Analysis of Algorithms

V. Adamchik

University of Southern California

CSCI 570

Lecture 5

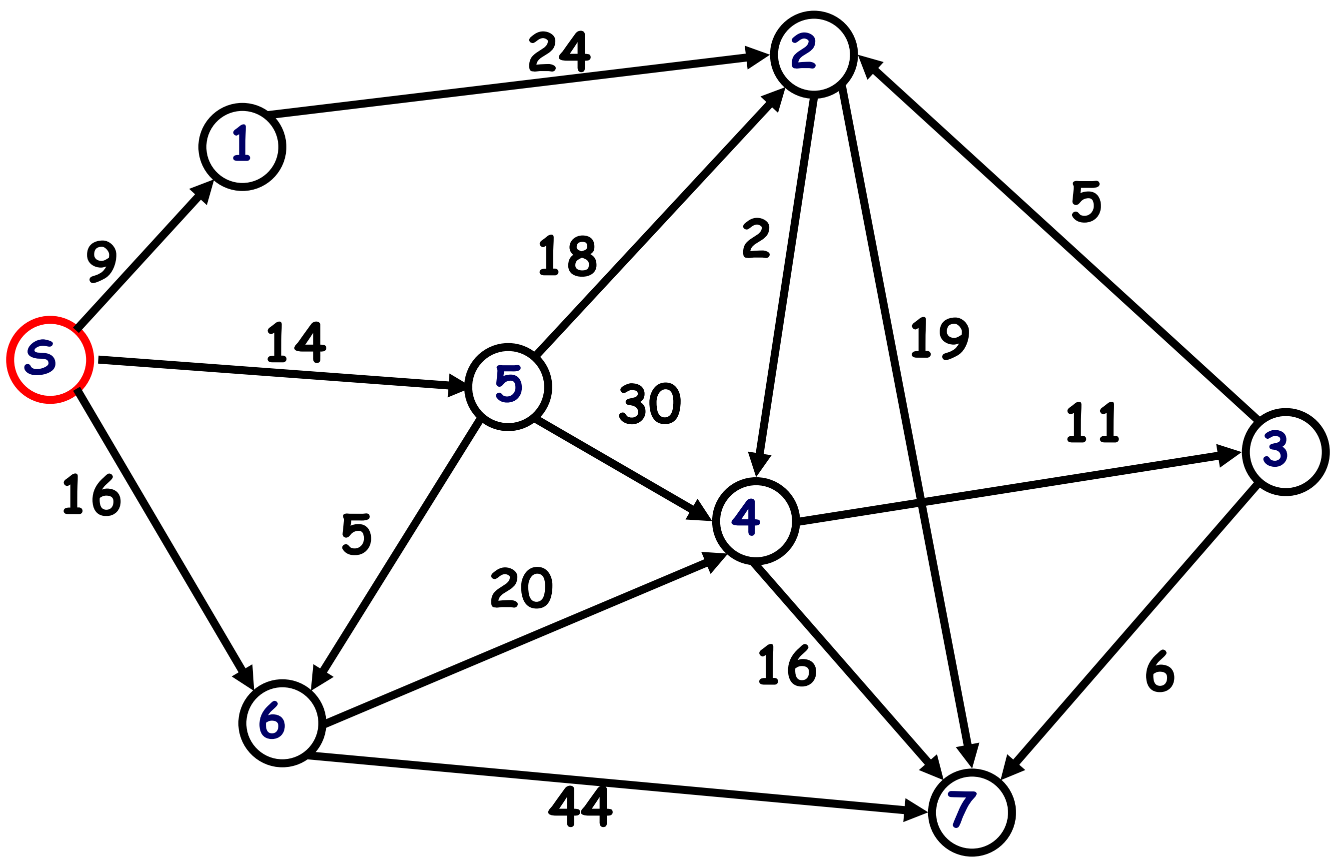
Fall 2023

Dijkstra's Algorithm
Divide and Conquer Algorithms

Reading: chapters 4 & 5

The Shortest Path Problem

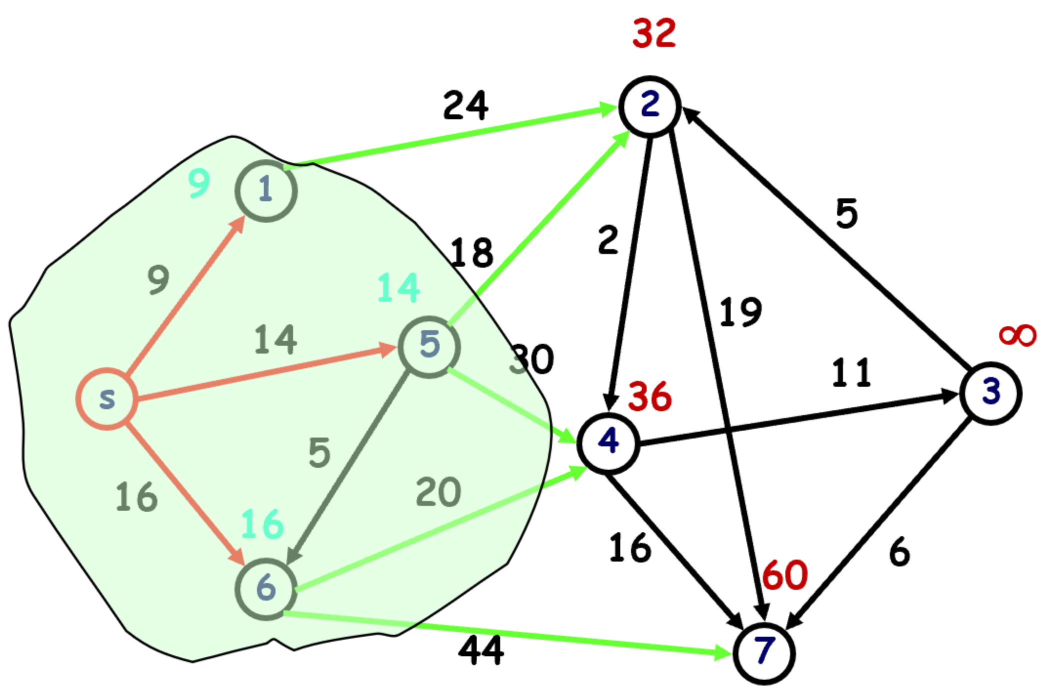
Given a positively weighted graph G with a source vertex s, find the shortest path from s to all other vertices in the graph.



