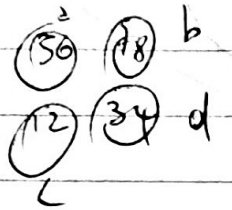
Algos — WEEK 1

Karatsuba multiplication

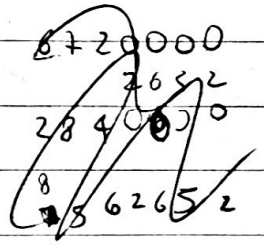


1 $a \cdot c = 672$

2 $b \cdot d = 2652$

3 $(a+b)(c+d) = 6164$

4 $3 - 2 - 1 = 2840$



$\leadsto$ ① · 10000 + ② + ④ · 100

Why? Because: expressing x and y as $(10^{n/2}a + b)$ and $(10^{n/2}c + d)$

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^n ac + 10^{n/2}(ad + bc) + bd$$

Only 3 recursive multiplications!

Divide and conquer design algorithm design paradigms
- Integer Multiplication
- Sorting
- Matrix multiplication
- Closest pair

Primitives for graphs
- Connectivity information
- Shortest paths
- Structure of information
- Social Networks

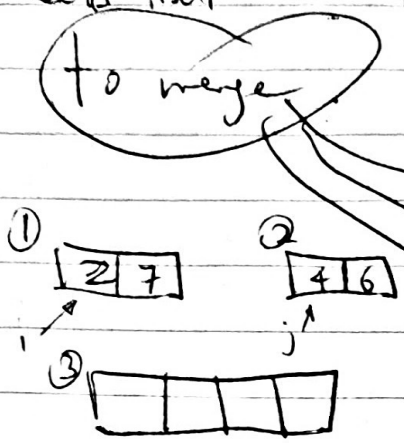
Randomization in algorithm design
- Quick sort
- Primality testing

Use and implementation of Data Structures
- Heaps
- Balanced binary search Trees

Merge Sort

recursive

Calls itself on halves of the array until it works to merge



Either The minimum element of ③ has to be either at the beginning of ① or of ②!

(compare i and j)

if it's in ②, move it to the right, repeat. (if it's in ②, move it)

Running Time:

C = output length n
A = 1st sorted array length n/2
B = 2nd sorted array length n/2
i = 1
j = 1
) init ①

for k = 1 to n ① assignment of k
if A(i) < B(j) : ② comparison (1)
C(k) = A(i) ③ assignment (1)
i++ ④ increment (1)
else [B(j) < A(i)]:
C(k) = B(j)
j++
end

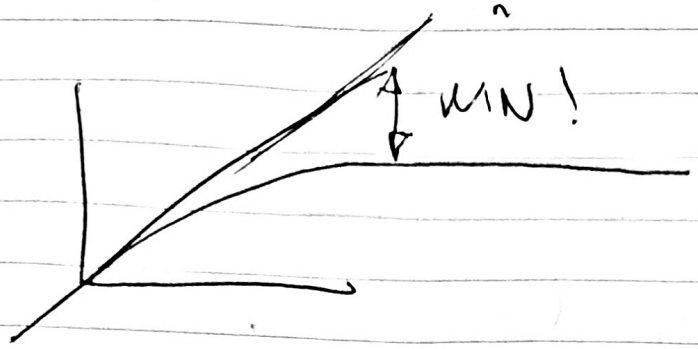
2 ① 2 operations

B(n) for loop is executed n times,
each of these times:
② assignment of k
① comparison
③ assignment
④ increment

(number of calls is given by the log(n) of the array split 4 times, 2 times, 1 time)

+ 4n+2 < 6n ② for Each merge number of merges exists by log, so

Comparing n^2/n to $\frac{6n\log n}{n}$:



