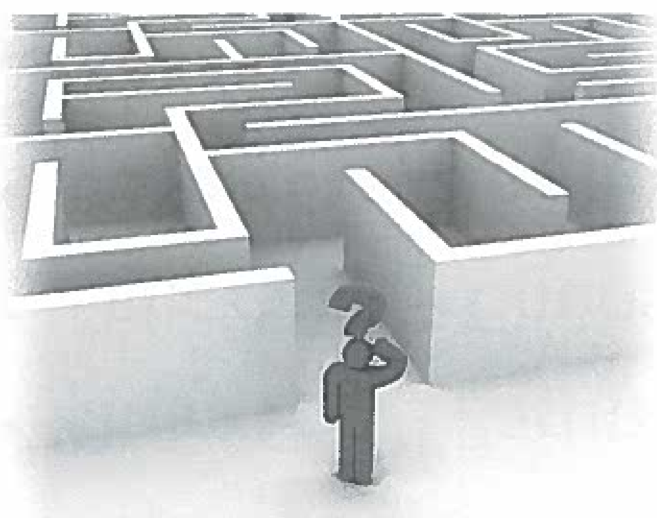
goal: problem of computing the ith smallest element of an input array (specifically median)
· By sorting, we can do it in $O(n \log n)$, but we can do better.

Monday: July 30, 2018

31

Linear-Time Selection

Week 4



Randomized Selection (Algorithm)

Design and Analysis of Algorithms I

We want to design a randomized algorithm that solves the problem of linear selection
· Expected running time: linear in terms of the size of array

select – 1

Prerequisites

Watch this after:

- QuickSort - Partitioning around a pivot
- QuickSort – Choosing a good pivot
- Probability Review, Part I

The Problem

For simplicity

Input : array A with n distinct numbers and a number

an integer btw 1 and n;

Output : i^{th} order statistic (i.e., i^{th} smallest element of input array)

10	8	2	4

3^{rd} order statistic

1^{st} order statistic is the minimum

n^{th} " " is the maximum.

Example : median. (middle element)

(i = (n+1)/2 for n odd,

i = n/2 for n even)

two possibilities here let's say we take the smaller ones.

Tim Roughgarden

Select_2

means solving another problem using a method you already know.

Reduction to Sorting

we know how sorting works,

This is what we know already

O(nlog(n)) algorithm

1) Apply MergeSort 2) return i^{th} element of sorted array

can we do better? why not linear

Fact : can't sort any faster [see optional video]

If we want to do better, we should use selection rather than the sorting.

Next : O(n) time (randomized) by modifying Quick Sort. to directly solve the selection problem.

Optional Video : O(n) time deterministic algorithm.

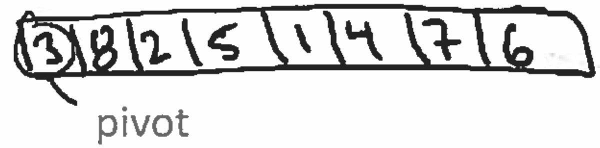
-- pivot = "median of medians" (warning : not practical)

This is the work horse for the selection algorithm

Partitioning Around a Pivot

<u>Key Idea</u> : partition array around a pivot element.

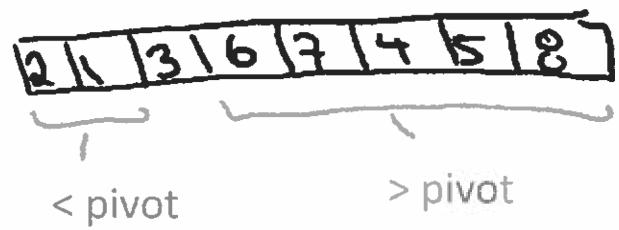
-Pick element of array



-Rearrange array so that

-Left of pivot => less than pivot

-Right of pivot => greater than pivot



<u>Note</u> : puts pivot in its "rightful position".

Tim Roughgarden

Ter
ver

Suppose we are looking for the 5th order statistic in an input array of length 10. We partition the array, and the pivot winds up in the third position of the partitioned array. On which side of the pivot do we recurse, and what order statistic should we look for?

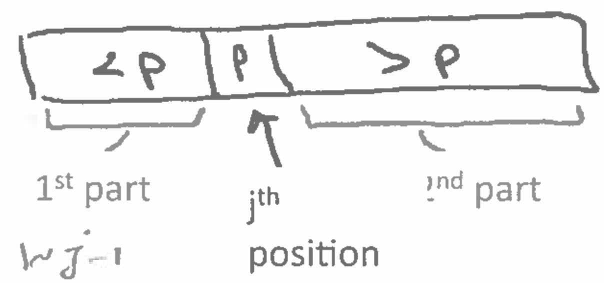
- ○ The 3rd order statistic on the left side of the pivot.
- ○ The 2nd order statistic on the right side of the pivot.
- ○ The 5th order statistic on the right side of the pivot.
- ○ Not enough information to answer question – we might need to recurse on the left or the right side of the pivot.

Randomized Selection

Rselect (array A, length n, order statistic i)

0) if n = 1 return A[1]
1) Choose pivot p from A uniformly at random
2) Partition A around p
   let j = new index of p
3) If j = i, return p *if we are lucky.*
4) If j > i, return Rselect(1st part of A, j-1, i) *new array size*
5) [if j<i] return Rselect (2nd part of A, n-j, i-j)

new order



Tim Roughgarden

Select - 4

Properties of RSelect

<u>Claim</u> : Rselect is correct (guaranteed to output ith order statistic)

<u>Proof</u> : by induction. [like in optional QuickSort video]

<u>Running Time ?</u> : depends on “quality” of the chosen pivots.

Ter
ver

Similar to the worst case scenario for the quick sort.

What is the running time of the RSelect algorithm if pivots are always chosen in the worst possible way?

- $\theta(n)$
- $\theta(n \log n)$
- $\theta(n^2)$ disaster
- $\theta(2^n)$

<u>Example</u> :

-- suppose i = n/2 (median)
-- suppose choose pivot = <u>minimum</u> every time
=> $\Omega(n)$ time in each of $\Omega(n)$ recursive calls

you need to recurse $\frac{n}{2}$ times each time on an array of size $n-i$

select_5

Running Time of RSelect?

Running Time ? : depends on which pivots get chosen.
(could be as bad as $\theta(n^2)$)

To have faster algo., any time we ~~recurse~~, the problem size should be smaller by a significant amount. So when we recurse we make a lot of progress.

Key : find pivot giving "balanced" split.

what we are looking for

Best pivot: the median ! (but this is circular)

gives us a 50 50 split

the running time for the best case.

$\Rightarrow$Would get recurrence $T(n) \leq T(n/2) + O(n)$

$\Rightarrow$ T(n) = O(n) [case 2 of Master Method] $a=1,\ b=2,\ d=1 \Rightarrow a < 2^1 = b^d \Rightarrow O(n^d) = O(n)$.

Hope : random pivot is "pretty good" "often enough"

gives the median

Tim Roughgarden

Running Time of RSelect

Rselect Theorem : for every input array of length n, the average running time of Rselect is O(n)

-- holds for every input [no assumptions on data]

-- "average" is over random pivot choices made by the algorithm

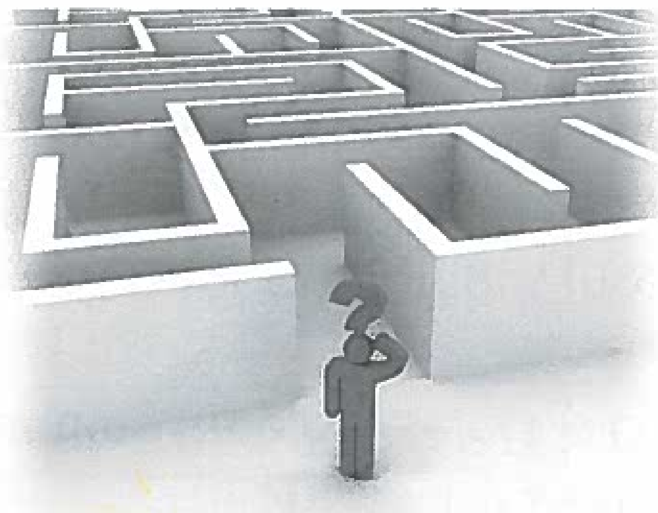
randomness is not in the data rather in the code.

general purpose subroutine

select_6

(8) 32

July 31, 2018



Design and Analysis of Algorithms I

Linear-Time Selection

Randomized Selection (Analysis)

Random – 1

Running Time of RSelect

Rselect Theorem : for every input array of length n, the average running time of Rselect is $O(n)$ In fact this is faster than sorting.

thus, we dont need to reduce it to sorting

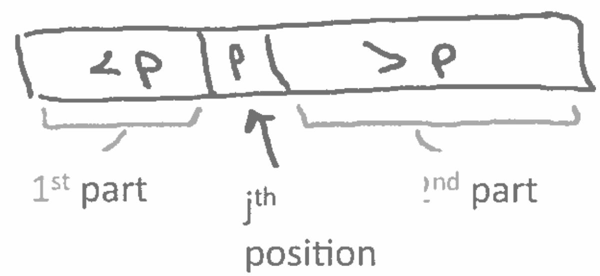
-- holds for every input [no assumptions on data]

-- "average" is over random pivot choices made by the algorithm

The algorithm is similar to quicksort, but we recurse once.

