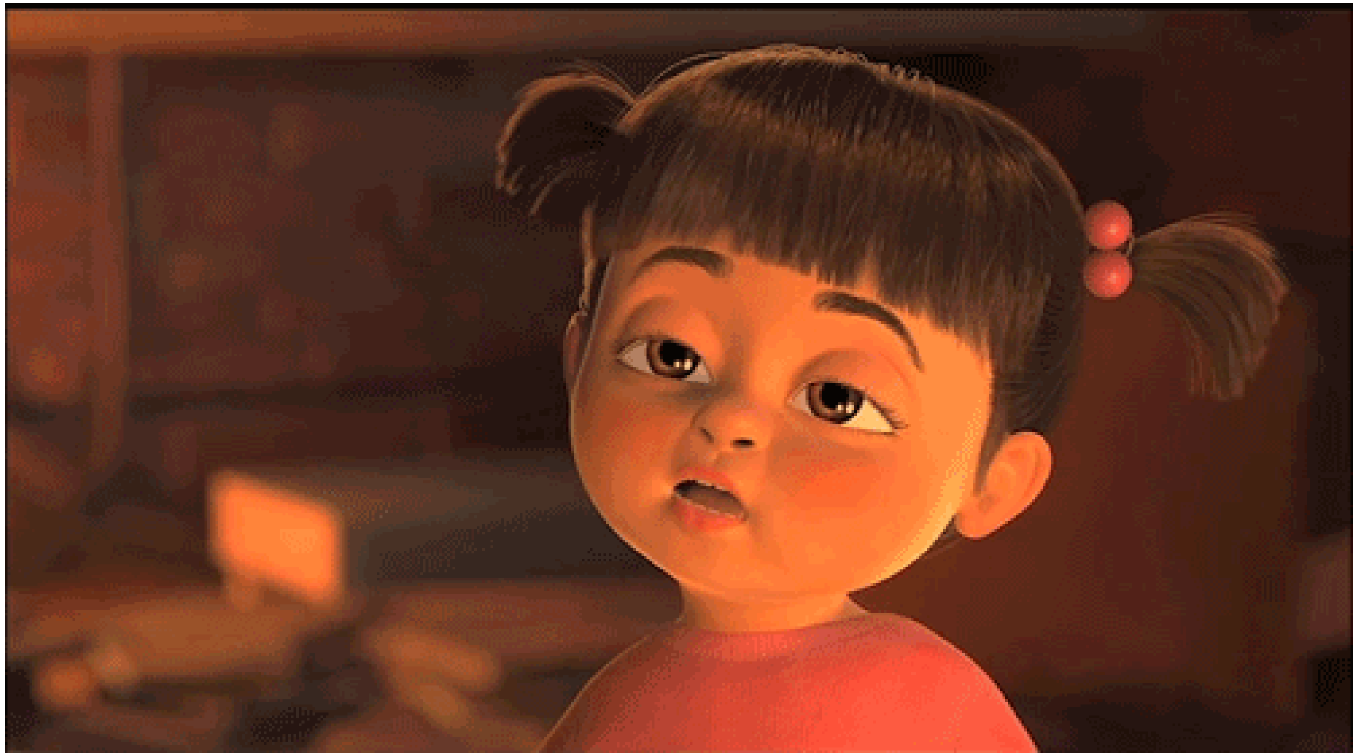
$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Master Theorem

$T(n) = aT(n/b) + f(n)$

1. If $f(n) = O\left(n^{\log_b a-\epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a}\, lg\, n\right)$
3. If $f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$, for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Thanks a lot



If you are taking a Nap, **wake up**........Lecture Over