Greedy Approach

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.

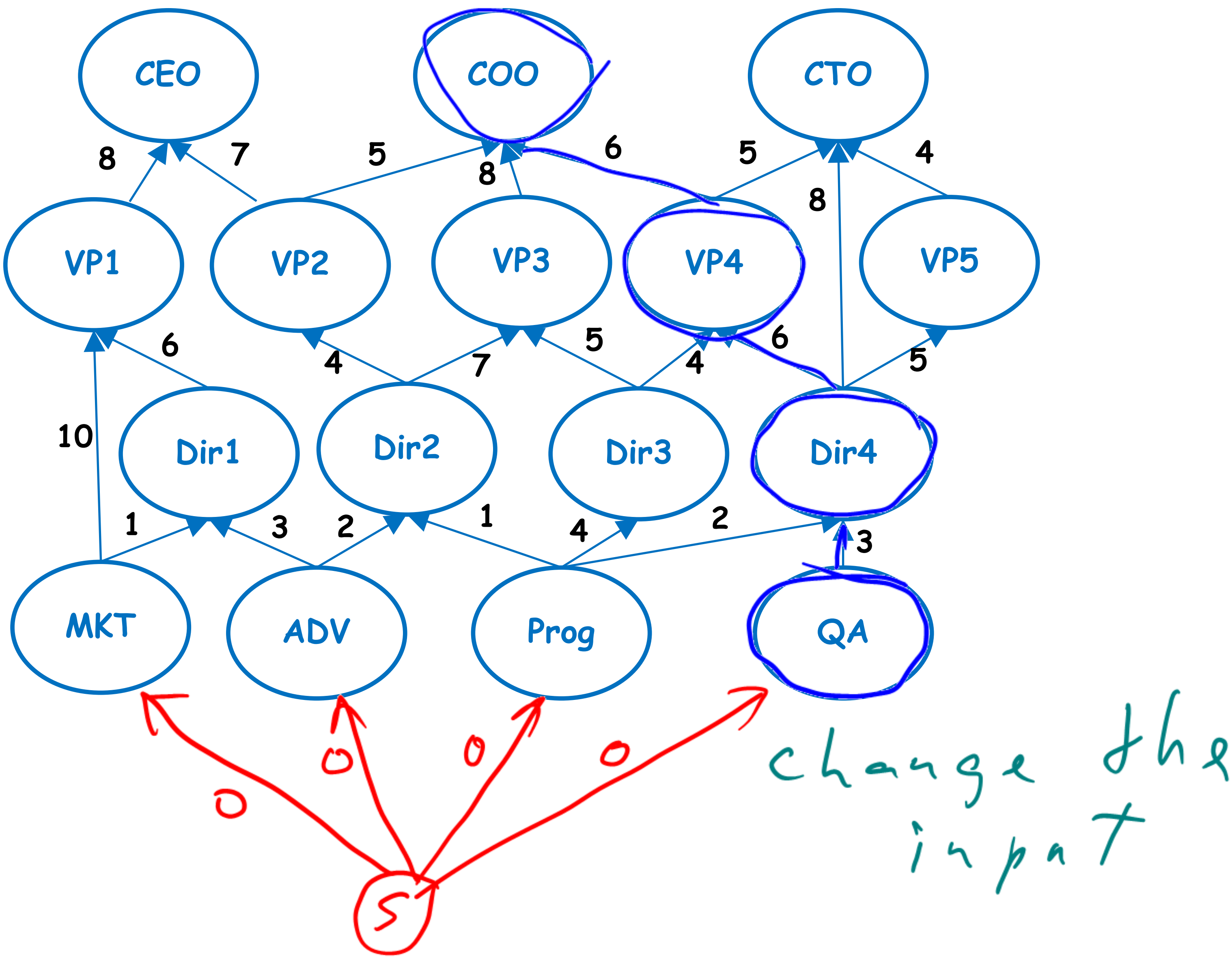


solution tree = { s, 1, 5, 6 }

heap = {2, 3, 4, 7}

Discussion Problem 1

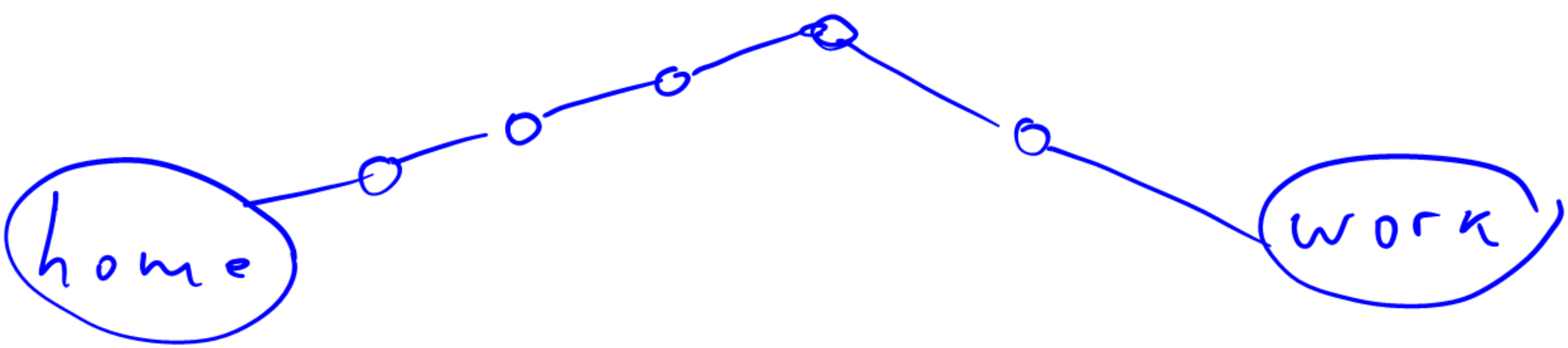
You are given a graph representing the several career paths available in industry. Each node represents a position and there is an edge from node v to node u if and only if v is a pre-requisite