Randomized Selection

Rselect (array A, length n, i^{th} order statistic ~~i~~)



0) if n = 1 return A[1]
1) Choose pivot p from A uniformly at random
2) Partition A around p
   let j = new index of p
3) If j = i, return p
4) If j > i, return Rselect(1^{st} part of A, j-1, i)
5) [if j<i] return Rselect (2^{nd} part of A, n-j, i-j)

why

work outside the recursive call is O(n) ≡ cn.

- we cannot use the indicators that we had for analysing quicksort running time.
- The workhorse for this randomized selection procedure is exactly the same as quick sort.
- All the work outside the recursive calls just partitions around pivot elements = O(n)

Proof I: Tracking Progress via Phases

Note : Rselect uses <= cn operations outside of recursive call [for some constant c > 0] [from partitioning]

- we want to have a big win over the quicksort, as we only have one recursive call.

Notation : Rselect is in phase j if current array size between $(\frac{3}{4})^{j+1} \cdot n$ and $(\frac{3}{4})^{j} \cdot n$ ≡ higher number of phases ≡ more progress.

phase j quantifies # of times we've made 75% progress relative to original input array

-X_j = number of recursive calls during phase j

# of phase j subproblems

Note : running time of RSelect $\leq \sum_{phases\ j} X_j \cdot c \cdot \boxed{(\frac{3}{4})^{j} \cdot n}$

a random variable

random variable

<= array size during phase j

Work per phase j subproblem

Tim Roughgarden

→ Ex- phase 0^j = array size is btw n ~ 0.75n. Depending on the choice of pivot you may get out of phase 0, in the next recursive call. If not in 0.75n, you are still in phase 0, though you are doing the second recursive call.

Proof II: Reduction to Coin Flipping

X_j = # of recursive calls during $\boxed{\text{phase } j}$ → Size between $(\frac{3}{4})^{j+1} \cdot n$ and $(\frac{3}{4})^{j} \cdot n$

Note : if Rselect chooses a pivot giving a 25 – 75 split (or better) then current phase ends ! ✓

(new subarray length at most 75 % of old length) ✓

$\boxed{<p \mid p \mid >p}$

Recall : probability of 25-75 split or better is 50%

now, this is equivalent to flipping coins.

So : $E[X_j]$ <= expected number of times you need to flip a fair coin to get one "heads"

(heads ~ good pivot, tails ~ bad pivot)

you stop when you get the first tail.

Tim Roughgarden

Random – 3

Proof III: Coin Flipping Analysis

Let N = number of coin flips until you get heads.

(a "geometric random variable")

– writing the expected value of N in terms of itself, and solve for itself.

great ⇒ Note : $E[N] = 1 + (1/2)*E[N]$

1st coin flip — Probability of tails — # of further coin flips needed in this case

⇒ solve for E[N])

head

Solution : $E[N] = 2$ (Recall $E[X_j] <= E[N]$) ✓

Putting It All Together

Expected running time of RSelect

$$\leq E[cn \sum_{phase\ j} (\frac{3}{4})^j X_j] \qquad (*)$$

$$= cn \sum_{phase\ j} (\frac{3}{4})^j \boxed{E[X_j]} \qquad \text{[LIN EXP]}$$

= E[# of coin flips N] = 2

$$\leq 2cn \sum_{phase\ j} \boxed{(\frac{3}{4})^j}$$

geometric sum,
<= 1/(1-3/4) = 4

$$\leq 8cn = O(n)$$ ✓

Q.E.D.

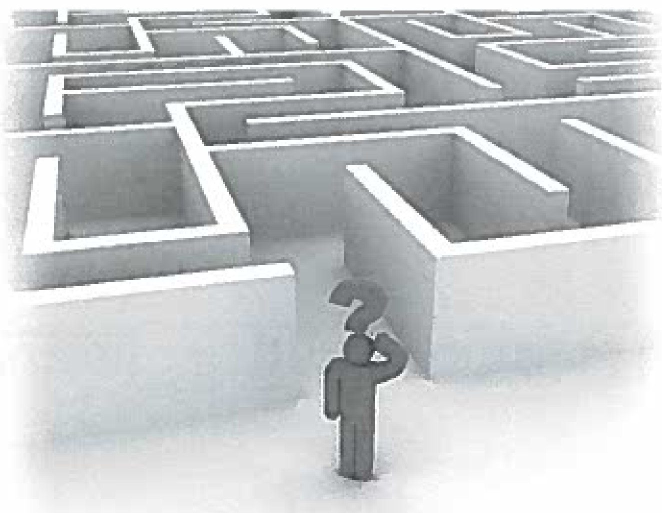
Tim Roughgarden

Random 4

33

~~[illegible]~~ August 1, 2018

we'll cover another algorithm for the selection problem in this lecture. the already discussed algorithm is fast enough, $O(n)$, but this new algorithm is cool enough not skip it – this is deterministic, no RANDOMIZATION what so ever and runs in $O(n)$.



Linear-Time Selection

Deterministic Selection (Algorithm)

Design and Analysis of Algorithms I

This algo, though is not as fast as RSelect in practice, bcs its ① hidden costs are larger, ② it does not operate in place

DS-1

The Problem

For simplicity

<u>Input</u> : array A with n distinct numbers and a number

<u>Output</u> : i^{th} order statistic (i.e., i^{th} smallest element of input array)

10	8	2	4

3^{rd} order statistic

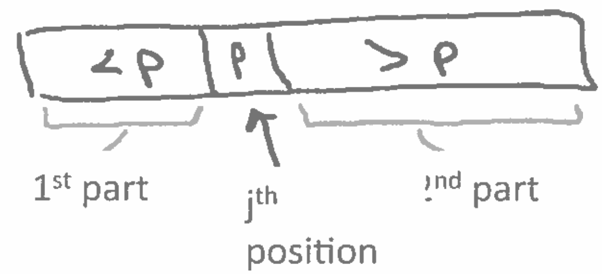
<u>Example</u> : median.

($i = (n+1)/2$ for n odd,
$i = n/2$ for n even)

Review the

Randomized Selection

Rselect (array A, length n, order statistic i)



0) if n = 1 return A[1]
1) Choose pivot p from A uniformly at random
2) Partition A around p
   let j = new index of p
3) If j = i, return p   very rare
4) If j > i, return Rselect(1st part of A, j-1, i)
5) [if j<i] return Rselect (2nd part of A, n-j, i-j)

Random will give us a pretty good split

Tim Roughgarden

Q: What if Random Pivot is not in our toolbox?
Then, the question is how we deterministically choose a good pivot. DS _2

Guaranteeing a Good Pivot

$\Rightarrow$ 50–50 split

Recall : "best" pivot = the median ! (seems circular!)

Now, we need a subroutine that deterministically,

Goal : find pivot guaranteed to be pretty good.

Key Idea : use "median of medians"!

this is the new implementation of choose pivot

A Deterministic ChoosePivot

ChoosePivot(A,n)

If n is not a multiple of 5, one group has size btw 1~4

-- logically break A into "n/5 groups" of "size 5 each"

-- sort each group (e.g., using Merge Sort) = and use the median, middle element

-- copy n/5 medians (i.e., middle element of each sorted group) into new array C

-- recursively compute median of C (!) second round of the tournament

-- return this as pivot

Tim Roughgarden

DS-3

Deterministic Selection

The DSelect Algorithm

Let's leave the base case

DSelect(array A, length n, order statistic i)

inline

1. Break A into groups of 5, sort each group
2. C = the n/5 "middle elements"
3. p = DSelect(C,n/5,n/10) [recursively computes median of C]
4. Partition A around p
5. If j = i return p
6. If j < i return DSelect(1^{st} part of A, j-1, i)
7. [else if j > i] return DSelect(2nd part of A, n-j, i-j)

ChoosePivot

length of array C

$(\frac{n}{10})^{th}$ order statistic

Same as before

- No infinite loop on recursive calls bcs the size of the array gets smaller and smaller.
- We will prove that this algo will terminate, in finite time, and it actually runs in linear time.

Te
ve

How many recursive calls does DSelect make?

- 0
- 1
- 2
- 3

One recursive call in line 3 of the Pseudo code
One recursive call in line (6 or 7).

In RSelect, we had only one recursive call

DS-4

Running Time of DSelect

Dselect Theorem : for every input array of length n, Dselect runs in O(n) time.

