Dijkstras

When algorithm proceeds all vertices are divided into two groups: vertices whose shortest path from the source **s**

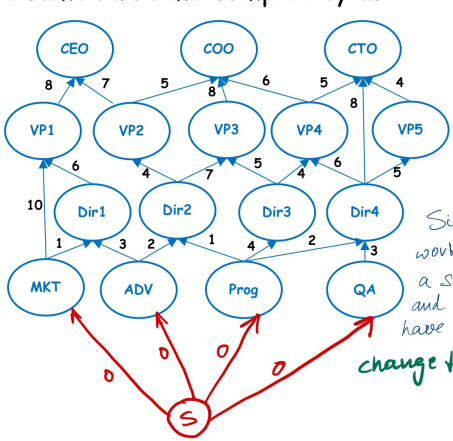
- is known
- is NOT discovered yet

Move vertices one at a time from the undiscovered set of vertices to the known set of the shortest distances, based on the shortest distance from the source.

Any shortest path problem use Dijkstras.

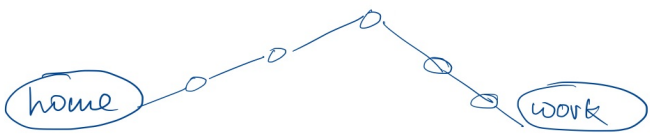
Discussion Problem 1

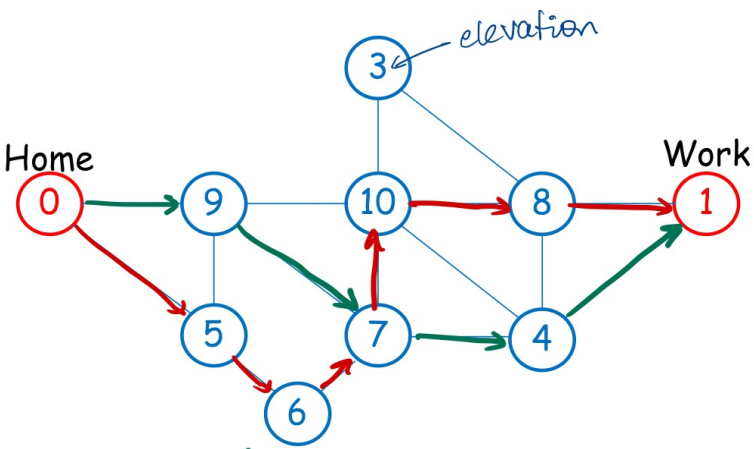
You are given a graph representing the several career paths available in industry. Each node represents a position and there is an edge from node v to node u if and only if v is a pre-requisite for u. Top positions are the ones which are not pre-requisites for any positions. Start positions are the ones which have no pre-requisites. The cost of an edge (v,u) is the effort required to go from one position v to position u. Salma wants to start a career and achieve a top position with minimum effort. Using the given graph can you provide an algorithm with the same run time complexity as Dijkstra's algorithm?



Discussion Problem 2

Hardy decides to start running to work in San Francisco to get in shape. He prefers a route to work that goes first entirely uphill and then entirely downhill. To guide his run, he prints out a detailed map of the roads between home and work. Each road segment has a positive length, and each intersection has a distinct elevation. Assuming that every road segment is either fully uphill or fully downhill, give an efficient algorithm to find the shortest path that meets Hardy's specifications.



