Runtime Analysis

SOME STUFF WRITTEN IN ONENOTE

Discussion problem 1

→ $\log n^n$, n^2, $n^{\log n}$, $n \log(\log n)$, $n^{1/\log n}$, $2^{\log n}$, $\log^2 n$, $n^{\sqrt{2}}$

→ $n \log n$ (5), n^2 (7), $n^{\log n}$ (8), $n \log(\log n)$ (4), $n^{1/\log n}$ (1), $2^{\log n}$ (3), $\log^2 n$ (2), $n^{\sqrt{2}}$ (6)

$$\frac{1}{\log_2 n} = \frac{1}{\frac{\log n}{\log 2}} = \frac{\log 2}{\log n} = 2$$

P.T.O

Log properties

① $\log_b x = \dfrac{\log x}{\log b}$

② $b^{\log_b x} = x$

③ $\log_b 1 = 0$

④ $\log_a a = 1$

⑤ Change of base rule

$\log_b a = \dfrac{\log_c a}{\log_c b}$

DP 2

$f(n) = O(g(n))$ $\qquad$ $2^{f(n)} = O(2^{g(n)})$

Let $f(n) = n$, $g(n) = n^2$ $\Rightarrow \exists c,\ n \le c \cdot n^2$, TRUE,
$f(n) = O(g(n))$

$$2^{f(n)} = 2^n \le c_1 \cdot 2^{g(n)} = c_1 2^{n^2}$$

$$\Downarrow$$

$$2^n \le c_1 \cdot 2^{n^2}$$

apply log on both sides

$$n \le c_2 \cdot n^2$$

TRUE

Proof by counterexample,

Let $f(n) = 2n$, $g(n) = n$ $\Rightarrow \exists,\ 2n \le c \cdot n \Rightarrow f(n) = O(g(n))$ for $c > 3$

$$2^{f(n)} = 2^{2n} \le c_1 \cdot 2^{g(n)} = c_1 \cdot 2^n$$

$4^n \le c_1 \cdot 2^n$

FALSE

Omega : Ω (lower bound)

$$f(n) = \Omega(g(n))$$

$$\exists\, c, n_0 \quad \forall n : n \geq n_0, f(n) \geq c \cdot g(n)$$

Example : $n^2 + 2n + 1 = \Omega(n^2)$

Proof

We need to find c, n_0

$n^2 + 2n + 1 \geq n^2$ for $n \geq 1$

$\therefore \quad n_0 = 1 \quad c = 1$

DP 3

$f(n) = \Omega(g(n))$, then $2^{f(n)} = \Omega(2^{g(n)})$?