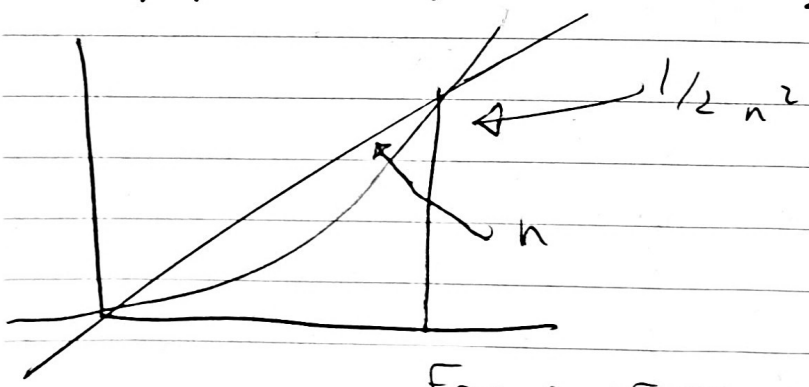
Principles:

Worst case Analysis!

A large case analysis and benchmarks requires previous knowledge of the domain the algorithm will be applied to.

Ignore small things

Asymptotic Analysis: focus on large input sizes n



FOCUS AFTER THIS BIT! SO

$6n\log(n)$ just becomes $n\log(n)$

Asymptotic Analysis

The Gist

Integers $O(n \log(n))$

Suppress constant factors and lower order terms,
- constant factors: too system dependent
- lower order terms: irrelevant for large inputs

Examples

1 - One Loop:

Does Array A contain integer t? $O(n)$

2 - Two loops:

Do Arrays A and B contain integer t? $O(n)$

3 - Two Nested Loops:

Do arrays A and B have a number in comon? $O(n^2)$

4 - Two Nested Loops (II)

Does array A have duplicate entries? $O(n)^2$

Big Oh ~~not~~ is the upper bound.

$T(n) = O(f(n))$ if and only if there exists constants $c, n_0 > 0$ s.t. $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$

c, n_0 cannot depend on n

If $T(n) = a_k n^k + \dots + a_1 n + a_0$ Then

$T(n) = O(n^k)$

Proof: ? I've done this in Analysis 1

$p(x) \leq c \cdot x^{n_{max}}$

$a_n x^n + a_{n-1} x^{n-1} \dots + a_0 x^0 \leq |a_n| x^n + |a_{n-1}| x^n \dots \leq c \cdot x^n$

Big Omega and Theta notation

Greater than or equal to (Best case) — equal to (Average)

$T(n) = \Omega(f(n))$: if and only : if $\exists$ constants c, n_0 such that $T(n) \geq c \cdot f(n)$

$T(n) = \Theta(f(n))$: if and only : if $T(n) = O(f(n))$ and : if

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

Little o notation

$T(n) = o(f(n))$ if and only : if for all constants $c > 0$, $\exists$ a constant n_0 s.t. $T(n) \leq c \cdot f(n) \; \forall \; n \geq n_0$

Big O says there exist a constant c and 1 value n_0 for which this is true, o says for all constants c, there exists an n_0 for which it is true

example $2x \to O(4x)$, since at $n_0 = 8$ for constant $c = 1$ it's

$2x \to O(x^2)$, since no matter the choice of c, there is always an n_0 for which it is true.

Additional Examples

1) $2^{n+10} = O(2^n)$

Need to find c, n_0, s.t.

$2^{n+10} \leq c \cdot 2^n \quad \forall n \geq n_0$

Note: $2^{n+10} = 1024\, 2^n$.

$1024\, 2^n \leq 1025\, 2^n$ for $n \geq n_0 = 0$

$\| \; c$

2) 2^{10n} is not $O(2^n)$

Proof by contradiction. IF $2^{10n} = O(2^n)$ then $\exists c, n_0$ s.t. $2^{10n} \leq c \cdot 2^n \; \forall n \geq n_0$

But then $2^{9n} \leq c \quad \forall n \geq n_0$, which is false!

