Analysis of Algorithms

V. Adamchik

Lecture 5

Spring 2023

CSCI 570

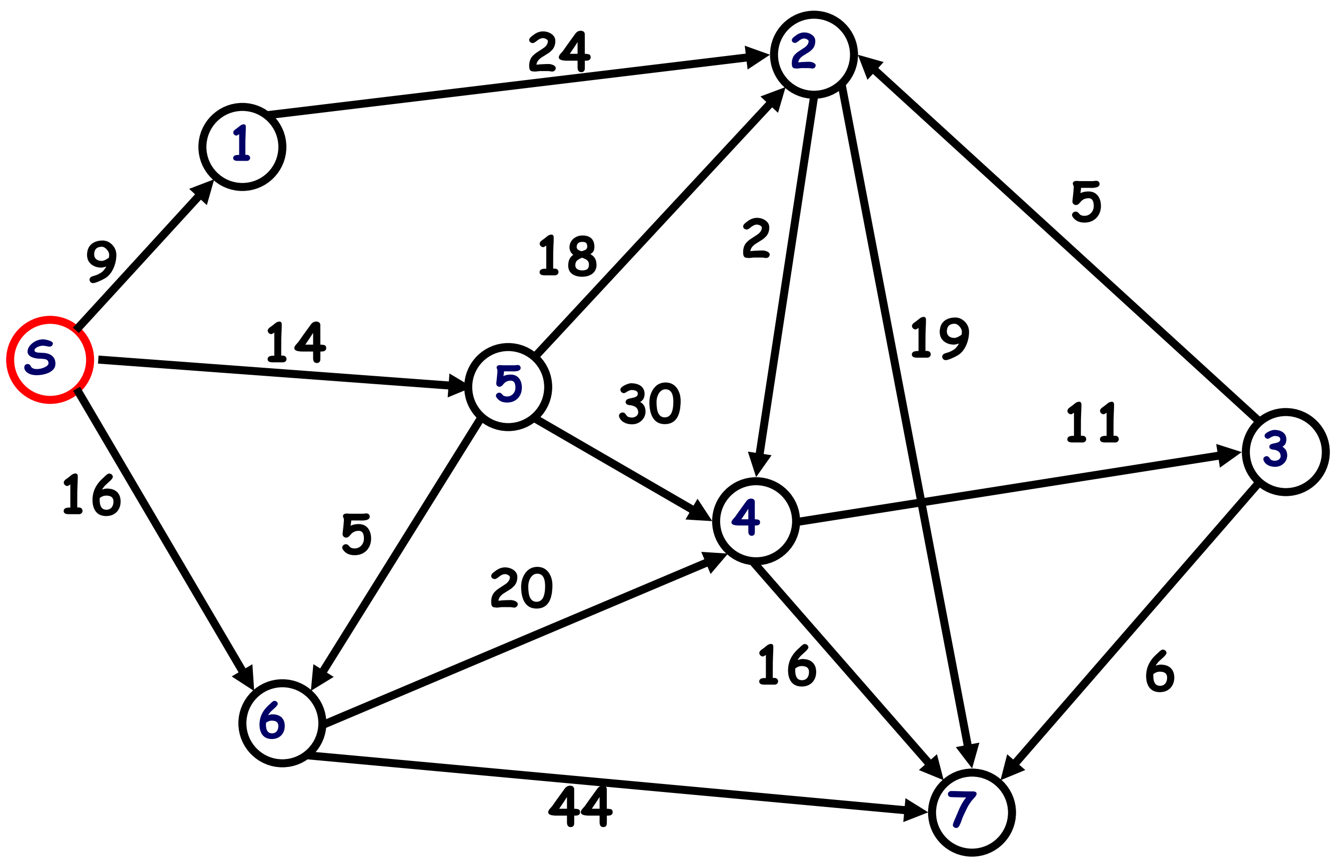
University of Southern California

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.