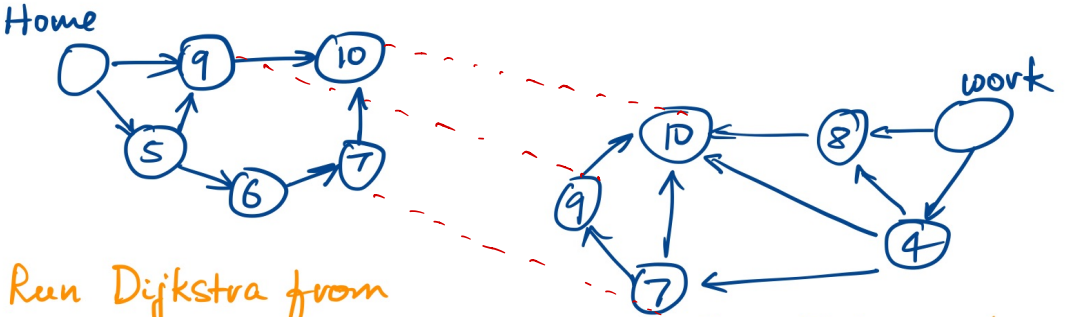


change the input

① create a new graph of uphill vertices starting from home

② create a uphill graph from work.
↳ which is the same as downhill from work

Uphill ~~Downhill~~ Uphill



Run Dijkstra from Home

Run Dijkstras from work

3. Find common vertices
4. Check all common vertices & those distances.

P.T.O

No Dijkstra

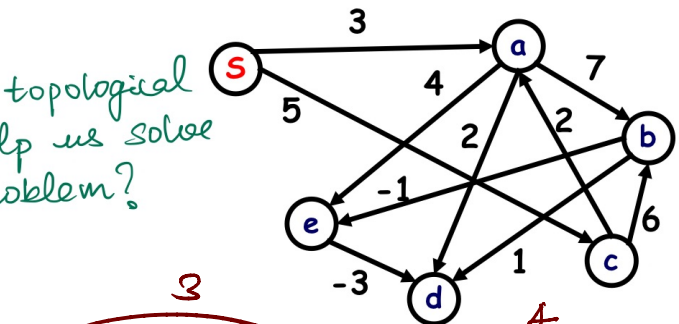
Discussion Problem 3

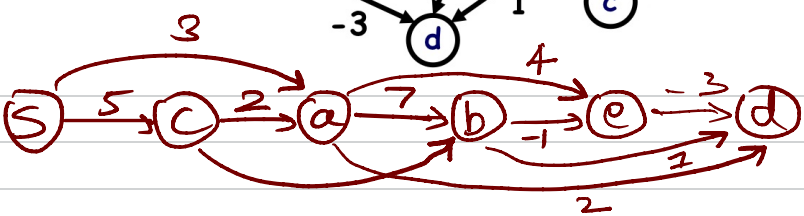
Design a linear time algorithm to find shortest distances in a DAG.

a) traversal

b) topological sort

how can topological sort help us solve this problem?





s	a	b	c	d	e	
0	3	—	5	—	—	s
		5+6				s, c
		3+7		3+2	3+4	s, c, a
	3	10	5	5	7	s, c, a, b
				7-3=4		s, c, a, b, e

Runtime : topological sort + $O(E)$

Discussion Problem 4

Why doesn't Dijkstra's greedy algorithm work on graphs with negative weights?

Run Dijkstra's:

$d(S) = 0$, $d(C) = 3$

$d(A) = 5$, $d(B) = 10$

⇒ the real shortest path

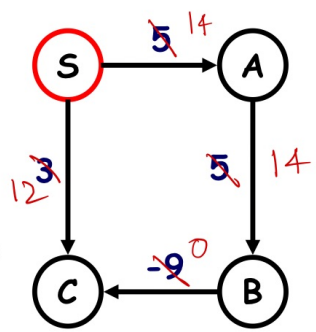
S - A - B - C

How do you fix Dijkstra?

a) reweight the graph

Run Dijkstra: S - C = 12

⇒ will still not result in the right shortest path.



b) another approach can be to use a regular queue instead of a priority queue & let it run V-1 times.

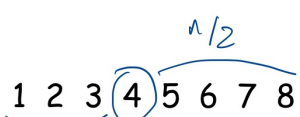
c) use a table & do dynamic programming

DIVIDE & CONQUER

Binary Search

Given a sorted array of size n:

- compare the search item with the middle
- if it's less, search in the lower half
- if it's greater, search in the upper half
- if it's equal or the entire array has been searched, terminate.