Proof by counter example

$f(n) = n$

$g(n) = 2n$

$\Rightarrow \quad 2^n \geq c \cdot 2^{2n}$

$2^n \geq c \cdot 4^n$

FALSE

Theta: Θ

$$f(n) = \Theta(g(n))$$

$$f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n))$$

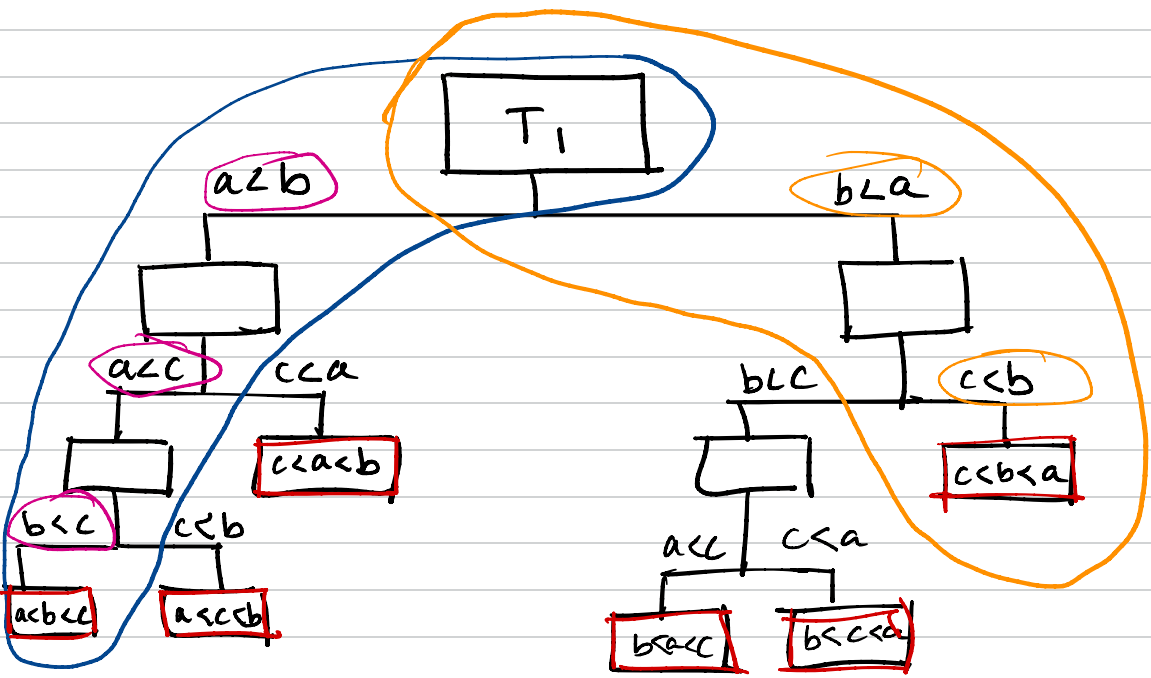
$$\exists\, c_1, c_2, n_0 \;\forall n : n \geq n_0,\; c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

---

1. $n = \Omega(n^2)$ — $n \geq c \cdot n^2$ FALSE
2. $n = \Theta(n + \log n)$ — $n = c \cdot (n + \log n)$ TRUE
3. $\log n = \Omega(n)$ — $\log n \geq c \cdot n$ FALSE
4. $n^2 = \Omega(n \log n)$ — $n^2 \geq c(n \cdot \log n)$ TRUE
5. $n^2 \log n = \Theta(n^2)$ — $n^2 \log n = c \cdot (n^2)$ FALSE
6. $3n^2 + 4n + 5 = \Theta(n^2)$ — $3n^2 = n^2$ TRUE
7. $2^n + 100n^2 + n^{100} = \Omega(n^{101})$ — $2^n \geq n^{101}$ TRUE
8. $(1/3)^n + 100 = \Theta(1)$ — $\left(\frac{1}{3}\right)^n = c \cdot 1$ TRUE

Sorting Lower Bound

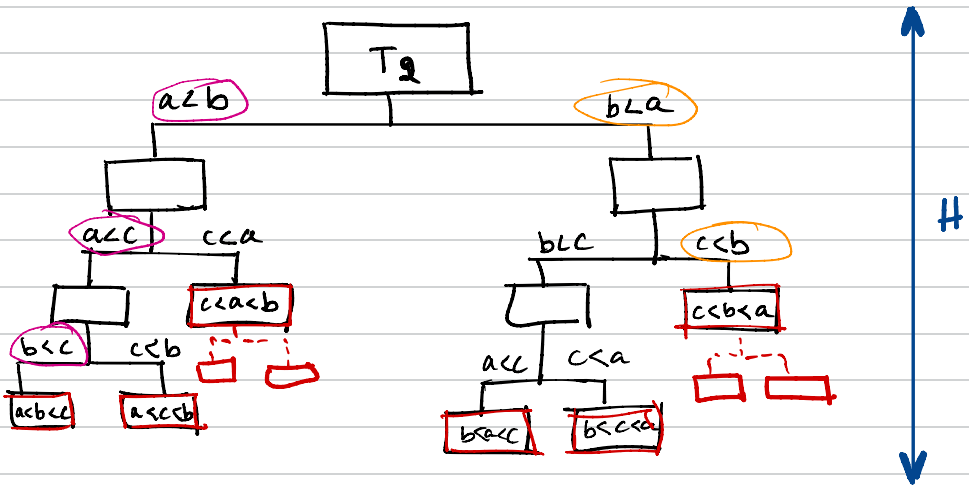
Proof that any deterministic comparison based algorithm must take $\Omega(n \log n)$ time to sort an array of n elements in the worst case.