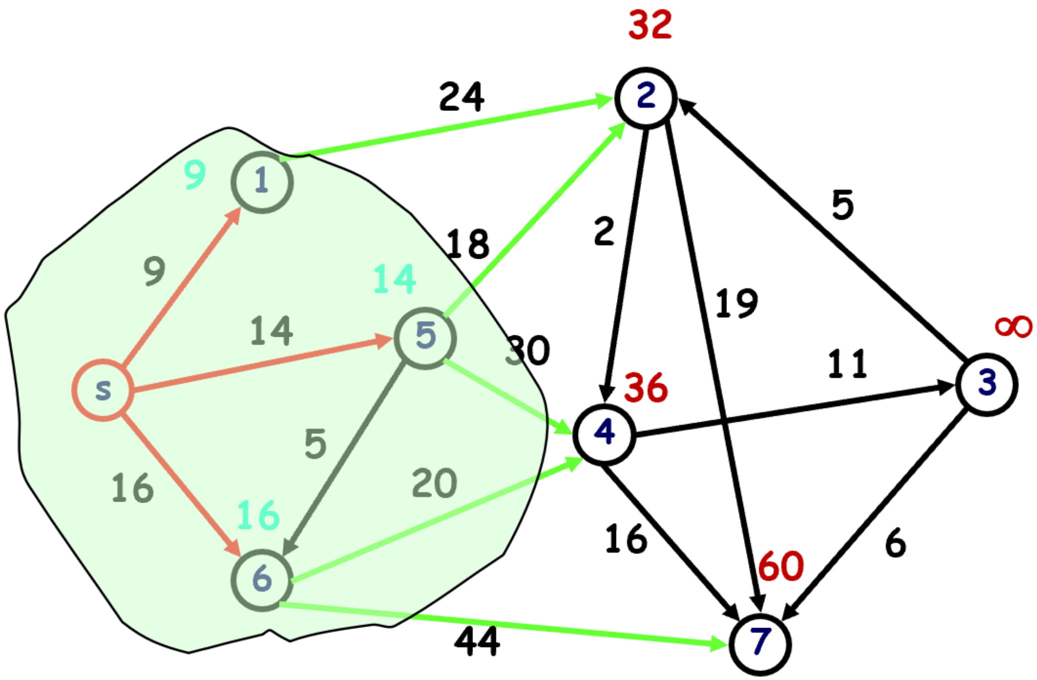


Greedy Approach

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}

Runtime Complexity

$D(v)$ – distance from s to v

heap

Algorithm:

1. DeleteMin, $O(\log V)$
2. Update (decreaseKey) distances in a heap

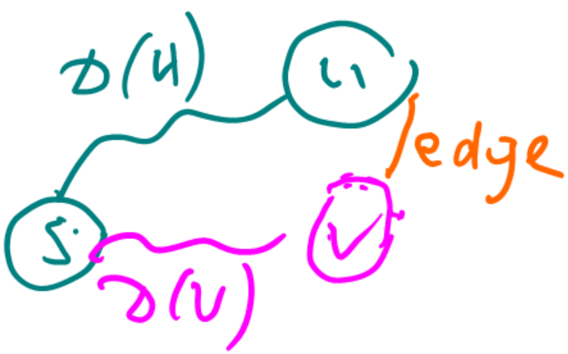
Compute: $\text{Min}\left(D(v),\ D(u) + (u,v)\right)$