The constants in the O notation are larger compared to Rselect

Warning : not as good as Rselect in practice

1) Worse constraints

2) not-in-place, we need extra memory

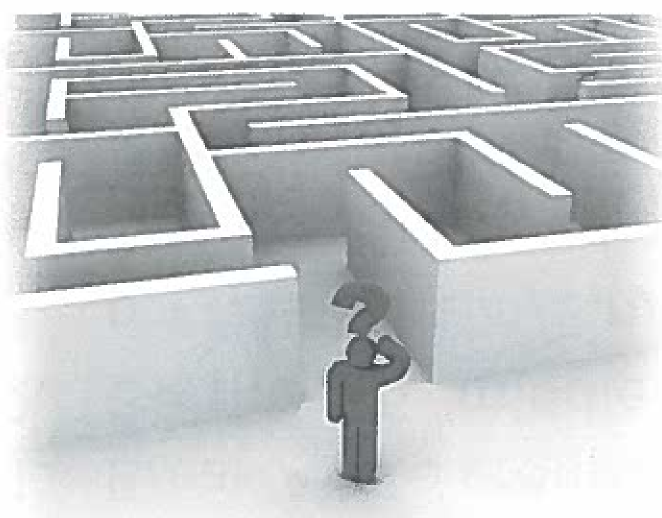
History : from 1973

Blum – Floyd – Pratt – Rivest – Tarjan

('95) ('78) ('02) ('86)

Aug 2, 2018

34



Design and Analysis
of Algorithms I

Linear-Time Selection

Deterministic Selection (Analysis)

DSA – 1

- In this case there two recursive calls, we haven't seen an algo with two recursive calls that runs in linear time, the best case is $O(n \log n)$.
- Outside the recursive call, we have a lot of work

The DSelect Algorithm

DSelect(array A, length n, order statistic i)

1. Break A into groups of 5, sort each group
2. C = the n/5 "middle elements"
3. p = DSelect(C,n/5,n/10) [recursively computes median of C]
4. Partition A around p
5. If j = i return p
6. If j < i return DSelect(1st part of A, j-1, i)
7. [else if j > i] return DSelect(2nd part of A, n-j, i-j)

Choose Pivot

Same as before

Tim Roughgarden

The cost of MergeSort is $O(n \log n)$. But, here we do merge-sort on small arrays. (of size 5). that is why the total cost here is $O(n)$.

Ter
ver

What is the asymptotic running time of step 1 of the DSelect algorithm?

- $\theta(1)$
- $\theta(\log n)$
- $\theta(n)$
- $\theta(n \log n)$

Note : sorting an array with 5 elements takes <= 120 operations

[why 120 ? Take m = 5 in our $6m(\log_2 m+1)$ bound for Merge Sort]

$6*5*(\log_2 5+1) <= 120$

So : <= $\underbrace{\boxed{(n/5)}}_{\text{\# of gaps}} * \underbrace{\boxed{120}}_{\text{ops per group}}$ = 24n = O(n) for all groups

DSA_2

We have two costs here: one recursive calls, the other one is local ones outside of the recursive

The DSelect Algorithm

DSelect(array A, length n, order statistic i)

1. Break A into groups of 5, sort each group $\theta(n)$
2. C = the n/5 "middle elements" $\theta(n)$
3. p = DSelect(C,n/5,n/10) [recursively computes median of C] $T(\frac{n}{5})$
4. Partition A around p $\theta(n)$
5. If j = i return p → constant time
6. If j < i return DSelect(1^{st} part of A, j-1, i)
7. [else if j > i] return DSelect(2nd part of A, n-j, i-j)

$T(?)$

length of array C

n^{th} order statistic

recursive call. In this case we don't know how big the recursive call is.

The size of the array that is passed to the recursive call depends on how good the pivot is

Tim Roughgarden

Rough Recurrence

Let $T(n)$ = maximum running time of Dselect on an input array of length n.

There is a constant $c >= 1$ such that :

to be explicit about the work outside the recursive call, rather than O(.) notation, we would like to be precise

1. $T(1) = 1$
2. $T(n) <= c*n + T(n/5) + T(?)$

sorting the groups
partition

recursive
call in line 3

recursive call in
line 6 or 7

Tim Roughgarden

The Key Lemma

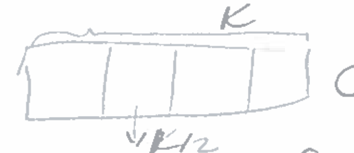
Key Lemma : 2^{nd} recursive call (in line 6 or 7) guaranteed to be on an array of size <= 7n/10   (roughly)

Ex: $n=20 \Rightarrow k=\frac{n}{5}=4$

→ the median

Upshot : can replace "?" by "7n/10"

assume k is even:



median of medians.

Rough Proof : Let k = n/5 = # of groups

Let x_i = i^{th} smallest of the k "middle elements" ← i^{th} smallest among the middle elements

[So pivot = $x_{k/2}$] ✓

We want to show that for this pivot we definitely get a 30–70 % split

i.e. Goal : >= 30% of input array smaller than $x_{k/2}$, → then we work with right side

or >= 30% is bigger → then we work with the left side of array.

Tim Roughgarden

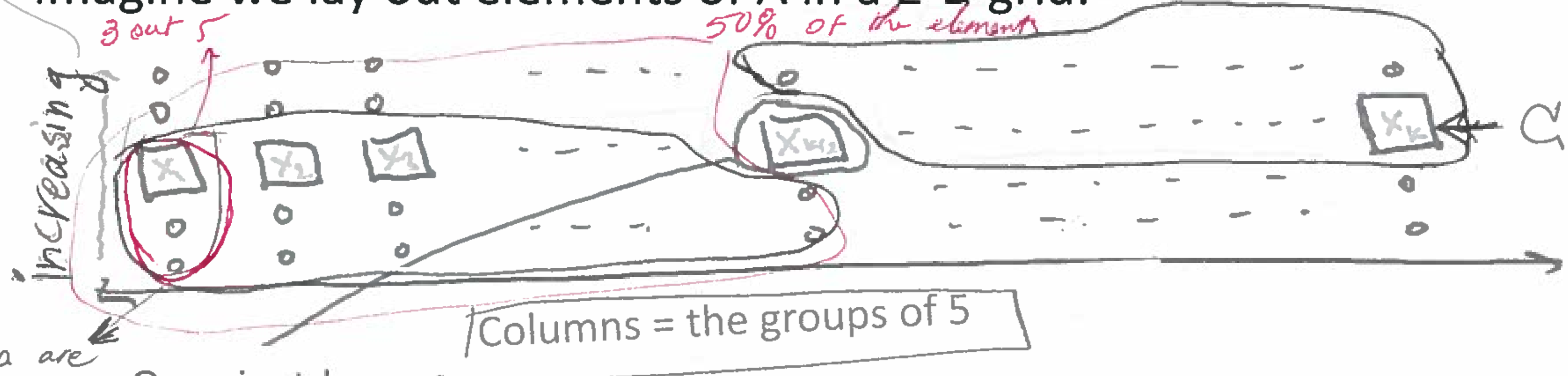
NOTE: we arrange the groups of 5 in the order of increasing middle element.
thus we have $x_1, x_2, \ldots$

Rough Proof of Key Lemma

Thought Experiment :

Imagine we lay out elements of A in a 2-D grid:

If n is not a multiple of 5 then one group is having 5 elements



Key point : $x_{k/2}$ bigger than 3 out of 5 (60%) of the elements in ~ 50% of the groups ✓

=> bigger than 30% of A (similarly, smaller than 30% of A)

Tim Roughgarden

$\frac{3}{5} \times \frac{1}{2} = \frac{3}{10} \sim 30\%$

DSA – 4

Example

Input   groups of 5 $k = \frac{20}{5} = 4$ $n = 20$ elements

7	2	17	12	13	8	20	4	6	3	19	1	9	5	16	10	15	18	14	11

After sorting groups of 5   middle elements

2	7	12	13	17	3	4	6	8	20	1	5	9	16	19	10	11	14	15	18

Our pivot !

The grid :

20	14	17	18
8	16	13	15
6	9	12	14
4	5	7	11
3	1	2	10

bigger than middle element.

x_1 $x_2 = k/2$

this region is smaller than 9;

Tim Roughgarden

(35)

Aug 2, 2018;

Design and Analysis of Algorithms I

Linear-Time Selection

Deterministic Selection (Analysis II)

DSA II 1

Rough Recurrence (Revisited)

Let $T(n)$ = maximum running time of Dselect on an input array of length n.

There is a constant $c \geq 1$ such that :

1. $T(1) = 1$
2. $T(n) \leq c*n + T(n/5) + T(?)$

$\leq 7n/10$ by Key Lemma

sorting the groups partition

recursive call in line 3

recursive call in line 6 or 7

Rough Recurrence (Revisited)

$T(1) = 1$, $T(n) \le cn + \underbrace{T(n/5) + T(7n/10)}_{\text{2 recursive calls.}}$

Constant $c \ge 1$

Note : different-sized subproblems => can't use Master Method!

Strategy : "hope and check"

we hope it is a linear time, and then we prove that

Hope : there is some constant a [independent of n]

Such that $T(n) \le an$ for all $n \ge 1$

[if true, then $T(n) = O(n)$ and algorithm is linear time]

Tim Roughgarden

DSA II_2

Analysis of Rough Recurrence

reverse engineering, wait until end.

T(1) = 1 ; T(n) <= cn + T(n/5) + T(7n/10)

Constant c>=1

Claim : Let a = 10c

Then $\boxed{\text{T(n) <= an for all n >= 1}}$

=> Dselect runs in O(n) time

Proof : by induction on n

Base case : T(1) = 1 <= a*1   a >= 1)

Inductive Step : [n > 1]

Inductive Hypothesis : $T(k) \leq ak \ \forall \ k < n$

We have $T(n) \underset{\text{GIVEN}}{\leq} cn + T(n/5) + T(7n/10)$

$$\underset{\text{IND HYP}}{\leq} cn + a(n/5) + a(7n/10)$$

$$= n(c + 9a/10) \underset{\text{Choice of a}}{=} an$$

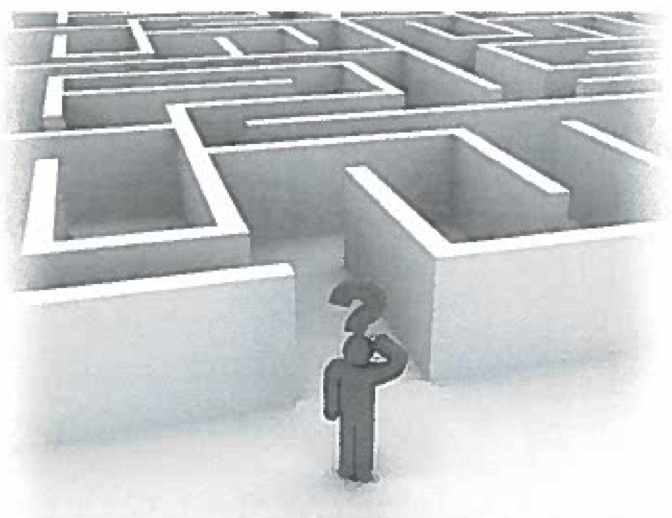
Q.E.D.