leaves $(T_1) = n!$

$3! = 3 \times 2 = 6$

leaves $(T_2) = 2^H$



leaves $(T_2) \geq$ leaves (T_1)

$2^H \geq n!$

$\log(n!) = \log(n(n-1)(n-2)\ldots 1)$
$\geq \log(n(n-1)\ldots(n-n/2))$
$\geq \log((n/2)^{n/2}) = \Omega(n \log n)$

H is a runtime complexity

Apply log on both sides

$H \geq \log(n!) \geq c \cdot n \log n$

proof in the textbook

$H = \Omega(n \log n)$

TREES AND GRAPHS

Simple graph – No self loops & no multi edges

Tree – It is a connected graph with no cycles.

PLESE SEE PROOF ON LECTURE NOTES

→ Adjacency Lists are used for the representation of sparse ($E = O(V)$) graphs

→ Adjacency Matrix representation is used for representation of the dense ($E = \Omega(V^2)$) graphs.

Adjacency Matrix Representation

- It requires a lot of space. For V edges we need to allocate $O(V^2)$ space
- It takes constant time to figure out if a is adjacent to b.

Adjacency List Representation

- It is a 1D array where each vertex is a linked list of all other vertices connected to that vertex.
- It requires $O(V+E)$ space
- It takes linear time to figure out if a is adjacent to b.

GRAPH TRAVERSALS

Depth-First-Search (DFS): It starts at a selected node and explores as far as possible along each branch before backtracking. DFS uses a stack for backtracking.

Breadth-First-Search (BFS): It starts at a selected node and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. BFS uses a FIFO queue for bookkeeping.

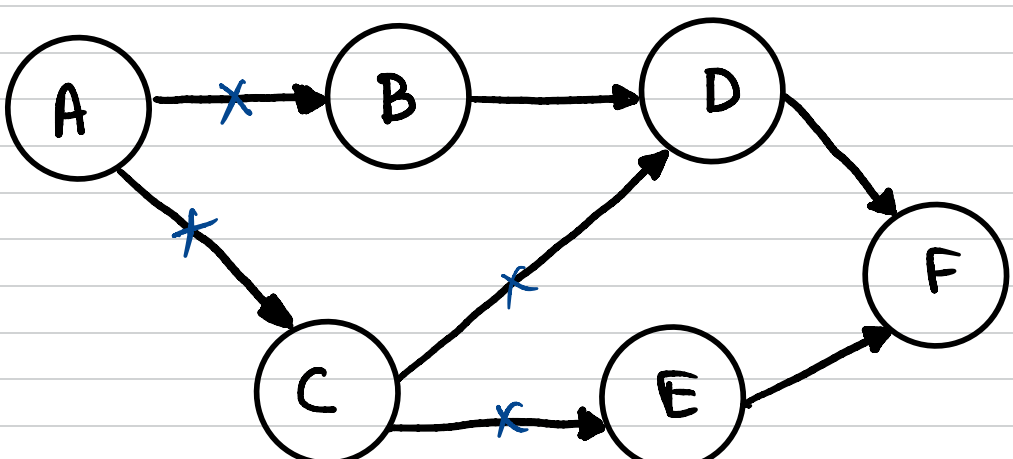
The runtime complexity is $O(V+E)$

Property 1 : They visit all the vertices in the connected component

Property 2 : The result of traversal is a spanning tree of the connected component.

- DP8 DOUBT

TOPOLOGICAL SORT ~~for~~ DAGs



Algo for topological sort:

1. Select a vertex that has zero indegree
2. Add the vertex to the output
3. Delete the vertex & its outgoing edges
4. Repeat

Output: A, C, E, B, D, F

How to do in linear time?

1. Select a vertex
2. Run DFS & return vertices that has no undiscovered leaving edges.
3. You may run DFS several times

You get vertices in reverse order

Output: F, E, D, B, C, A

Only linear Algos for graphs ever are !!

1. DFS
2. BFS
3. Topological Sort

NOTHING ELSE !!