binary heap $O(\log V)$

Runtime: $O(V\log V + E\log V)$

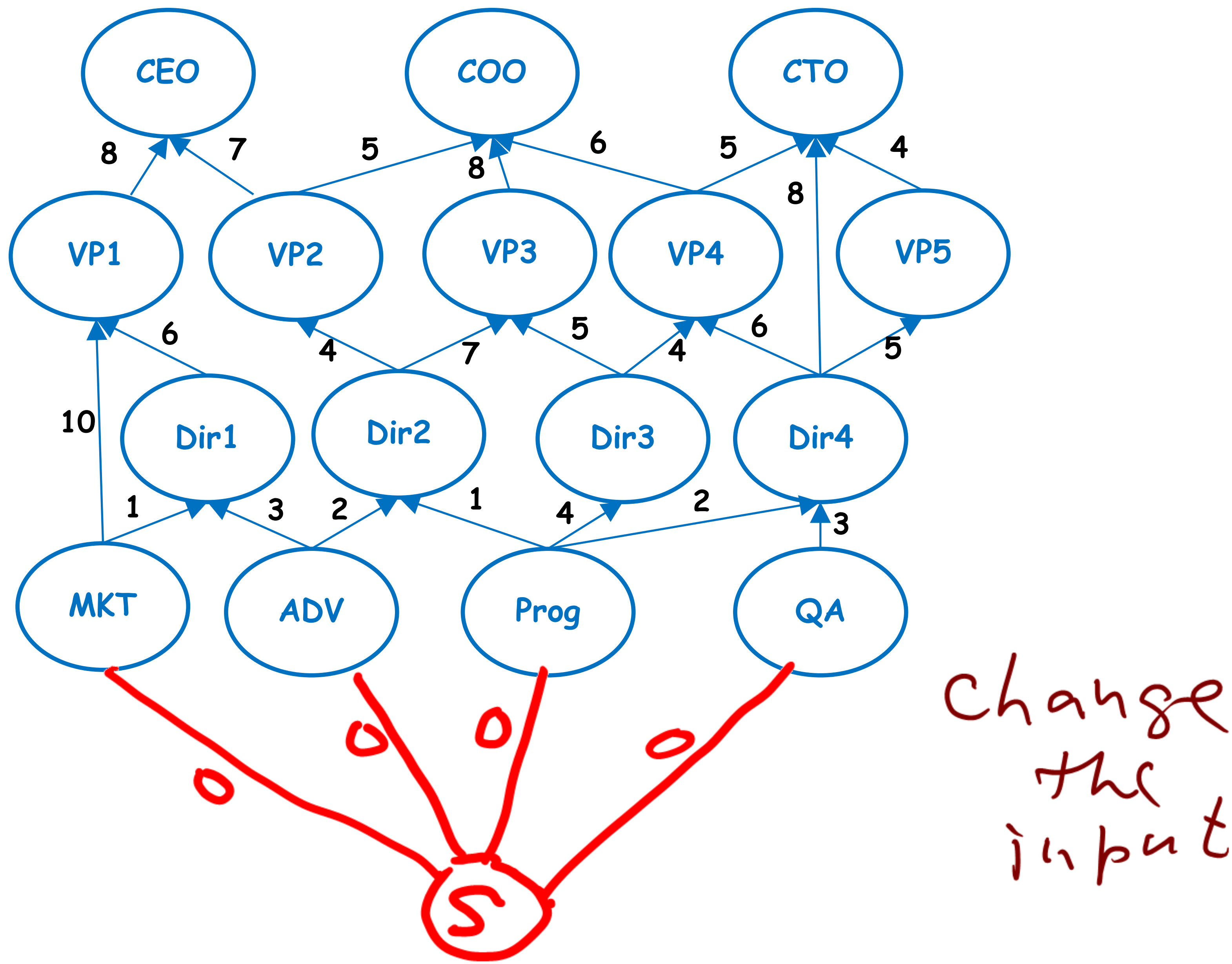
Fibonacci heap

$O(V\log V + E)$



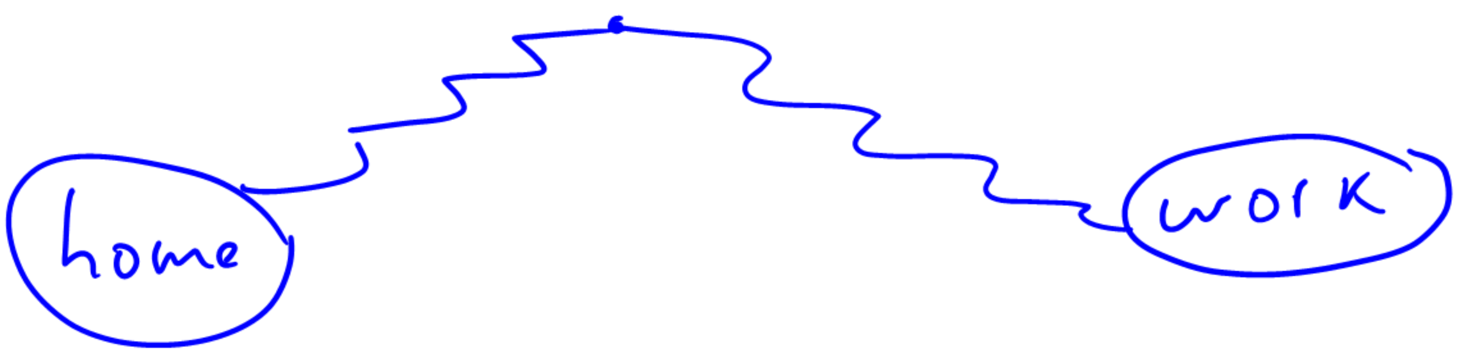
Discussion Problem 1

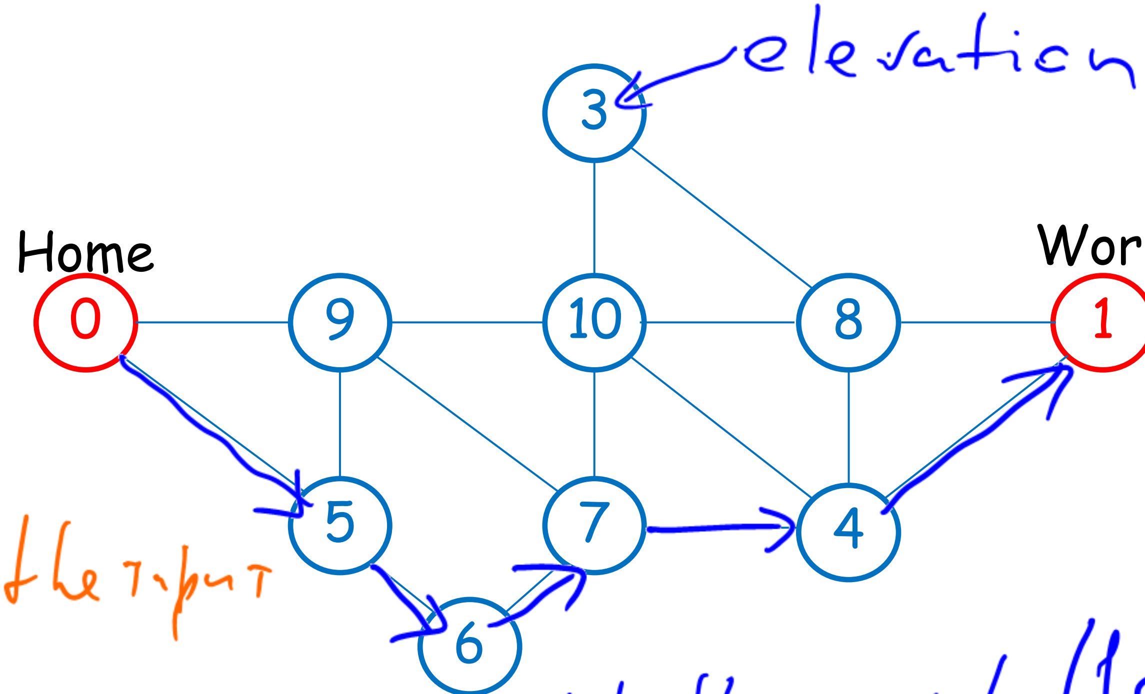
You are given a graph representing the several career paths available in industry. Each node represents a position and there is an edge from node v to node u if and only if v is a pre-requisite for u. Top positions are the ones which are not pre-requisites for any positions. Start positions are the ones 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
VP1
VP2
VP3
VP4
VP5
Dir1
Dir2
Dir3
Dir4
MKT
ADV
Prog
QA
8
7
5
8
6
5
8
4
6
4
7
5
4
6
5
10
1
3
2
1
4
2
3
0
0
0
0
S
Change the input