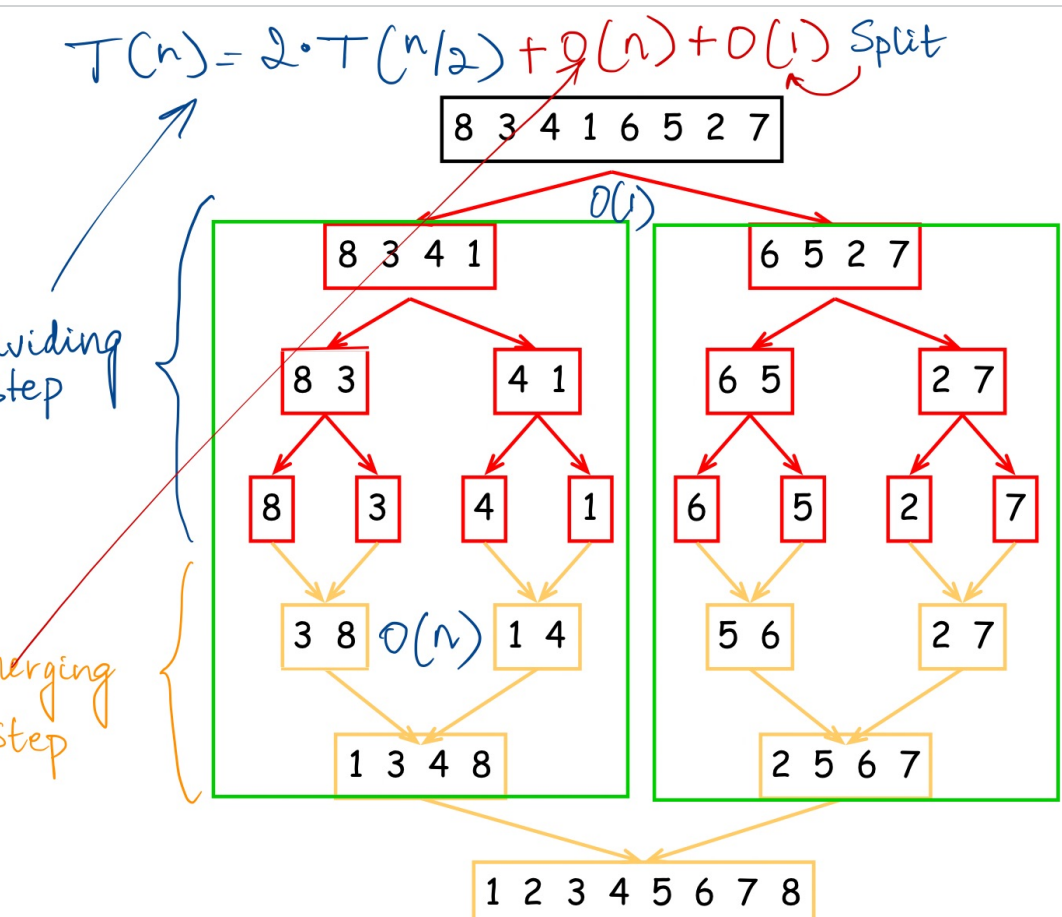
Linear	Binary
n	n
n-1	n/2
n-2	n/2
	Ω(n)

Mergesort

divides an unsorted list into two equal or nearly equal sub lists

sorts each of the sub lists by calling itself recursively, and then

merges the two sub lists together to form a sorted list



D&C Recurrences

Suppose T(n) is the number of steps in the worst case \needed to solve the problem of size n.

We define the runtime complexity T(n) by a recurrence equation.

Binary Search: $T(n) = T(n/2) + O(1)$

MergeSort: $T(n) = 2T(n/2) + O(n)$

Suppose T(n) is the number of steps in the worst case needed to solve the problem of size n.