Tim Roughgarden

"not in the book"

Aug 3, 2018

36



Design and Analysis of Algorithms I

Linear-Time Selection

An $\Omega(n \log n)$ Sorting Lower Bound

Can we do better than $O(n \log n)$ for sorting?

For example linear time

Didn't get it

Seit 1

A Sorting Lower Bound

Theorem : every "comparison-based" sorting algorithm has worst-case running time $\Omega(n \log n)$ ≡ we cannot do better.

this assumption is not necessary.

[assume deterministic, but lower bound extends to randomized]

Comparison-Based Sort : accesses input array elements only via comparisons ~ "general purpose sorting method" ≡ not direct manipulation on a single element.

Examples : Merge Sort, Quick Sort, Heap Sort

Good for data from distributions

good for small integers

good for medium-size integers

Non Examples : Bucket Sort, Counting Sort, Radix Sort

Tim Roughgarden

Proof Idea

Fix a comparison-based sorting method and an array length n

⇒Consider input arrays containing {1,2,3,...,n} in some order.

⇒ "n!" such inputs (different ordering possibilities.)

Suppose algorithm always makes <= k comparisons to correctly sort these n! inputs.

=> Across all n! possible inputs, algorithm exhibits <= 2^k distinct executions i.e., resolution of the comparisons

not clear

Tim Roughgarden

Sort_2

Proof Idea (con'd)

By the Pigeonhole Principle : if $2^k < n!$, execute identically on two distinct inputs => must get one of them incorrect.

So : Since method is correct,

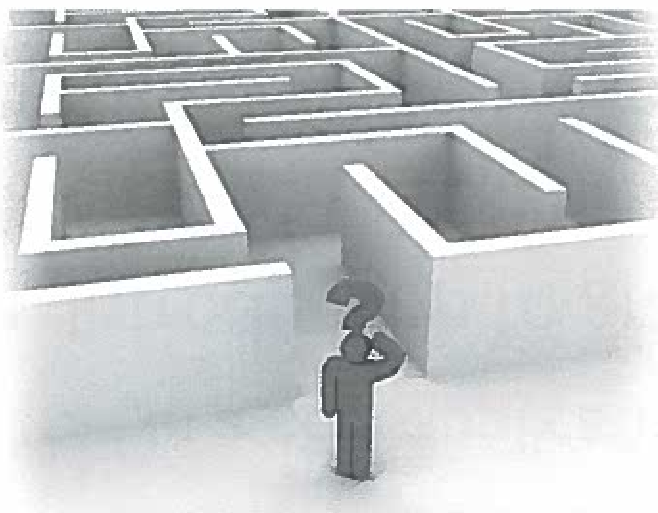
$$2^k \geq n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$$\Rightarrow k \geq \frac{n}{2} \cdot \log_2 \frac{n}{2} = \Omega(n \log n)$$

Aug 3, 2018 39

Contraction , becoming smaller or narrower.
Algorithm

Overview



Design and Analysis
of Algorithms I

graph-1

We study Graphs bcs you can define Data structure based on graphs. Ex: Tree is a graph with n vertices and $n-1$ edges.

Goals for These Lectures

- Further practice with randomized algorithms
  - In a new application domain (graphs)
- Introduction to graphs and graph algorithms

The contraction algo is kind of new.

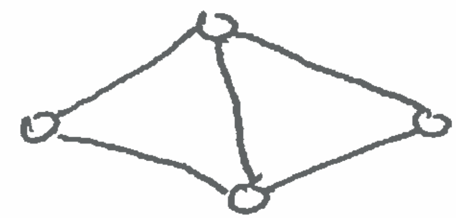
Also: “only” 20 years ago!

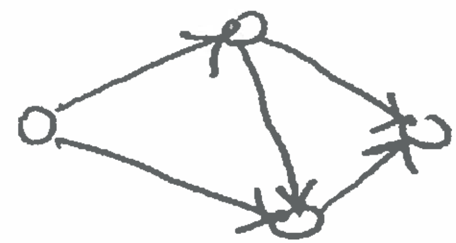
A graph represents pairwise relationships among a set of objects,

See algorithm 2;

Graphs

Two ingredients
- Vertices aka nodes (V)
- Edges (E) = pairs of vertices
  - can be undirected [unordered pair] or directed [ordered pair] (aka arcs)





Examples: road networks, the Web, social networks, precedence constraints, etc.

courses in a program.

Tim Roughgarden

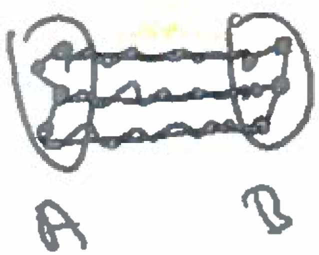
graph - 2

After partitioning we have edges with both end points in one partition, and edges with both end points on two different partitions.

Cuts of Graphs

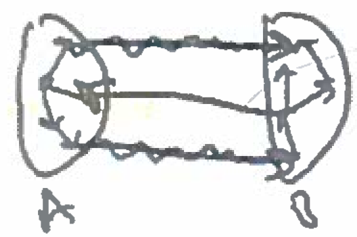
Definition: a cut of a graph (V, E) is a partition of V into two non-empty sets A and B.

3 cuts



[undirected]

2 cuts.



not a cut.

[directed]

Definition: the crossing edges of a cut (A, B) are those with:

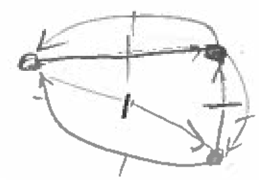
- the one endpoint in each of (A, B) [undirected]
- tail in A, head in B [directed]

from the left to the right.

Tim Roughgarden

Location of vertices:

Vertices	1	2	3	4	
1	0	0	0	0	: All vertices in A (-1)
2	0	0	0	1	
3	0	0	1	0	
4	0	1	0	0	
5	1	0	0	0	
6	1	1	0	0	
7	1	0	1	0	
8	1	0	0	1	
9	1	1	1	0	
10	1	1	0	1	
11	1	1	1	0	
12	1	1	1	1	(-1) All vertices in B
13	0	1	1	0	
14	0	1	0	1	
15	0	0	1	1	
✓ 16	0	0	1	1	



$2^3 - 2 = 6$.

directional

If vertix is in A $\rightarrow 0$

" " " " B $\rightarrow 1$

total partitions : $2^4 - 2 = 16 - 2 = 14$

Ter

ver

Roughly how many cuts does a graph with n vertices have? ≡ All possibles of partitioning a graph with n Vertic

See the definition of cut.

≡ partition

- n
- n^2
- 2^n -2
- n^n

For each node we have either 0 or 1.

cut / not cut

(two choices).

$\underbrace{2 \times 2 \times \cdots \times 2}_{n \times} = 2^n$ goods.

graph-3

The Minimum Cut Problem imp:

- INPUT: An undirected graph $G = (V, E)$.

[Parallel  edges allowed]

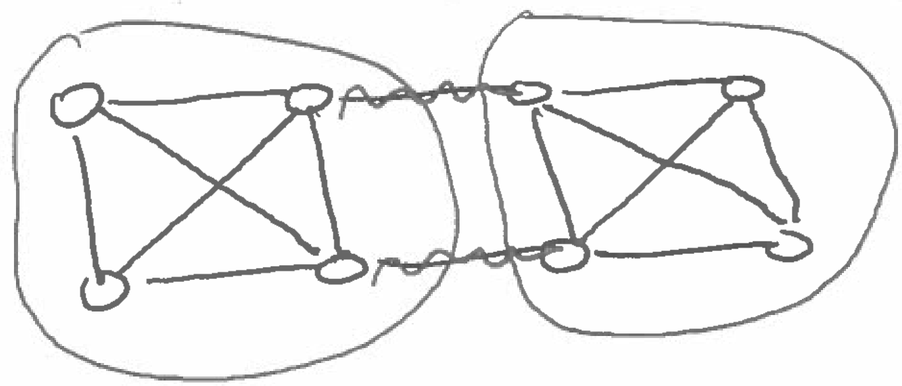
[See other video for representation of the input]

imp

- GOAL: Compute a cut with fewest number of crossing edges. (a min cut)

Te
ve

What is the number of edges crossing a minimum cut in the graph shown below?