Discussion Problem 2

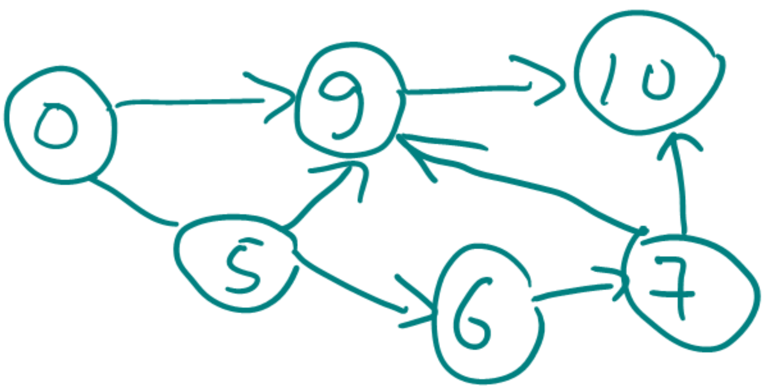
Hardy decides to start running to work in San Francisco to get in shape. He prefers a route to work that goes first entirely uphill and then entirely downhill. To guide his run, he prints out a detailed map of the roads between home and work. Each road segment has a positive length, and each intersection 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.



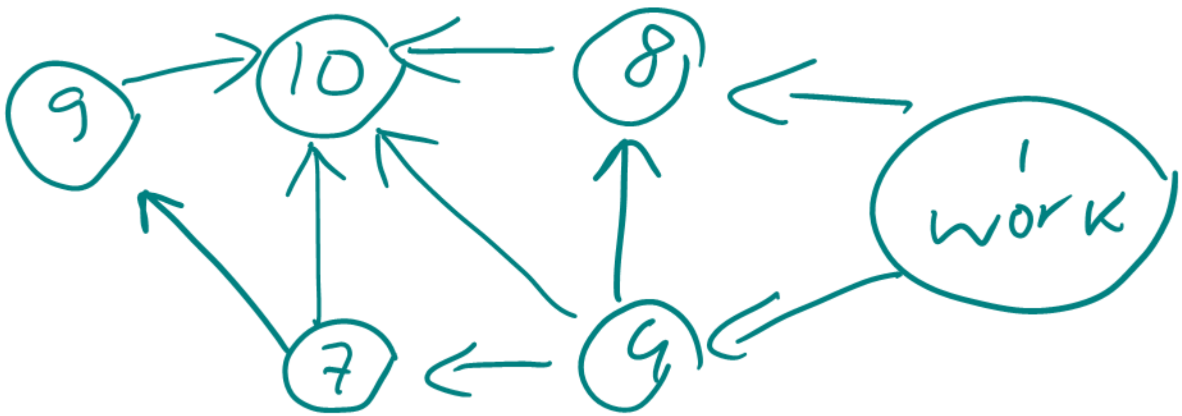


Change the input

① G_1, create an uphill graph (from home)



② G_2, create an uphill graph from work



Algorithm:

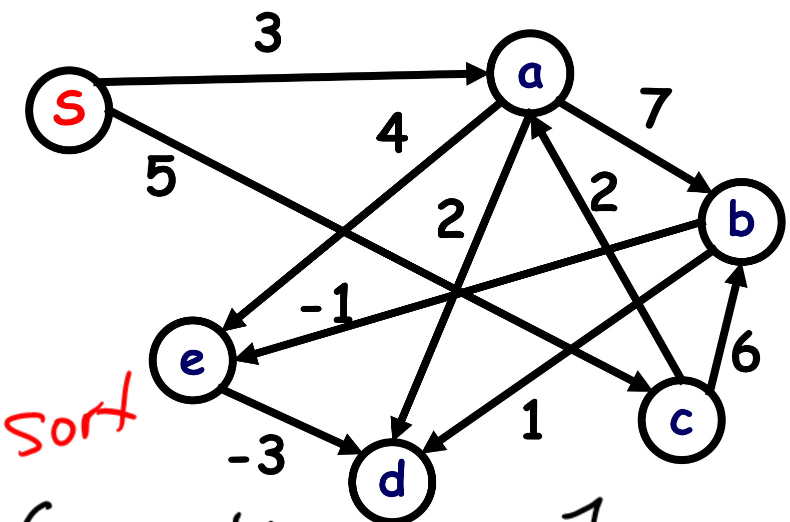
- Run Dijkstra on G_1 $\longrightarrow T_1$
- Run Dijkstra on G_2 $\longrightarrow T_2$
- find common vertices in T_1 and T_2

Discussion Problem 3

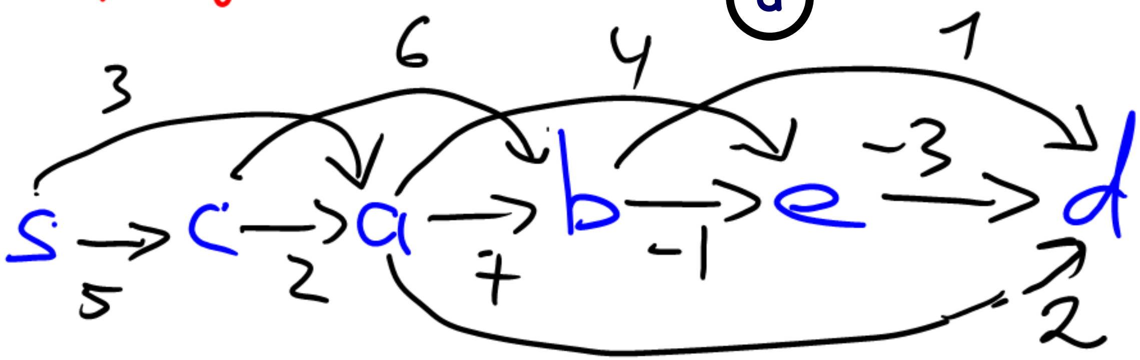
Design a linear time algorithm to find shortest distances in a DAG.

a) traversal

b) topological sort



Run topological sort



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

Runtime: $O\Big(\overbrace{(V+E)}^{\text{top. sort}} + V\cdot E\Big)$

$O((V+E)+E) = O(E)$ non-linear?