for u. Top positions are the ones which are not pre-requisites for any positions. Start positions are the ones 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?

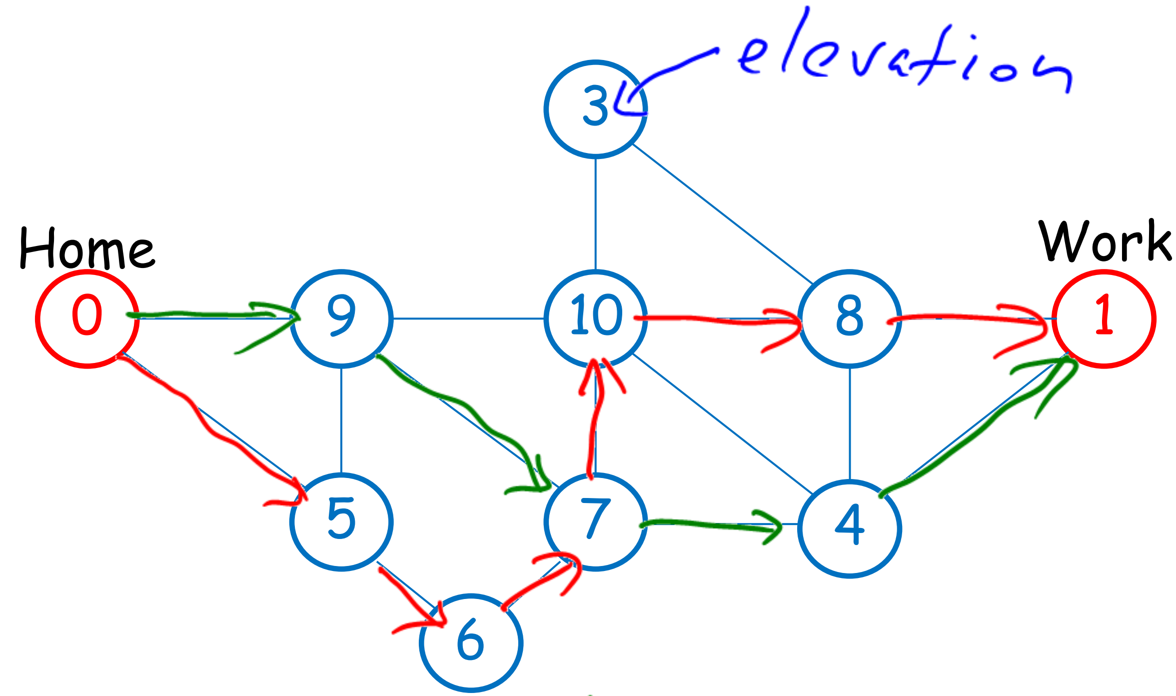


CEO
COO
CTO
8
7
5
8
6
5
8
4
VP1
VP2
VP3
VP4
VP5
6
4
7
5
4
6
5
10
Dir1
Dir2
Dir3
Dir4
1
3
2
1
4
2
3
MKT
ADV
Prog
QA
0
0
0
0
s
change the input

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.