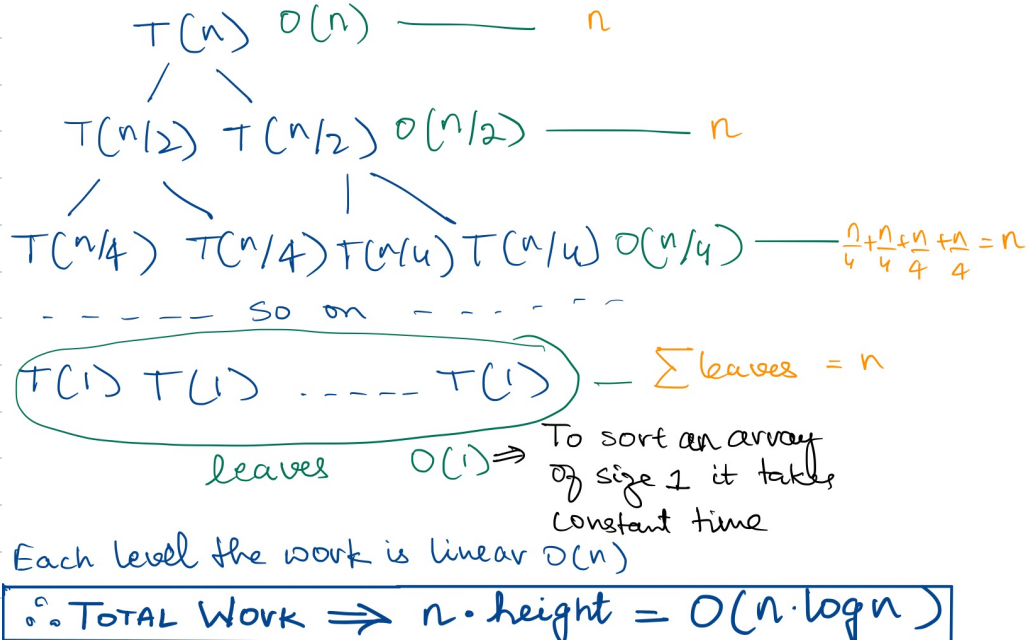
Let us divide a problem into $a \geq 1$ subproblems, each of which is of the input size n/b where $b > 1$.

The total complexity T(n) is obtained by

$$T(n) = a \cdot T(n/b) + f(n)$$

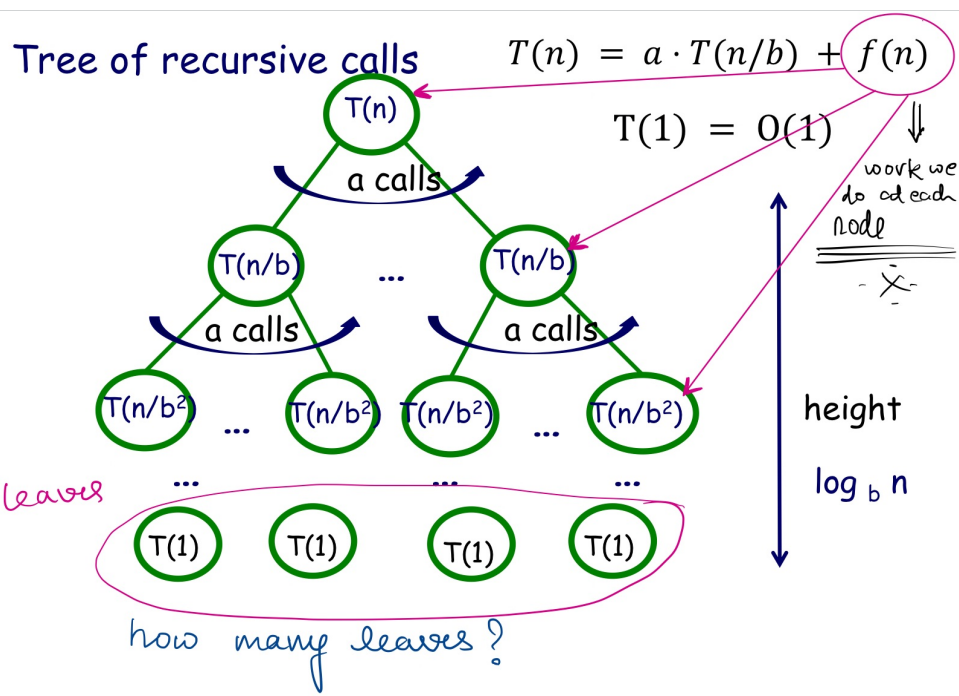
Here f(n) is a complexity of combining subproblem solutions (including complexity of *dividing* step).