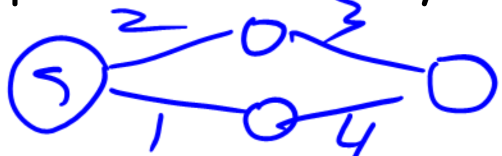
Do we do so many updates?

Do we update any edge twice?

Review Questions

(T/F) If all edges in a connected undirected graph have distinct positive weights, the shortest path between any two vertices is unique.

example



$2+3 = 1+4 = 5$

(T/F) Suppose we have calculated the shortest paths from a source to all other vertices. If we modify the original graph, G, such that weights of all edges are increased by 2, then the shortest path tree of G is also the shortest path tree of the modified graph. $+2$

example

$s-x-y = 2$

$s-y = 5$

(T/F) Suppose we have calculated the shortest paths from a source to all other vertices. If we modify the original graph G such that weights of all edges are doubled, then the shortest path tree of G is also the shortest path tree of the modified graph. $*2$

$(s-x-y) < s-y$

$2 \times (s-x-y) < 2(s-y)$

Discussion Problem 4

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

Run Dijkstra: $d(c)=3$

OPT Solution: S-A-B-C $=5+5-9=1$

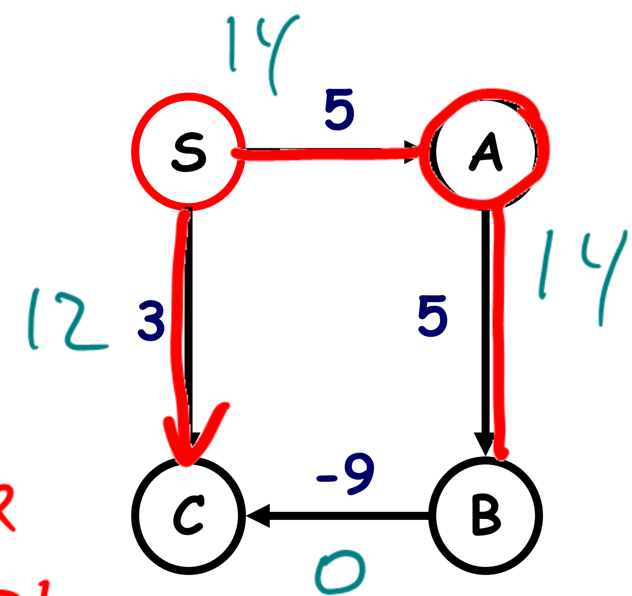
How to fix Dijkstra?

① Re-weight: add 9

Run Dijkstra: S-C

no, we did not get the original OPT solution

② DP



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.