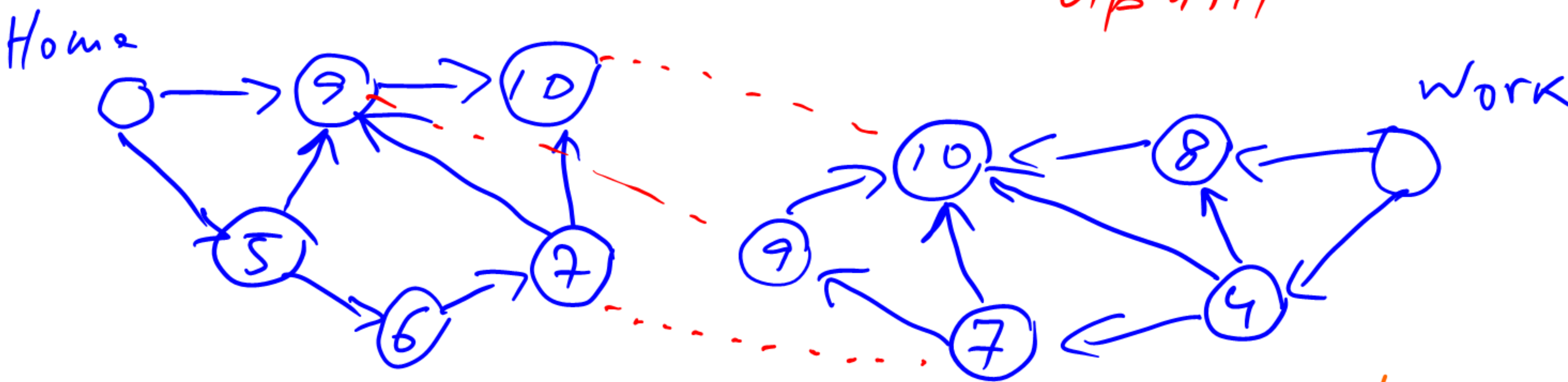
Change the input

① create a new graph of uphill vertices starting from Home

② create a downhill graph from Work

Uphill

~~Downhill~~ Uphill



Run Dijkstra from Home

Run Dijkstra from Work

3. find common vertices
4. check all common vertices and those distances

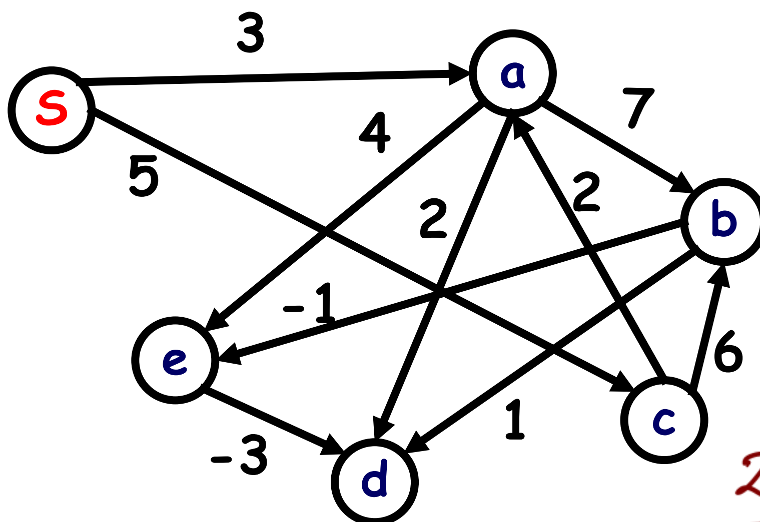
Discussion Problem 3

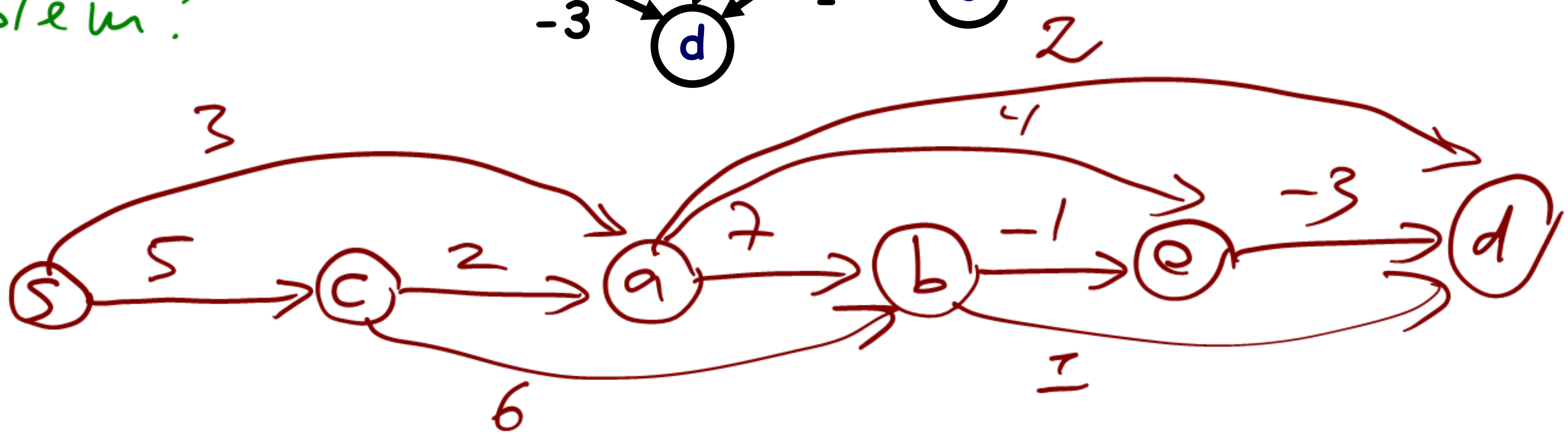
no Dijkstra

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

a) traversal
b) topological sort

how topological sort helps us to solve this problem?