- 1
- 2
- 3
- 4

graph-4

In coldwar, b/w US and Soviet U, how to cut roads to disrupt transportation

A Few Applications

- indentify network bottlenecks / weaknesses → min cut problem ⊛
- community detection in social networks → from the same school / region / same interest / family / ≡ regions that are highly interconnected, weakly connected to the rest of graph
- image segmentation
  - input = graph of pixels
  - use edge weights

    [(u,v) has large weight ⟺ "expect" u,v to come from some object]

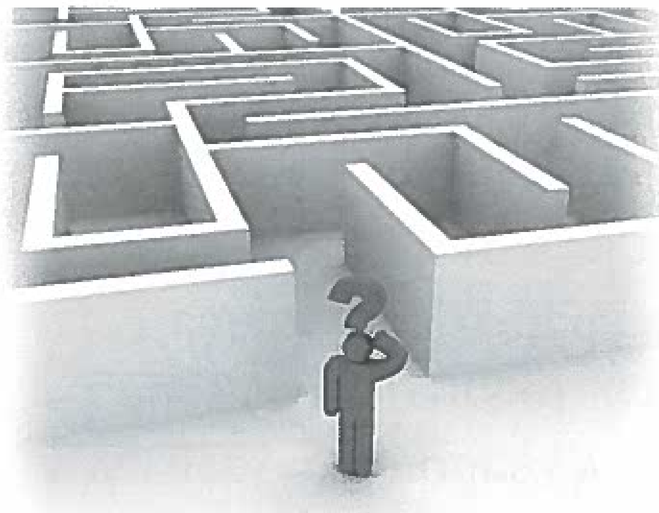
  hope: repeated min cuts identifies the primary objects in picture.

38

See chapter 1 of the Second book.

Graph Algorithms

Representing Graphs

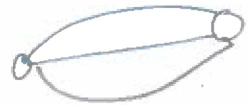


Design and Analysis of Algorithms I

rep_1

there are two parameters that indicate how big a graph is. ① # $V = n$
② # $E = m$

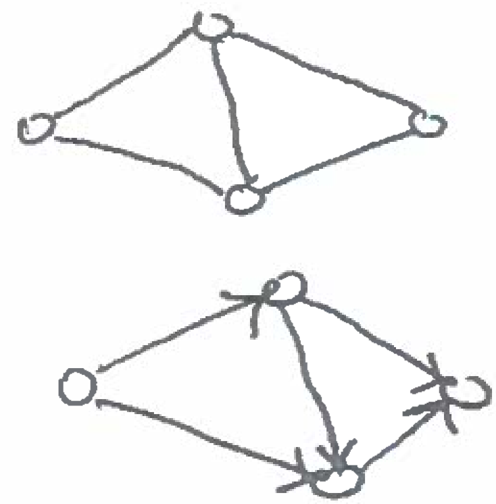
parallel edges



Graphs

<u>Two ingredients</u>

- <u>Vertices</u> aka nodes (V)
- <u>Edges</u> (E) = pairs of vertices
  - can be <u>undirected</u> [unordered pair] or <u>directed</u> [ordered pair] (aka <u>arcs</u>)

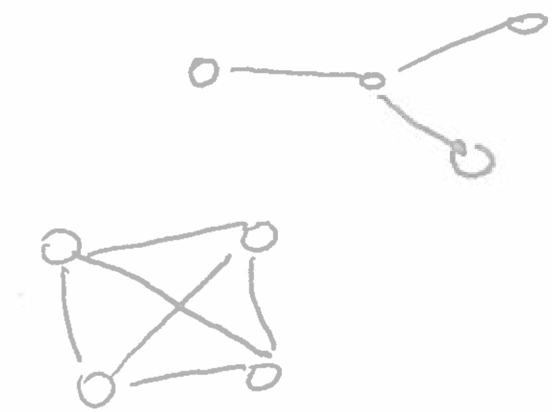


<u>Examples</u>: road networks, the Web, social networks, precedence constraints, etc.

Ter
ver

Consider an undirected graph that has n vertices, no parallel edges, and is connected (i.e., “in one piece”). What is the minimum and maximum number of edges that the graph could have, respectively ?

- ○ $n - 1$ and $n(n-1)/2$
- ○ $n - 1$ and n^2
- ○ n and 2^n
- ○ n and n^n



vep_2

Sparse vs. Dense Graphs

Let $\underline{n}$ = # of vertices, $\underline{m}$ = # of edges.

In most (but not all) applications, m is $\overbrace{\Omega(n)}^{\text{at least}}$ and $\overbrace{O(n^2)}$ if no parallel edges.

- in a "sparse" graph, m is or is close to $O(n)$
- in a "dense" graph, m is closer to $\theta(n^2)$

Some algorithm are good for sparse graphs, some others for dense

Two ways to represent a graph:

1 – The Adjacency Matrix

Represent G by a n x n 0-1 matrix A where

$A_{ij} = 1 \Leftrightarrow$ G has an i-j edge (i)——(j)

$A_{n \times n}$

Variants

- A_{ij} = # of i-j edges (if parallel edges)
- A_{ij} = weight of i-j edge (if any)
- $A_{ij} = \begin{cases} +1 \text{ if } \bigcirc \!\longrightarrow\! \bigcirc \\ -1 \text{ if } \bigcirc \!\longleftarrow\! \bigcirc \end{cases}$

Tim Roughgarden

rep_3

Te
ve

How much space does an adjacency matrix require, as a function of the number n of vertices and the number m of edges?

- ○ $\theta(n)$
- ○ $\theta(m)$
- ○ $\theta(m+n)$
- ○ $\theta(n^2)$

if we have a dense graph, this method is fine, but for sparse graph, this is super wasteful representation.

(m linear → sparse).

Method 2 for representing a graph.

2 - Adjacency Lists

Ingredients

- array (or list) of vertices
- array (or list) of edges
- each edge points to its endpoints
- each vertex points to edges incident on it

Te
ve

How much space does an adjacency list representation require, as a function of the number n of vertices and the number m of edges?

- ○ $\theta(n)$
- ○ $\theta(m)$
- ○ $\theta(m+n)$
- ○ $\theta(n^2)$

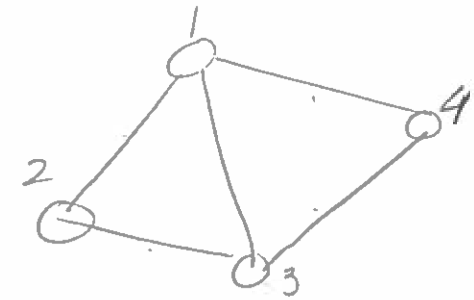
There are two vectors for adjacency list: one keeps vertices and the other one keeps edges:

1	4	6	9	10

size $= n+1$

2	3	4	1	3	1	2	4	1	3
1	2	3	4	5	6	7	8	9	10

$2m$



Adjacency Lists

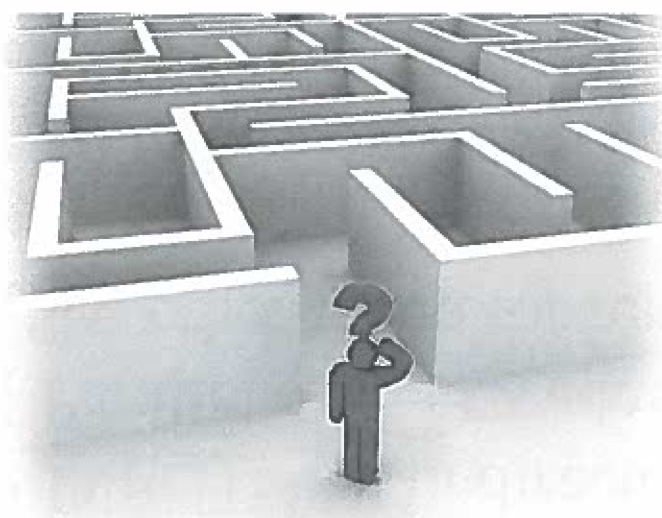
Ingredients	Space
• array (or list) of vertices	$\theta(n)$
• array (or list) of edges	$\theta(m)$
• each edge points to its endpoints	$\theta(m)$
• each vertex points to edges incident on it	$\theta(m)$
	$\theta(m+n)$
	$[or\ \ \theta(max\{m,n\})]$

one-to-one correspondence !

Question: which is better?
Answer: depends on graph density and operations needed.
This course: focus on adjacency lists.

Vep-5

Aug 4, 2015



Design and Analysis of Algorithms I

Contraction Algorithm (Karger)

The Algorithm

C-Alg – 1

The Minimum Cut Problem

- INPUT: An undirected graph $G = (V, E)$.

[Parallel  edges allowed]

[See other video for representation of the input]

- GOAL: Compute a cut with fewest number of crossing edges. (a min cut)

Random Contraction Algorithm

In this algorithm, there is a while loop, one main loop each iteration of loop decrease the no. of vertices by one, until there are only two vertices.