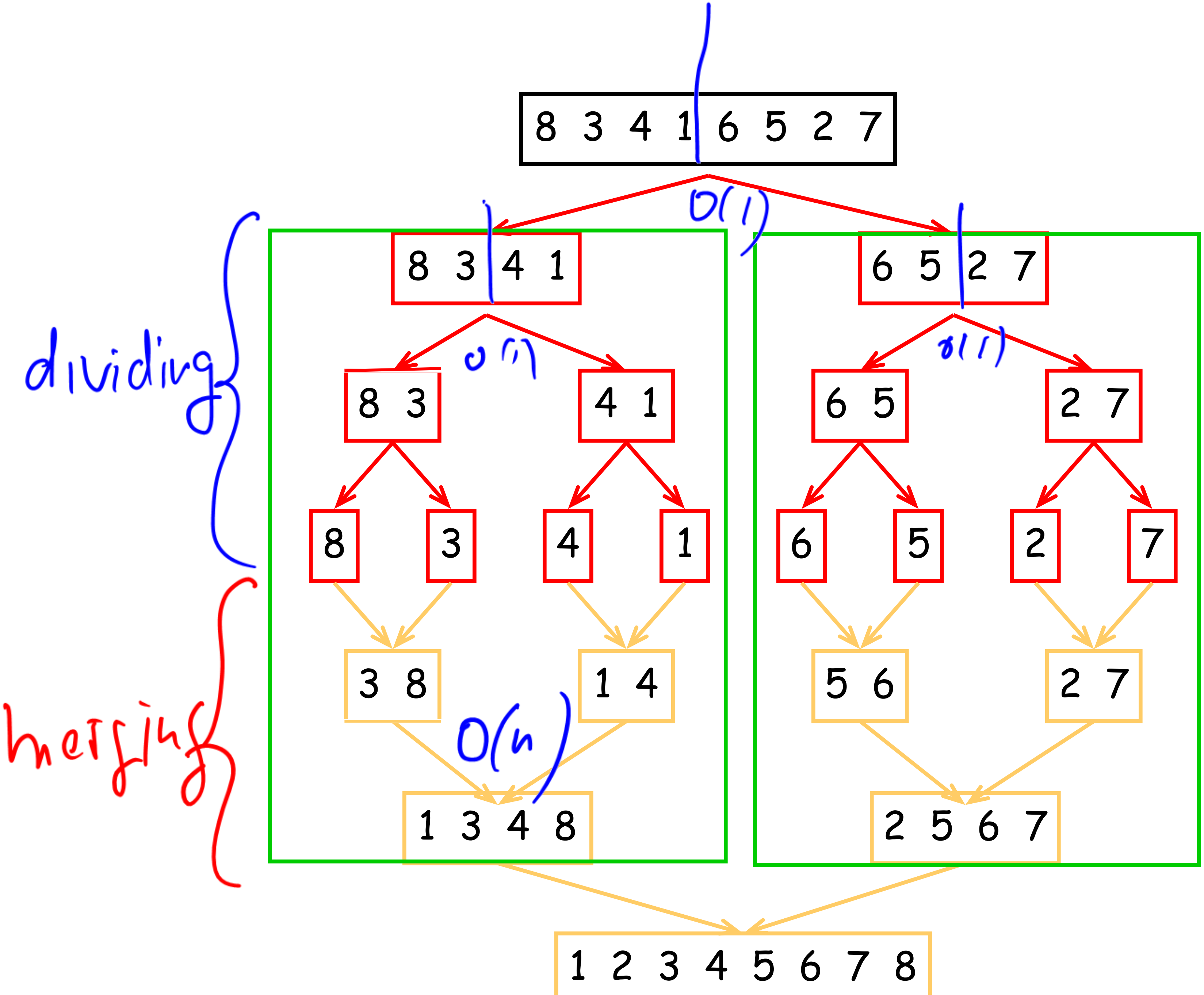
1 2 3 4 5 6 7 8

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



8 3 4 1 6 5 2 7
O(1)
8 3 4 1
6 5 2 7
o(1)
o(1)
dividing
8 3
4 1
6 5
2 7
8
3
4
1
6
5
2
7
merging
3 8
1 4
5 6
2 7
O(n)
1 3 4 8
2 5 6 7
1 2 3 4 5 6 7 8

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.

$O(\log n)$

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

$O(n \log n)$

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

# of comparisons

merge two sorted arrays

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

subproblems

merge

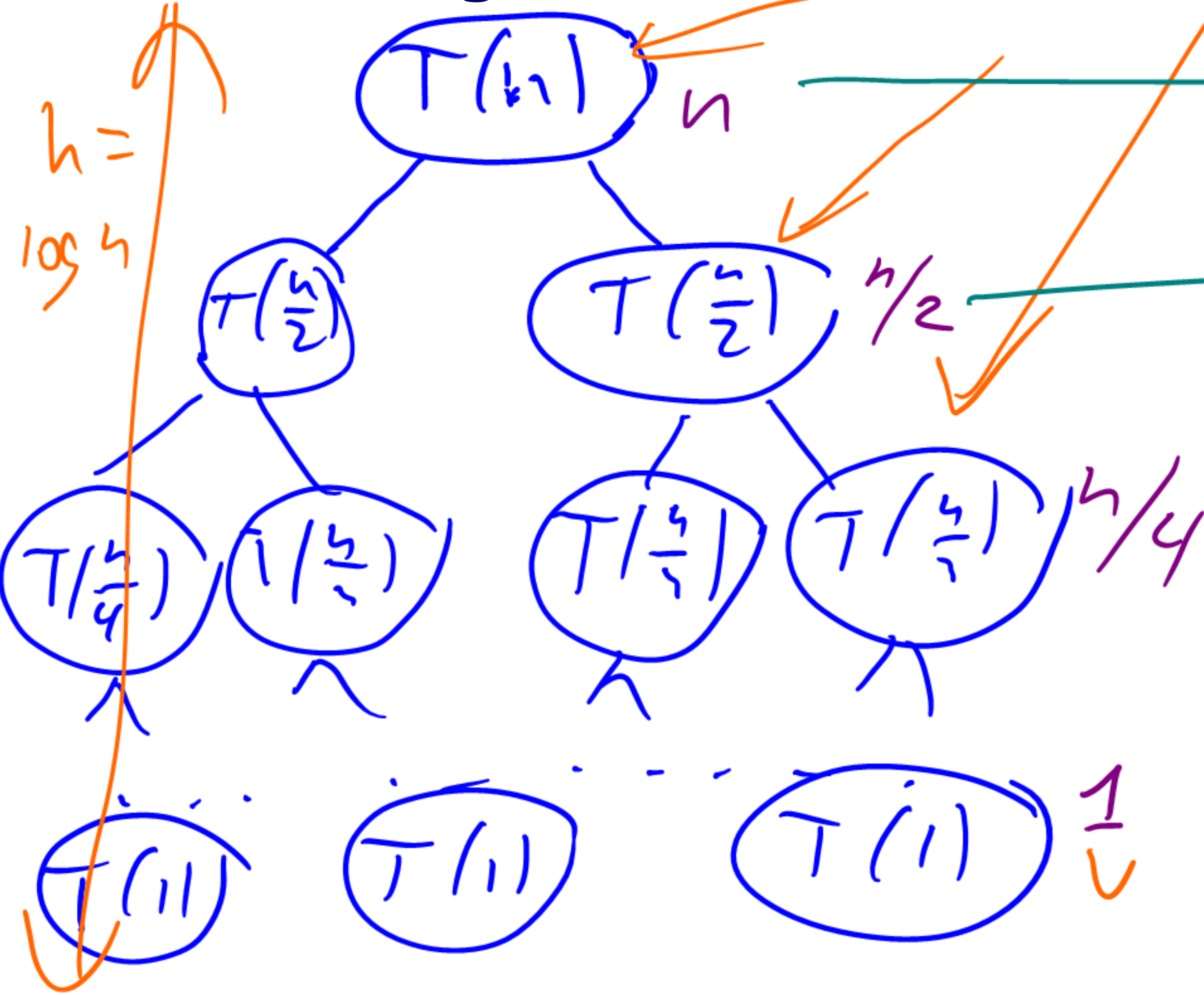
$$\boxed{T(n) = a \cdot T(n/b) + f(n)}$$