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

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

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

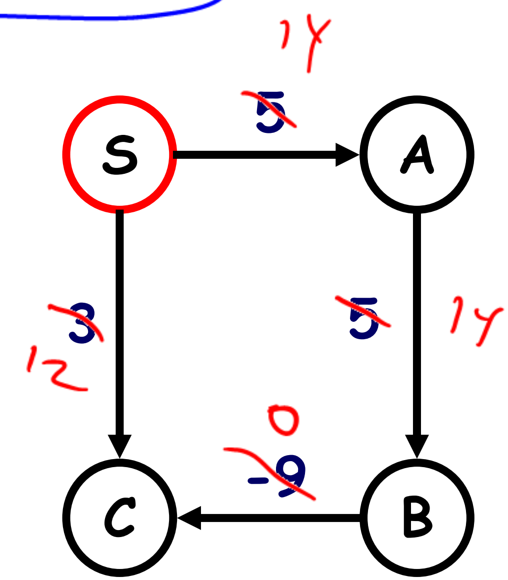
S - A - B - C

?

How do you fix Dijkstra?

a) re-weight the graph

Run Dijkstra: $s - c = 12$



Break

Divide and Conquer Algorithms

A divide-and-conquer algorithm consists of

- dividing a problem into smaller subproblems
- solving (recursively) each subproblem
- then combining solutions to subproblems to get solution to original problem

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.

$$\underbrace{1\quad 2\quad 3}_{n/2}\quad 4\quad \overbrace{5\quad 6\quad 7\quad 8}^{n/2}$$

linear	Binary
n	n
n - 1	n/2
n - 2	n/4

O(n)

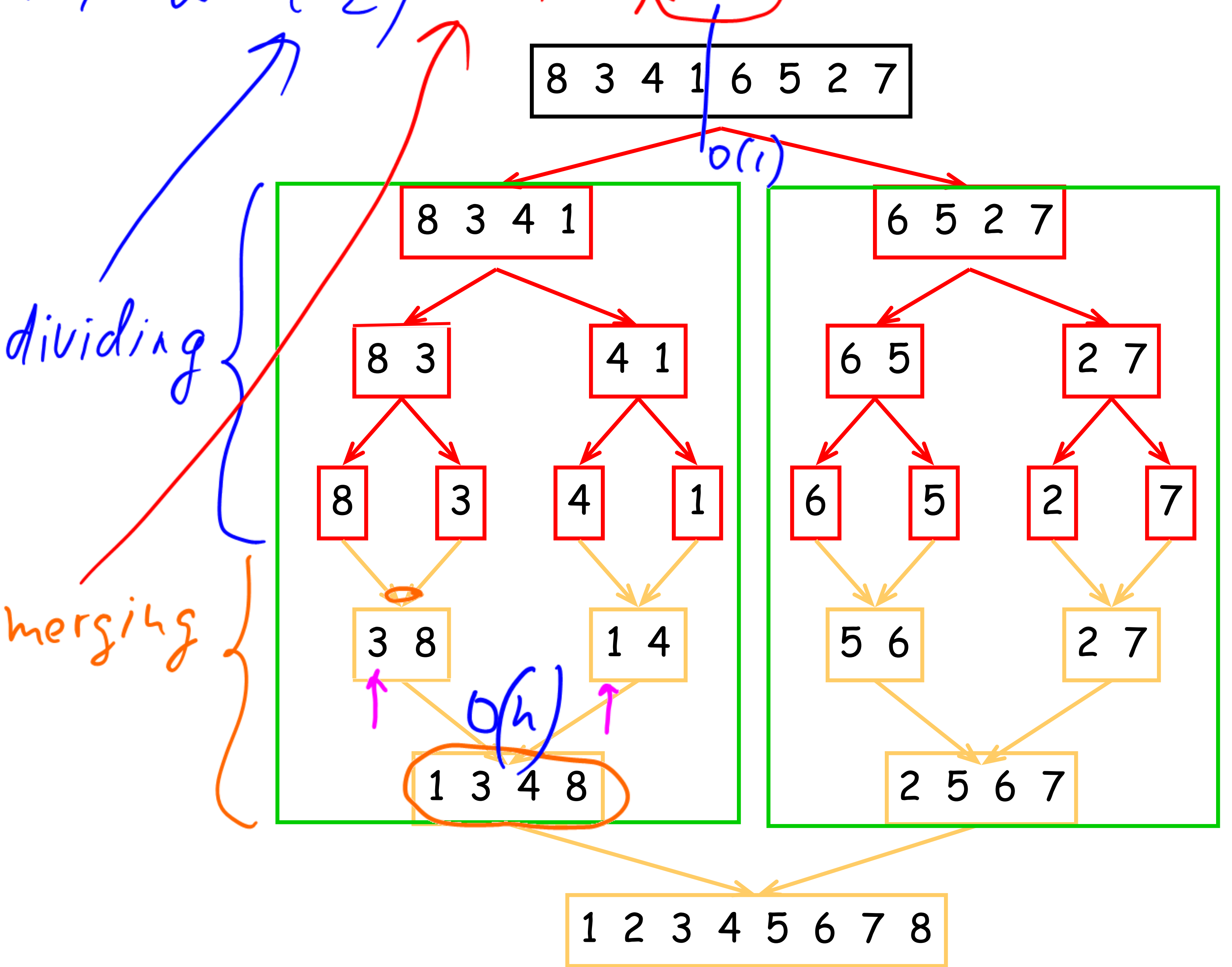
Mergesort

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

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n) + O(1)$$ split



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

MergeSort: $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

D&C Recurrences

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

> 0

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

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

Mergesort: tree of recursive calls

$T(n)$ $O(n)$ —— n

$T(\frac{n}{2})$ $T(\frac{n}{2})$ $O(\frac{n}{2})$ —— $n = \frac{n}{2} + \frac{n}{2}$