[due to Karger, early 90s]

In this main loop:

While there are more than 2 vertices:

- pick a remaining edge (u,v) uniformly at random
- merge (or "contract") u and v into a single vertex : this will create ① parallel edges, even if we didn't have them before. ② self loop, edges with both nodes are the same
- remove self-loops

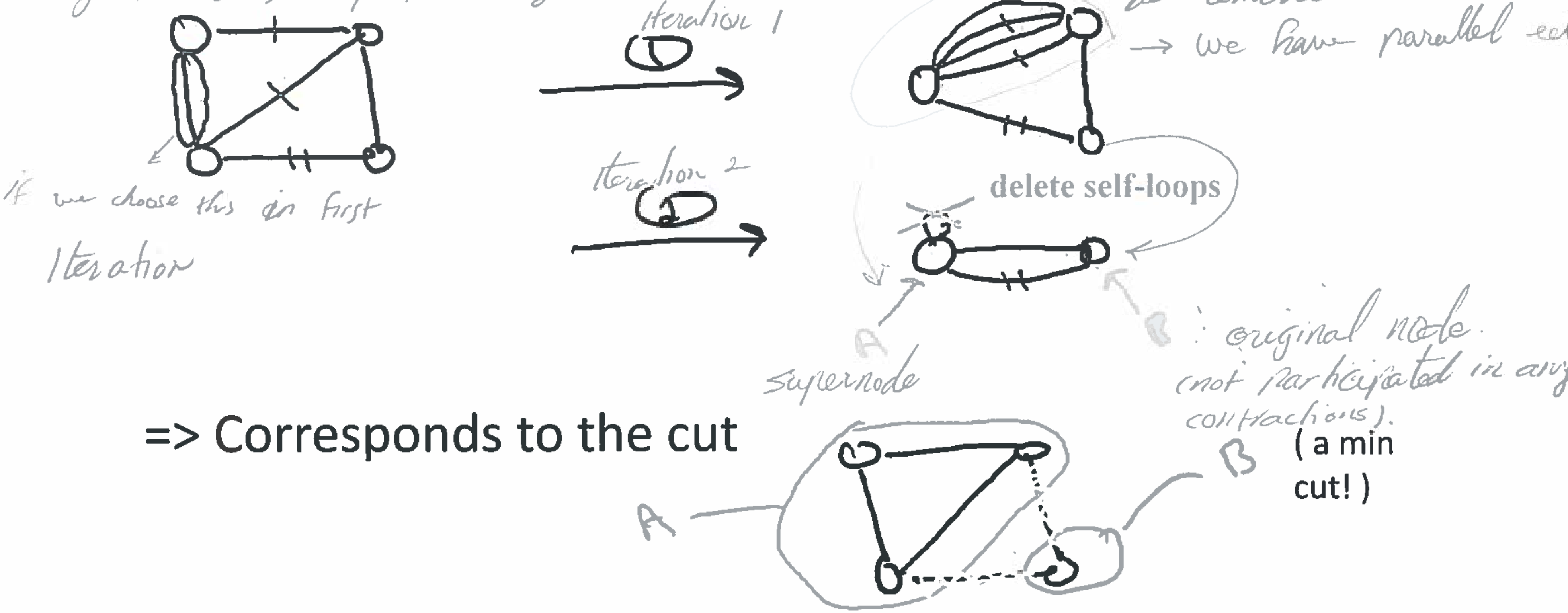
return cut represented by final 2 vertices. (after n-2 iterations)

C-Algo-2

any edge that we choose, will disappear.

Example

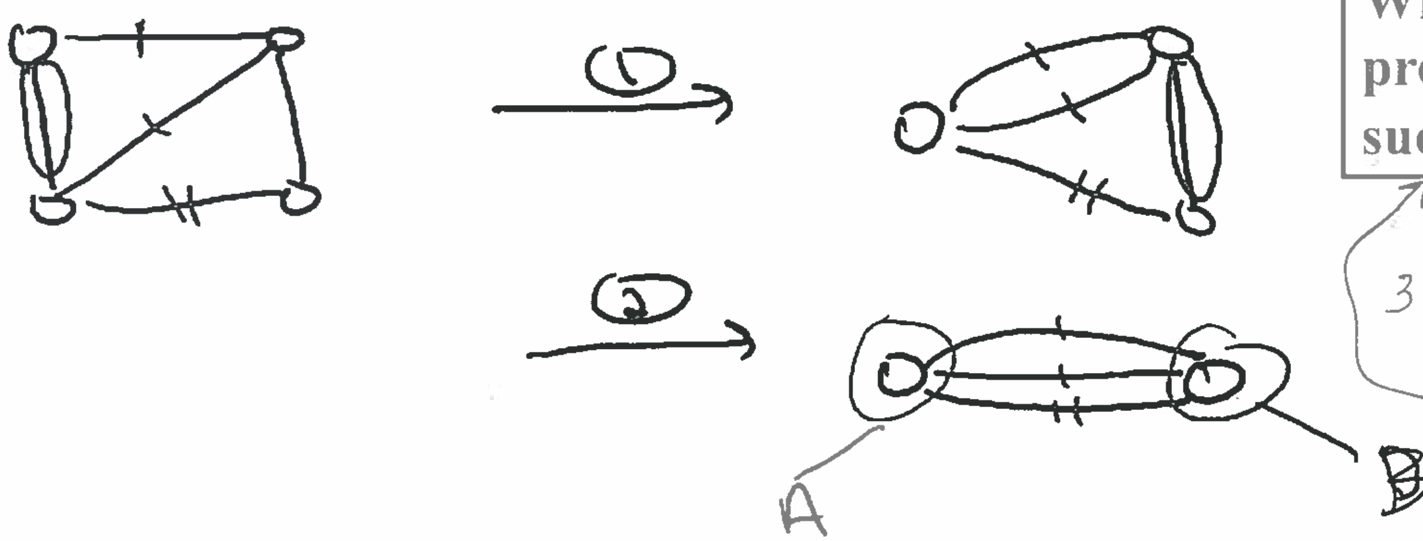
{4 nodes} {5 edges} 20% probability to choose any of the edges.



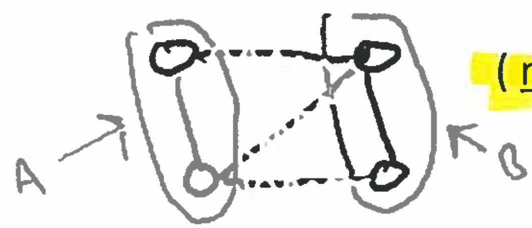
=> Corresponds to the cut

(a min cut!)

Example (con'd)



- Corresponds to the cut



(not a min cut!)

KEY QUESTION: What is the probability of success?

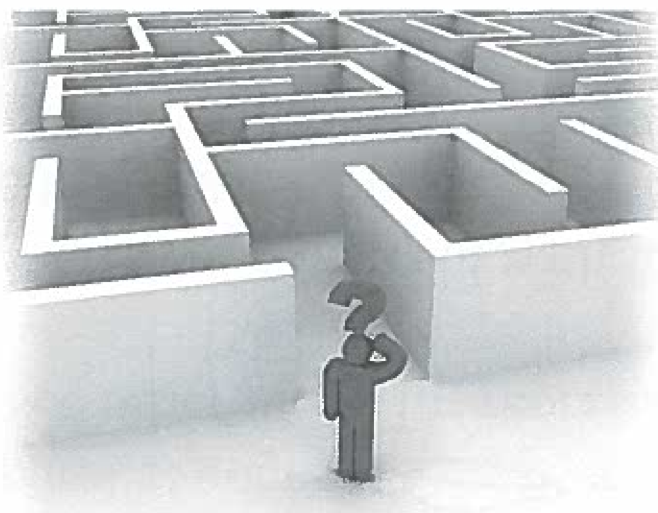
3 crossing edges

Tim Roughgarden

contraction algorithm, sometimes identifies the min cut, and sometimes it does not. So, is it a useful algo? Or

C-Algo-3

40



Design and Analysis of Algorithms I

Contraction Algorithm

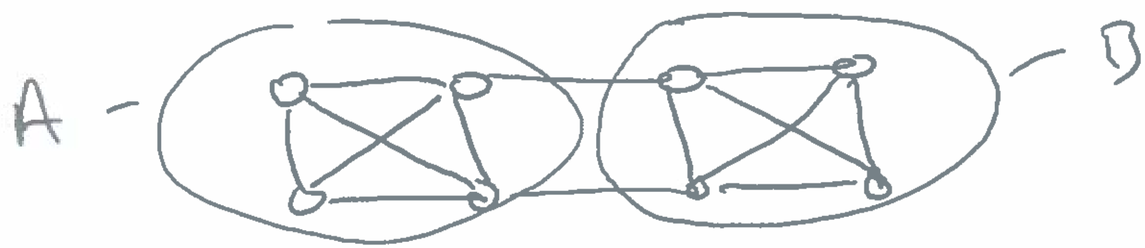
The Analysis

C-Ana 1

The Minimum Cut Problem

Input: An undirected graph G = (V, E).
[parallel edges  allowed]
[See other video for representation of input]

Goal: Compute a cut with fewest number of crossing edges.
(a min cut)



Random Contraction Algorithm

[due to Karger, early 90s]

While there are more than 2 vertices:

- pick a remaining edge (u,v) uniformly at random
- merge (or "contract") u and v into a single vertex
- remove self-loops

return cut represented by final 2 vertices.

c_Ana_2

A graph may have multiple minimum cuts. We only select one of them. So, if the algorithm ends up the other minimum cut, we do not count it as success

The Setup

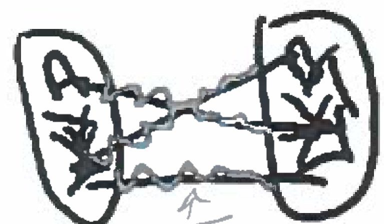
Question: what is the probability of success?

Fix a graph G = (V, E) with n vertices, m edges.