input size

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

$T(n) = 2 \cdot T(n/2) + O(n)$

Mergesort: tree of recursive calls



n

$\frac{n}{2} + \frac{n}{2} = n$

$\frac{n}{4} + \frac{n}{4} + \frac{n}{4} + \frac{n}{4} = n$

$\sum$ leaves $= n$

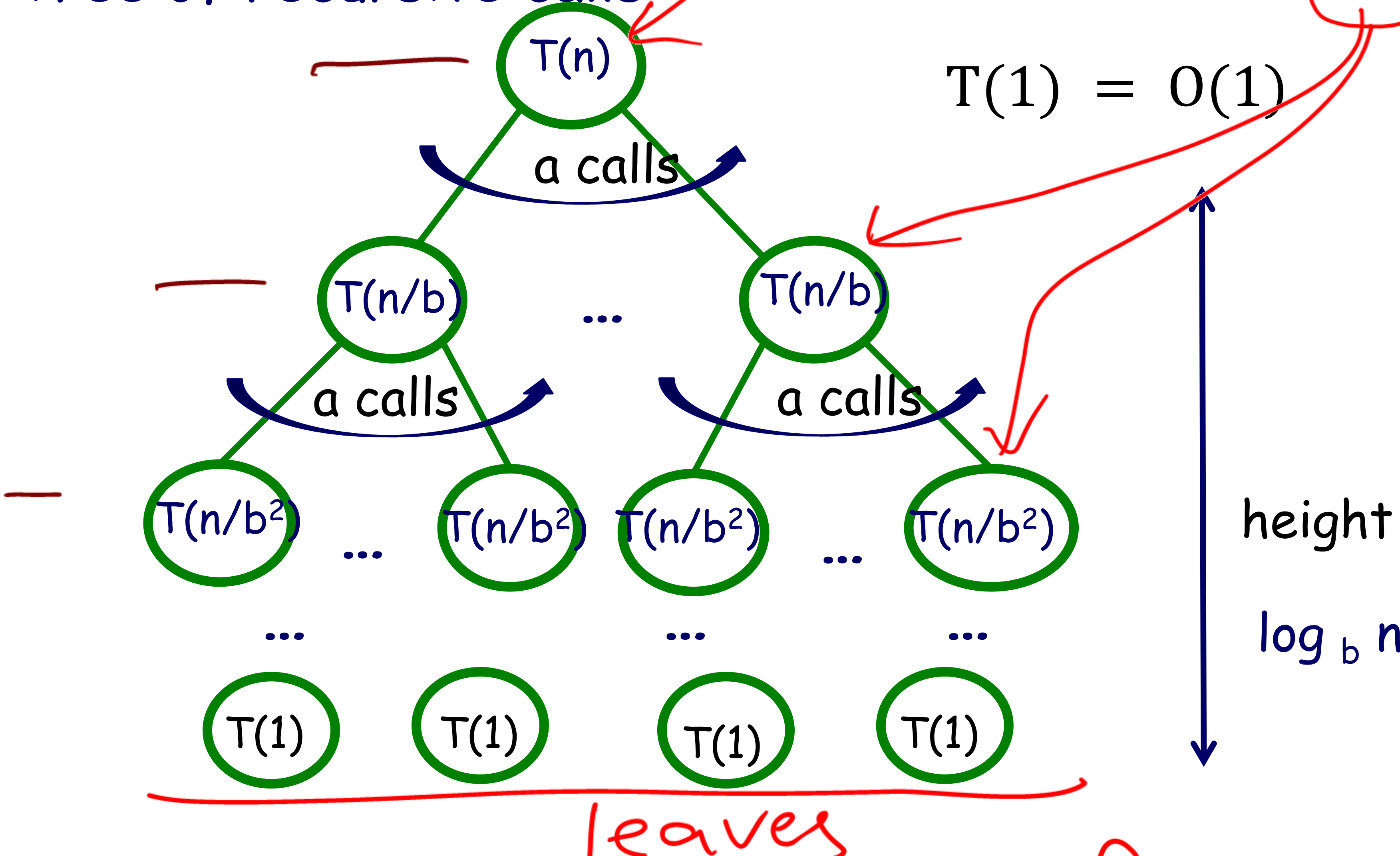
sum it up

$n + n + \dots + n = n \log n$

Tree of recursive calls

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

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



how many leaves?