$T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $O(\frac{n}{4})$ —— $\frac{n}{4} + \frac{n}{4} + \frac{n}{4} + \frac{n}{4} = n$

. . . . so on . . .

. . . .

$T(1)$ $T(1)$ . . . $T(1)$ —— $\sum$ leaves $= n$

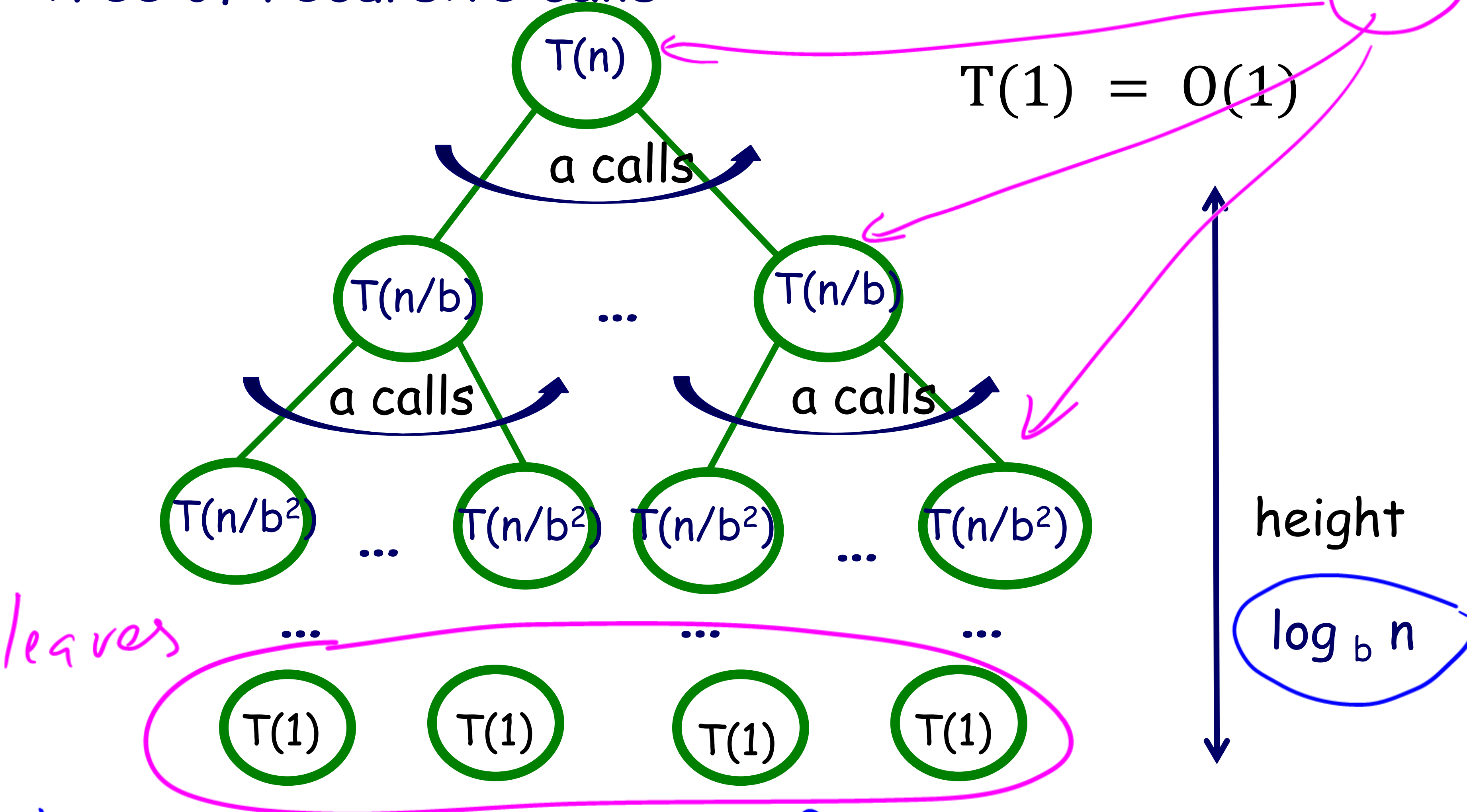
leaves $O(1)$

Total work: $n \cdot \text{height} = O(n \log n)$

Tree of recursive calls

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

$$T(1) = O(1)$$



how many leaves?