Mergesort: tree of recursive calls



Each level the work is linear $O(n)$

∴ TOTAL WORK ⇒ $n \cdot \text{height} = O(n \cdot \log n)$



Counting leaves

Let h = height

# of leaves = a^h

$$T(n) = a \cdot T(n/b) + f(n)$$

$$a^h = a^{\log_b n} = \left(a^{\log_a n}\right)^{1/\log_a b} = n^{\log_b a}$$

$$\frac{1}{\log_a b} = \log_b a$$

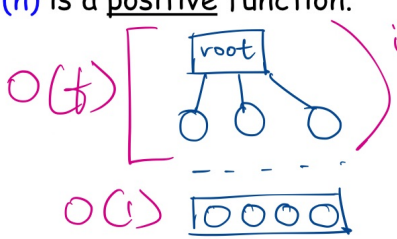
$$\log_b n = \frac{\log_a n}{\log_a b}$$

The Master Theorem

The master method provides a straightforward ("cookbook") method for solving recurrences of the form

$$T(n) = a \cdot T(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants (not necessarily integers) and $f(n)$ is a positive function.



$$T(n) = \sum_{\text{internal nodes}} O(f) + \sum_{\text{leaves}} O(1)$$

The Master Theorem

$T(n) = a \cdot T(n/b) + f(n)$, $a \geq 1$ and $b > 1$ any

Let $c = \log_b a$.

Case 1: (only leaves)

if $f(n) = O(n^{c-\varepsilon})$, then $T(n) = \Theta(n^c)$ for some $\varepsilon > 0$.

Case 2: (all nodes) Merge Sort

if $f(n) = \Theta(n^c \log^k n)$, $k \geq 0$, then $T(n) = \Theta(n^c \log^{k+1} n)$

extra log

Case 3: (only internal nodes)

if $f(n) = \Omega(n^{c+\varepsilon})$, then $T(n) = \Theta(f(n))$ for some $\varepsilon > 0$.

REFER PPT for multiplicati

P.T.O

Finding the maximum subsequence sum (MSS)

$3, -4, 5, -2, -2 \mid 6, -3, 5, -3, 2$

A_1 A_2

$[l_1, r_1, max_1] = MSS(A_1)$ recursive

$[l_2, r_2, max_2] = MSS(A_2)$ recursive

$[l_3, r_3, max_3] = span(A_1, A_2)$ iterative

return $MAX(max_1, max_2, max_3)$

Implementation of span

(-2) must be a part of span

(6) must be a part of span

Compute partial sums

$0, -3, [1, -4, -2 \mid 6, 3, 8], 5, 7$

l_3 r_3

Runtime : $T(n) = 2 \cdot T(n/2) + \text{span } O(n)$

$T(n) = \Theta(n \cdot \log n)$

Review

Design a new Mergesort algorithm in which instead of splitting the input array in half we split it in the ratio 1:3.
Write down the recurrence relation for the number of comparisons.
What is the runtime complexity of this algorithm?

$$T(n) = T(n/4) + T(3n/4) + O(n)$$

$$T(n) = \Theta(n \log n)$$

↘ we know this because of 1st lecture, the above recurrence relation has nothing to do w/ it.