Fix a minimum cut (A, B).

Let k = # of edges crossing (A, B). (Call these edges F)

size of minimum cut.

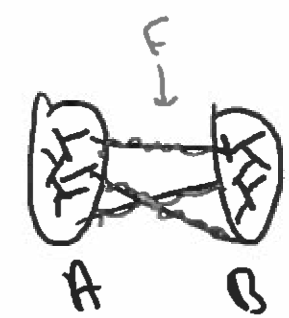


F crossing edges. in the min cut.

Tim Roughgarden

What Could Go Wrong?

1. Suppose an edge of F is contracted at some point
   ⇨ algorithm <u>will not</u> output (A,B). ✓



2. Suppose only edges inside A or inside B get contracted ⇨ algorithm <u>will</u> output (A, B).

i.e. none of F is contracted.

the algorithm succeed only IF

<u>Thus</u>: Pr [output is (A, B)] = Pr [never contracts an edge of F]

each iteration separately. Success at iteration i

Let S_i = event that an edge of F contracted in iteration i.

<u>Goal</u>: Compute $\Pr[\neg S_1 \wedge \neg S_2 \wedge \neg S_3 \wedge \ldots\ldots \wedge \neg S_{n-2}]$

iterations

not symbol = S_1 does not happen.

Tim Roughgarden

c-Ana-3

Te
ve

What is the probability that an edge crossing the minimum cut (A, B) is chosen in the first iteration (as a function of the number of vertices n, the number of edges m, and the number k of crossing edges)?

- k/n
- k/m
- k/n^2
- n/m

$$\Pr[S_1] = \frac{\#\text{ of crossing edges}}{\#\text{ of edges}} = \frac{k}{m}$$

bound based on # edges.

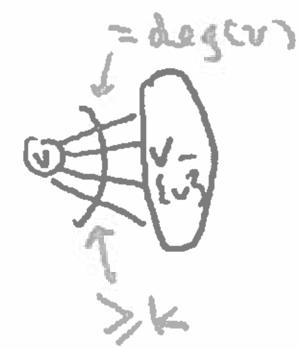
This is an exact calc for the first iteration.

Since we don't know how the # of edges changes with each iteration, but we know how # of vertices change with each iteration, one per iteration

The First Iteration

Key Observation: $\underbrace{\text{degree}}_{\text{\# of incident edges}}$ of each vertex is at least k correct.

Reason: each vertex v defines a cut ({v}. V-{v}).



Since $\underbrace{\sum_v degree(v)}_{\geq kn} = 2m$, we have $\boxed{m \geq \frac{kn}{2}}$ $\Leftarrow 2m \geq kn$

If we have a bunch of nodes, any time we add one edge, the number of edges goes up by one, and sum of the degrees goes up by 2;

Since $\Pr[S_1] = \frac{k}{m}$, $\Pr[S_1] \leq \frac{2}{n}$

$Pr[S_1] = \frac{k}{m} \leq \frac{k}{\frac{kn}{2}} = \frac{2}{n}$

Tim Roughgarden

cAna 4

next baby step

means we don't screw up in the first two iterations, not showing up in the first iteration nor in the second iteration.

The Second Iteration

conditional probability

check this equality

Recall: $\Pr[\neg S_1 \wedge \neg S_2] = \underbrace{\Pr[\neg S_2 | \neg S_1]}_{= 1 - \frac{k}{\text{\# of remaining edges}}} . \Pr[\neg S_1] \geq (1 - \frac{2}{n})$

1 – [chance that we screw up]

what is this?

we don't have clear understanding on this. instead, we write it in terms of vertices, we know what this is.

Note: all nodes in contracted graph define cuts in G (with at least k crossing edges).

➢ all degrees in contracted graph are at least k

So: # of remaining edges $\geq \frac{1}{2}k(n-1)$

So $\Pr[\neg S_2 | \neg S_1] \geq 1 - \frac{2}{(n-1)}$

All Iterations

In general: extending the pattern:

$$\Pr[\neg S_1 \wedge \neg S_2 \wedge \neg S_3 \wedge .... \wedge \neg S_{n-2}]$$

check this equality

$$= \underline{\Pr[\neg S_1]}\ \underline{\Pr[\neg S_2 | \neg S_1]}\ \Pr[\neg S_3 | \neg S_2 \wedge \neg S_1] ...... \Pr[\neg S_{n-2} | \neg S_1 \wedge ... \wedge \neg S_{n-3}]$$

$$\geq (1 - \frac{2}{n})(1 - \frac{2}{n-1})(1 - \frac{2}{n-2}) ..... (1 - \frac{2}{n-(n-4)})(1 - \frac{2}{n-(n-3)})$$

$$= \frac{\cancel{n-2}}{n} \cdot \frac{\cancel{n-3}}{n-1} \cdot \frac{\cancel{n-4}}{\cancel{n-2}} ...... \frac{2}{\cancel{4}} \cdot \frac{1}{\cancel{3}} = \frac{2}{n(n-1)} \geq \frac{1}{n^2}$$

→ thus why we talk about this algorithm.

Problem: low success probability! (But: non trivial) bcs still shocking high.

recall $\simeq 2^n$ cuts !

Tim Roughgarden

(Ana-5

Repeated Trials

Solution: run the basic algorithm a large number N times, remember the smallest cut found.

Question: how many trials needed?

Let T_i = event that the cut (A, B) is found on the i^{th} try.

➢ by definition, different T_i's are independent

So: $\Pr[\text{all N trails fail}] = \Pr[\neg T_1 \wedge \neg T_2 \wedge \ldots \wedge \neg T_N]$

If only one succeed, we are good.

$$= \prod_{i=1}^{N} \Pr[\neg T_i] \leq \boxed{(1 - \frac{1}{n^2})^N}$$

By independence !

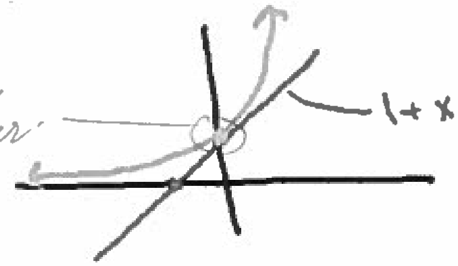
Tim Roughgarden

Repeated Trials (con'd)

Calculus fact: $\forall$real numbers x, $1+x \le e^x$

Pr[all trials fail] $\le (1 - \frac{1}{n^2})^N$

kiss each other



So: if we take $N = n^2$, Pr[all fail] $\le (e^{-\frac{1}{n^2}})^{n^2} = \frac{1}{e}$ 30%

If we take $N = n^2 \ln n$, Pr[all fail] $\le (\frac{1}{e})^{\ln n} = \frac{1}{n}$

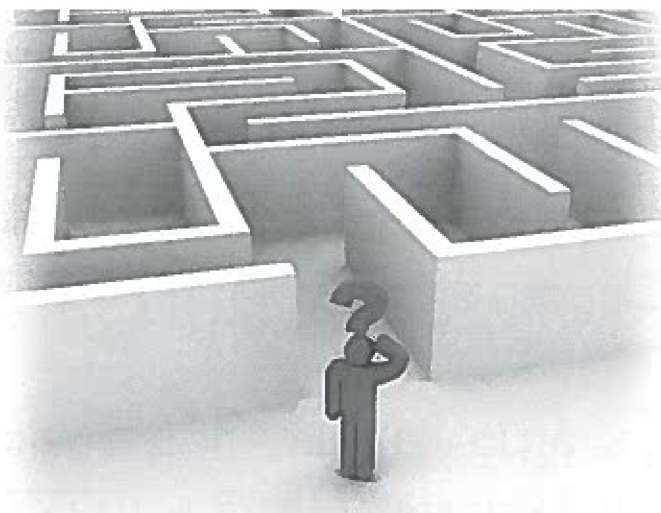
trial.

Running time: polynomial in n and m but slow $(\Omega(n^2 m))$ $n^2 \lg m$ expensive

But: can get big speed ups (to roughly $O(n^2)$) with more ideas.

C-Ana-6

Aug 4, 2018 AD



Design and Analysis
of Algorithms I

Contraction Algorithm

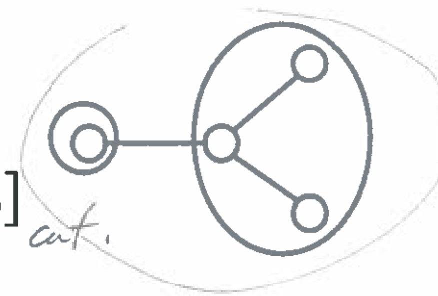
Counting Mininum Cuts

MC — 1

Algo to count min cuts.

The Number of Minimum Cuts

tree.



NOTE: A graph can have multiple min cuts.
[e.g., a tree with n vertices has (n-1) minimum cuts]

3 min cut.

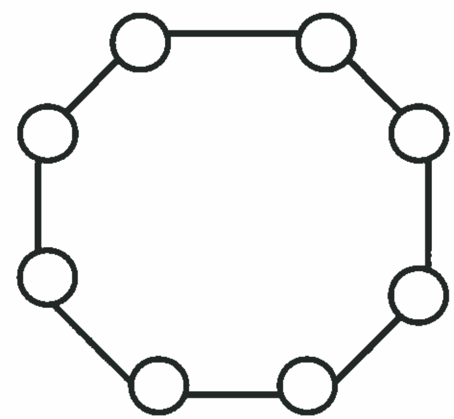
QUESTION: What's the largest number of min cuts that a graph with n vertices can have?