Counting leaves

Let h = height

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

$\#$ of leaves $= a^h$

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

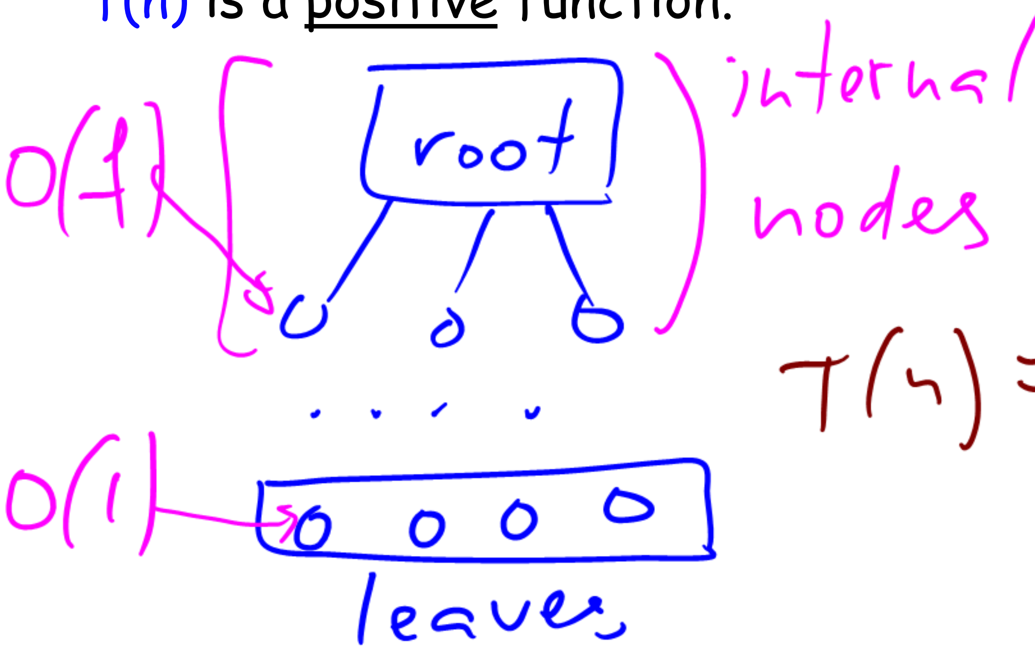
$$\log_b n = \frac{\log_a n}{\log_a b}$$

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 <u>positive</u> function.



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

The Master Theorem

$T(n) = a \cdot T(n/b) + \underbrace{f(n)}_{\geq 0}$, $a \geq 1$ 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)   Mergesort

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

Case 3: (only internal nodes)

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

Discussion Problem 5

Solve the recurrence by the Master Theorem:

$$T(n) = 16\ T(n/4) + 5\ n^3$$

$a = 16$
$b = 4$
$c = \log_b a = 2 \rightarrow$ # of leaves $O(n^2)$
$f(n) = O(n^3)$

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

Case 1: if $f(n)=O(n^{c-\varepsilon})$, then $T(n)=\Theta(n^c)$
Case 2: if $f(n)=\Theta(n^c \log^k n)$, then $T(n)=\Theta(n^c \log^{k+1} n)$
Case 3: if $f(n)=\Omega(n^{c+\varepsilon})$, then $T(n) = \Theta(f(n))$

where $c = \log_b a$.

$T(n) = \Theta(n^3)$

Discussion Problem 6

$c = \log_b a$

Solve the recurrence by the Master Theorem:

Recurrence		leaves	$f(n)$	$T(n)$
1. $A(n) = 3\,A(n/3) + 15$	case 1	$O(n)$	$O(1)$	$\Theta(n)$
2. $B(n) = 4\,B(n/2) + n^3$	case 3	$O(n^2)$	$O(n^3)$	$\Theta(n^3)$
3. $C(n) = 4\,C(n/2) + n^2$	case 2	$O(n^2)$	$O(n^2)$	$\Theta(n^2 \log n)$
4. $D(n) = 4\,D(n/2) + n$				$\Theta(n^2)$
5. $E(n) = 4\,E(n/2) + n^2 \log n$	case 2, $k=1$	$O(n^2)$	$O(n^2 \log n)$	$\Theta(n^2 \log^2 n)$

Break

Integer Multiplication

Given two n-digit integers a and b, compute a × b.

Brute force solution: $O(n^2)$ bit operations.

$$15451\,7766 = 15451 \cdot 10^4 + 7766$$

n bits

$$a \times b = (x_1 \cdot 10^{n/2} + x_0) \cdot (y_1 \cdot 10^{n/2} + y_0) =$$

$$= x_1 \cdot y_1 \cdot 10^n + (x_1 \cdot y_0 + x_0 \cdot y_1) 10^{n/2} + x_0 \cdot y_0$$

$O(1)$ $\quad$ $O(n)$

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + O(1) + O(n)$$

$$T(n) = \Theta(n^2)$$

```
    1234
x   1111
    ----
    1234
   1234
  1234
+1234
 -------
 1370974
```

1961

Karatsuba's algorithm

Divide-and-conquer algorithm. Split each integer in two parts and consider their product:

$$a \times b = (x_1 \cdot 10^{n/2} + x_0) \cdot (y_1 \cdot 10^{n/2} + y_0)$$

$$x_1 y_0 + x_0 y_1 = (x_0 + x_1)(y_0 + y_1) - x_0 y_0 - x_1 y_1$$

Hashtable

$$a \times b = x_1 y_1 10^n + [\quad] 10^{\frac{n}{2}} + x_0 y_0$$

$$T(n) = 3 \cdot T\left(\frac{n}{2}\right) + O(n), \quad T(n) = \Theta\left(n^{\log 3}\right) = \Theta\left(n^{1.58}\right)$$

Discussion Problem 7

Consider another divide and conquer algorithm for integer multiplication. The key idea is to divide a large integer into 3 parts (rather than 2) of size approximately n/3 and then multiply those parts. What would be the runtime complexity of this multiplication?