Counting leaves

$h = \text{height}$

$h = \log_b n$

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

how many leaves?

$$a^h = a^{\log_b n} = a^{\frac{\log_a n}{\log_a b}} =$$

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

$$= n^c$$

$$c = \log_b a$$

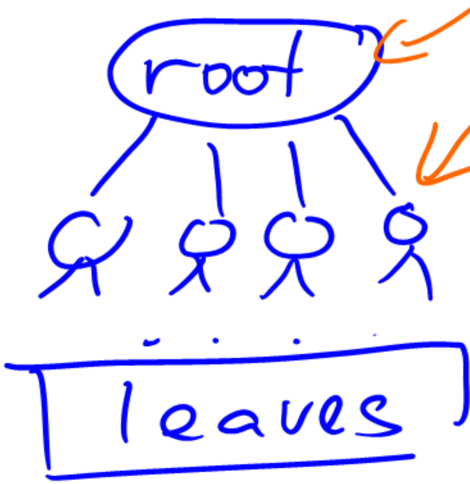
The Master Theorem

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

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

internal nodes

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is a <u>positive</u> function.



internal nodes, $O(f(n))$

$O(1)$

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

The Master Theorem

$$T(n) = a \cdot T(n/b) + f(n), \quad a \geq 1 \text{ and } b > 1$$

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:

$f(n)$

a

$$T(n) = \underbrace{16}_{a}\ T(n/\underbrace{4}_{b}) + \overbrace{5\,n^3}^{f(n)}$$

b

$c = \log_b a = 2$

leaves $= O(n^c) = O(n^2)$

$f(n) = O(n^3)$

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

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

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 $c = \log_b a$.

Discussion Problem 6

Solve the recurrence by the Master Theorem:

a=3, b=3, c=1, case 1

1. $A(n) = 3\ A(n/3) + 15$

$A(n) = \Theta(n)$

a=4, b=2, c=2, case 3

2. $B(n) = 4\ B(n/2) + n^3$

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

case 2, k=0

3. $C(n) = 4\ C(n/2) + n^2$

$C(n) = \Theta(n^2 \log n)$

4. $D(n) = 4\ D(n/2) + n$

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

Integer Multiplication

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

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

$$= x_1 \cdot 10^{n/2} + x_0$$

$$15451|7766 = \underbrace{15451}_{x_1} \cdot 10^4 + \underbrace{7766}_{x_0}$$

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

expand

? O(1)

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

```
      1234
  ×   1111
  --------
      1234
     1234
    1234
  + 1234
  --------
   1370974
```

$O(n^2)$

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

additions

By Master theorem: $T(n) = \Theta(n^2)$

1961

Karatsuba's algorithm

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

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

new multipl.

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

expand

$$= \cancel{x_0 y_0} + x_0 y_1 + x_1 y_0 + \cancel{x_1 y_1} - \cancel{x_0 y_0} - \cancel{x_1 y_1}$$

$$= \underbrace{x_1 \cdot y_0}_{O(1)} + \underbrace{x_0 \cdot y_1}_{O(1)}$$

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

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?

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

$$a \cdot b = (x_2 \cdot 10^{2n/3} + x_1 \cdot 10^{n/3} + x_0) \cdot (y_2 10^{2n/3} + y_1 \cdot 10^{n/3} + y_0) =$$

reduce 9 to 5

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

1962, Ph.D. Cook from MIT

$$T(n) = \Theta(n^{\log_3 5})$$

CPU - integer mutipl.
GPU - matrix mult.

Matrix Multiplication

NVIDIA

TPU - 3D matrix
AI-chip

$O(n^3)$

A B C = AxB

$O(n^2)$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

$(a_{11}, a_{12}) \cdot (b_{11}, b_{21}) \to O(n^2)$

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

Matrix Multiplication

The usual rules of matrix multiplication holds for
block matrices

$O(n^2)$

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

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

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

   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

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

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

reduce

Runtime:

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

1968

Strassen's Algorithm

how many addition? 18

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

$a_{21} b_{11} + a_{22} \cdot b_{21}$

$s_1 = (a_{12}-a_{22})(b_{21}+b_{22})$

$s_2 = (a_{11}+a_{22})(b_{11}+b_{22})$

$s_3 = (a_{11}-a_{21})(b_{11}+b_{12})$

$s_4 = (a_{11}+a_{12})b_{22}$

$s_5 = a_{11}(b_{12}-b_{22})$

$s_6 = a_{22}(b_{21}-b_{11})$

$s_7 = (a_{21}+a_{22})b_{11}$

10

It takes 7 multiplications

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

$T(n) = \Theta(n^{2.8})$

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

Fast Matrix Multiplication

1969, Strassen $O(n^{2.808})$.

1978, Pan $O(n^{2.796})$ $\leftarrow$!!

1979, Bini $O(n^{2.78})$ $\leftarrow$

1981, Schonhage $O(n^{2.548})$

1981, Pan $O(n^{2.522})$

1982, Romani $O(n^{2.517})$ NSA

1982, Coppersmith and Winograd $O(n^{2.496})$ $\leftarrow$ ~~library~~

1986, Strassen $O(n^{2.479})$

1989, Coppersmith and Winograd $O(n^{2.376})$ $\leftarrow$ library

2010, Stothers $O(n^{2.374})$

2011, Williams $O(n^{2.3728642})$

2014, Le Gall $O(n^{2.3728639})$

goal!

$n^2 \log n$

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

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

is the $\underbrace{\text{maximum}}_{\text{max}}$.

A_1 A_2

For example,

$$A = \{3, -4, \boxed{5, -2, -2, 6, -3, 5,} -3, 2\}$$

l r

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

Algorithm

$(l_1, r_1, max_1) = MSS(A_1)$; recursive

$(l_2, r_2, max_2) = MSS(A_2)$; recursive

$(l_3, r_3, max_3) = span(A_1 \cup A_2)$ iterative

return $MAX(max_1, max_2, max_3)$

Finding the Maximum Subsequence Sum (MSS)

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

max max

Span Max = 9

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