ANSWER: $$\binom{n}{2} = \frac{n(n-1)}{2}$$

Proof of the previous statement.

The Lower Bound

Consider the n-cycle.



(n = 8)

NOTE: Each pair of the n edges defines a distinct minimum cut (with two crossing edges).

- ➢ has $\geq \binom{n}{2}$ min cuts

Tim Roughgarden

MC — 2

The Upper Bound

Let (A_1, B_1) , (A_2, B_2) , …, (A_t, B_t) be the min cuts of a graph with n vertices.

By the Contraction Algorithm analysis (without repeated trials):

→ particular output no any minimum cut.

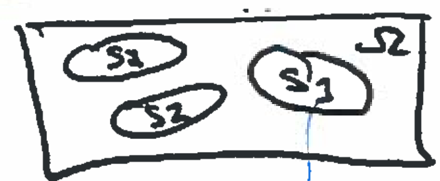
$$\underbrace{\Pr[output = (A_i, B_i)]}_{S_i} \geq \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}} \quad \forall i = 1, 2, \ldots, t$$

distinct cuts.

$Pr(\Omega) = 1$

Note: S_i's are disjoint events (i.e., only one can happen)

➢ their probabilities sum to at most 1



Thus: $\frac{t}{\binom{n}{2}} \leq 1 \Rightarrow t \leq \binom{n}{2}$

t different min cut.

$Pr(S_1) = Pr(S_2) = Pr(S_3) = \frac{1}{\binom{n}{2}}$

QED !

Tim Roughgarden

Max no. of min cut in every graph

Programming Assignment #4

Quiz, 1 question

adjacency list

1 point

1. Download the following text file:

kargerMinCut.txt

The file contains the adjacency list representation of a simple undirected graph. There are 200 vertices labeled 1 to 200. The first column in the file represents the vertex label, and the particular row (other entries except the first column) tells all the vertices that the vertex is adjacent to. So for example, the 6^{th} row looks like : "6 155 56 52 120 ......". This just means that the vertex with label 6 is adjacent to (i.e., shares an edge with) the vertices with labels 155,56,52,120,......,etc

Your task is to code up and run the randomized contraction algorithm for the min cut problem and use it on the above graph to compute the min cut. (HINT: Note that you'll have to figure out an implementation of edge contractions. Initially, you might want to do this naively, creating a new graph from the old every time there's an edge contraction. But you should also think about more efficient implementations.) (WARNING: As per the video lectures, please make sure to run the algorithm many times with different random seeds, and remember the smallest cut that you ever find.) Write your numeric answer in the space provided. So e.g., if your answer is 5, just type 5 in the space provided.

Enter answer here

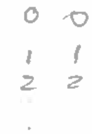
Upgrade to submit

code for randomized contraction algorithm.

find the implementation of edge contraction there two vectors in the code. id and sz

, id: size (num Vertices +1)

initially, each entry has the same vertix no.



After each contraction, the vertix with higher no., gets the vertix no of the lower size.

until there are only 2 vertices left.

sz: size (no. of vertices+1) initially all =1

After each contraction, we add one to the lower vertix no. In this way, we know how many vertices are contracted in each vertix.

Answer : 17

week 4

① How many different min cuts are there in a tree with n nodes

answer → $\binom{n}{2}$.

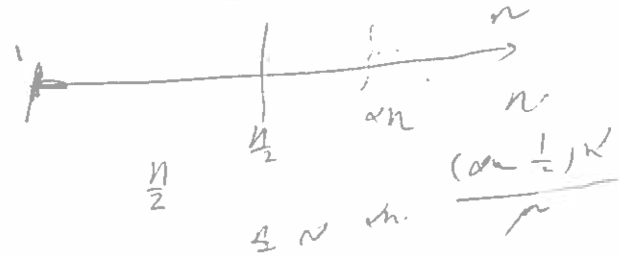
② output ... $P = \frac{1}{\binom{n}{2}}$:

ans:

For every graph G, with n nodes, there exists a min cut (A, B) of G such

$0 \le \alpha < 1$, $P[\text{out} = (A,B)] \ge P$

③ α : randomized median



answer $\alpha = \frac{1}{2}$

through out $(1-\alpha)$.

④ : $0 < \alpha < 1$ — random select.

n → $(1-\alpha)n$.

remaining → αn

$\frac{n}{\frac{1}{\alpha}}$

$(1-$

$-\frac{\log n}{\lg \alpha}$

5

Final Exam

Quiz, 10 questions

1 point

1. Recall the Partition subroutine that we used in both QuickSort and RSelect. Suppose that the following array has just been partitioned around some pivot element: 3, 1, 2, 4, 5, 8, 7, 6, 9

Which of these elements could have been the pivot element? (Hint: Check all that apply, there could be more than one possibility!)

- [ ] 4
- [ ] 9
- [ ] 2
- [ ] 5
- [ ] 3

1 point

2. Here is an array of ten integers: 5 3 8 9 1 7 0 2 6 4

Suppose we run MergeSort on this array. What is the number in the 7th position of the partially sorted array after the outermost two recursive calls have completed (i.e., just before the very last Merge step)? (When we say "7th" position, we're counting positions starting at 1; for example, the input array has a "0" in its 7th position.)

- ( ) 2
- ( ) 3
- ( ) 1
- ( ) 4

1 point

3. What is the asymptotic worst-case running time of MergeSort, as a function of the input array length n?

- ( ) $\theta(n^2)$
- ( ) $\theta(n)$
- ( ) $\theta(n \log n)$
- ( ) $\theta(\log n)$

1 point

4. What is the asymptotic running time of Randomized QuickSort on arrays of length n, in expectation (over the choice of random pivots) and in the worst case, respectively?

- ( ) $\Theta(n^2)$ [expected] and $\Theta(n^2)$ [worst case]
- ( ) $\Theta(n \log n)$ [expected] and $\Theta(n \log n)$ [worst case]
- ( ) $\Theta(n)$ [expected] and $\Theta(n \log n)$ [worst case]
- ( ) $\Theta(n \log n)$ [expected] and $\Theta(n^2)$ [worst case]

1 point

5. Let f and g be two increasing functions, defined on the natural numbers, with $f(1), g(1) \geq 1$. Assume that $f(n) = O(g(n))$. Is $2^{f(n)} = O(2^{g(n)})$? (Multiple answers may be correct, check all that apply.)

- [ ] Maybe, maybe not (depends on the functions f and g).
- [ ] Never
- [ ] Always
- [ ] Yes if $f(n) \leq g(n)$ for all sufficiently large n

1 point

6.

← **Final Exam**
Quiz, 10 questions

Let $0 < \alpha < .5$ be some constant. Consider running the Partition subroutine on an array with no duplicate elements and with the pivot element chosen uniformly at random (as in QuickSort and RSelect). What is the probability that, after partitioning, both subarrays (elements to the left of the pivot, and elements to the right of the pivot) have size at least α times that of the original array?

- ◯ $1 - \alpha$
- ◯ $2 - 2\alpha$
- ◯ $1 - 2\alpha$
- ◯ α

---

1 point

7. Suppose that a randomized algorithm succeeds (e.g., correctly computes the minimum cut of a graph) with probability p (with $0 < p < 1$). Let ϵ be a small positive number (less than 1).

How many independent times do you need to run the algorithm to ensure that, with probability at least $1 - \epsilon$, at least one trial succeeds?

- ◯ $\frac{\log(p)}{\log \epsilon}$
- ◯ $\frac{\log \epsilon}{\log(1-p)}$
- ◯ $\frac{\log(1-p)}{\log \epsilon}$
- ◯ $\frac{\log \epsilon}{\log(p)}$

---

1 point

8. Suppose you are given k sorted arrays, each with n elements, and you want to combine them into a single array of kn elements. Consider the following approach. Divide the k arrays into $k/2$ pairs of arrays, and use the Merge subroutine taught in the MergeSort lectures to combine each pair. Now you are left with $k/2$ sorted arrays, each with $2n$ elements. Repeat this approach until you have a single sorted array with kn elements. What is the running time of this procedure, as a function of k and n?

- ◯ $\theta(nk \log n)$
- ◯ $\theta(nk \log k)$
- ◯ $\theta(n \log k)$
- ◯ $\theta(nk^2)$

---

1 point

9. Running time of Strassen's matrix multiplication algorithm: Suppose that the running time of an algorithm is governed by the recurrence $T(n) = 7 * T(n/2) + n^2$. What's the overall asymptotic running time (i.e., the value of $T(n)$)?

- ◯ $\theta(n^{\log_2(7)})$
- ◯ $\theta(n^2 \log n)$
- ◯ $\theta(n^2)$
- ◯ $\theta(n^{[illegible]})$

---

1 point

10. Recall the Master Method and its three parameters a, b, d. Which of the following is the best interpretation of b^d, in the context of divide-and-conquer algorithms?

- ◯ The rate at which the total work is growing (per level of recursion).
- ◯ The rate at which the work-per-subproblem is shrinking (per level of recursion).
- ◯ The rate at which the number of subproblems is growing (per level of recursion).
- ◯ The rate at which the subproblem size is shrinking (per level of recursion).

Upgrade to submit

Final Exam
Quiz, 10 questions