$154517766 = 154 \cdot 10^6 + 517 \cdot 10^3 + 766$

$T(n) = 9T(\frac{n}{3}) + O(n), \quad T(n) = \Theta(n^2)$

$< \log 3$

5 multiplications

$T(n) = 5T(\frac{n}{3}) + O(n), \quad T(n) = \Theta(n^{\log_3 5})$

Matrix Multiplication

A B C = AxB

$$n \underset{n}{\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} \overset{O(n)}{a_{11}b_{11} + a_{12}b_{21}} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

CPU – integers

GPU – matrix

TPU – tensor, 3D matrix

Runtime: $O(n \cdot n^2) = O(n^3)$

Matrix Multiplication

The usual rules of matrix multiplication holds for
block matrices

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

8 multiplications

$$A + B = 4\underset{4}{\boxed{}} + \underset{4}{\boxed{}}\,4$$

D&C Algorithm

Let $n = 2^k$ and $M(A,B)$ denote the matrix product

1. if A is 1x1 matrix, return $a_{11} * b_{11}$.
2. write $$A = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) \qquad B = \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right)$$

   where A_{ij} and B_{ij} are $n/2 \times n/2$ matrices.
3. Compute $C_{ij} = M(A_{i1},B_{1j}) + M(A_{i2},B_{2j})$
4. Return $$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Runtime: $T(n) = 8 \cdot T\left(\frac{n}{2}\right) + \overset{\text{additions}}{O(n^2)}$

$T(n) = \Theta(n^3)$

1968

Strassen's Algorithm

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} s_1 + s_2 - s_4 + s_6 & s_4 - s_5 \\ s_6 + s_7 & s_2 - s_3 + s_5 - s_7 \end{pmatrix}$$

$$s_1 = (a_{12} - a_{22})\,(b_{21} + b_{22})$$
$$s_2 = (a_{11} + a_{22})\,(b_{11} + b_{22})$$
$$s_3 = (a_{11} - a_{21})\,(b_{11} + b_{12})$$
$$s_4 = (a_{11} + a_{12})\,b_{22}$$
$$s_5 = a_{11}\,(b_{12} - b_{22})$$
$$s_6 = a_{22}\,(b_{21} - b_{11})$$
$$s_7 = (a_{21} + a_{22})\,b_{11}$$

It takes 7 multiplications

$$T(n) = 7 \cdot T\left(\frac{n}{2}\right) + O(n^2)$$
$$T(n) = \Theta\left(n^{\log 7}\right)$$

2.8

$$s_6 + s_7 = a_{22} b_{21} - \cancel{a_{22} b_{11}} + a_{21} b_{11} + \cancel{a_{22} b_{11}}$$

Fast Matrix Multiplication

1969, Strassen $O(n^{2.808})$.
1978, Pan $O(n^{2.796})$
1979, Bini $O(n^{2.78})$
1981, Schonhage $O(n^{2.548})$
1981, Pan $O(n^{2.522})$
1982, Romani $O(n^{2.517})$
1982, Coppersmith and Winograd $O(n^{2.496})$
1986, Strassen $O(n^{2.479})$
1989, Coppersmith and Winograd $O(n^{2.376})$
2010, Stothers $O(n^{2.374})$
2011, Williams $O(n^{2.3728642})$
2014, Le Gall $O(n^{2.3728639})$

Finding the Maximum Subsequence Sum

Given an array A[0,…, n-1] of integers, design a D&C algorithm that finds a subarray A[i, …, j] such that

$$A[i] + A[i+1] + \ldots + A[j]$$

is the maximum.

For example,

$$A = \{3, -4, \underbrace{5, -2, -2, 6, -3, 5,}_{\text{sum}} -3, 2\}$$

Output: {5, -2, -2, 6, -3, 5}

Sum = 5-2-2+6-3+5 = 9

Finding the Maximum Subsequence Sum (MSS)

3, -4, 5, -2, -2, | 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(max1, max2, max3)

Finding the Maximum Subsequence Sum (MSS)

3, -4, 5, -2, -2, | 6, -3, 5, -3, 2

$\underbrace{3, -4, 5, -2, -2}_{A_1}$ | $\underbrace{6, -3, 5, -3, 2}_{A_2}$

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 | 6, 3, 8, 5, 7

l_3 r_3

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

$O(n)$

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

Review

For each of the following recurrences, give an expression for the runtime T(n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

1. $T(n) = 16\ T(n/4) + 5\ n^3 + \log n$ $\quad T(n) = \Theta(n^3)$
2. $T(n) = 4\ T(n/2) + n^2 \log n + \cancel{n^2}$ $\quad T(n) = \Theta(n^2 \log^2)$
3. $T(n) = 4\ T(n/8) - n^2$ negative $\quad T(n) = NA$
4. $T(n) = 2^n\ T(n/2) + n$ $\quad T(n) = NA$
5. $T(n) = 0.2\ T(n/2) + n \log n$ $\quad T(n) = NA$

≥ 1

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?

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

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