Problem Problem Terribly

$f(n) \quad g(n)$

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

$x^2 + x = O(x^2)$

$2^n \log_2(2^n) = O(2^n \cdot \log_2(2^n)) \rightarrow cn2^n = O(n2^n)$

$x^2 \log_2(x^{2c}) = O(x^2 \log_2(x^2))$

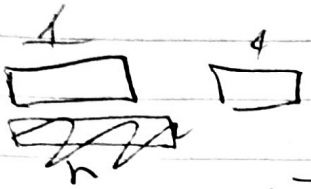
$f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$

$f_n \cdot \log_2(f(n)^x) \leq \quad \forall n \geq n_0$ Time

$f(n) \leq c\,g(n)$ for $\forall n \geq n_0$

$2x^2$ x^2 $\qquad 2\log_3(3n) = O(\log_3 n)$

$2^{2x^2} = 4^{x^2} \leq 8 < 2^{x^2}$



→ Then .... $2n+n$ steps

$2n$ for 2

$2n+n \quad < \quad 3n+2n = 5n$

$3n+n$ steps $= 4n = 3n$

$4n+n$ steps $\quad 5n = 10$

$6n = 20n$

$2n + kn \qquad 23n$

$(2+k)n \quad 2+k \quad (2+k_{n_2})\,n$

k_n

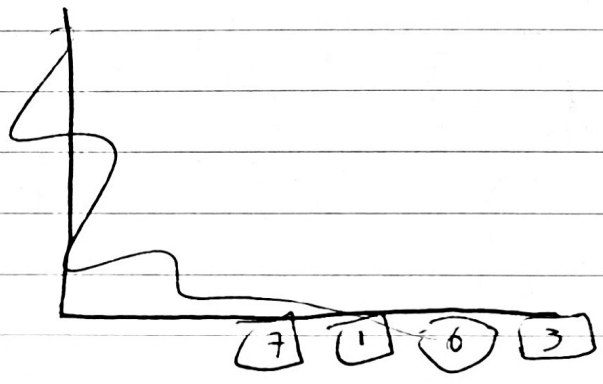
k

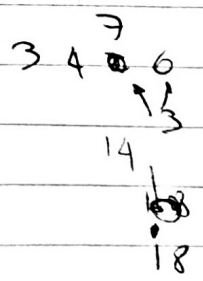
$k_n \quad (1+2+3+4+5+6+7)\,n$

$(2+3+4+5+6+7)\,n$

$\frac{k}{2} \quad k/2$

$\left(k \cdot \frac{k-1}{2}\right) +$





8

5823 7

6245 3

1000000

5 // 2 = 2

5 − 2 = 3

58

62

453

62 453

58 237

$(10^3(ac) + bd)$

$(10^3(ac) + bd)$

$= 10^6 ac + bd + 10^3(ad + bc)$

$ad + bc = (a+b)(c+d)$ $ac + ad + bc + bd$

WEEK 2 L1

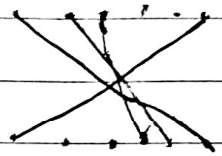
1. Divide into smaller sub-problems
2. Conquer via recursive calls
3. Combine solutions of subproblems into one for the original problem

EXAMPLE 1

Input: Array A containing numbers 1, 2, 3, ..., in some arbitrary order

Output: number of inversions = number of pairs (a, b) such that a precedes b and $a > b$

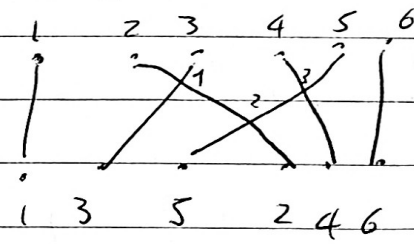
((..i..j..), A[i] > A[j])



EXAMPLE 2

Example 1 ~~1 3 5 2 4 6~~ (1, 3, 5, 2, 4, 6)

Inversions
(3, 2)
(5, 4)
(5, 2)



Motivation: numerical similarity measure between two Lists eg:

Online shopping, people that bought your stuff also bought:

Largest number of inversions in a size n set is:

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

Approaches:
Brute force double loop, $O(n^2)$
Can we do better? Yes, with divide and conquer.

Call an inversion (i, j):

- ① - left inversion if $i, j \leq n/2$ → can compute recursively
- ② - right if $i, j > n/2$
- ③ - split if $i \leq n/2$, $j > \frac{n}{2}$ How? Clever.

For [1,3,5, 2,4,6] ① and ② both return 0!
So all of our inversions are split and come from ③

if ② is linear, $O(n \log(n))$

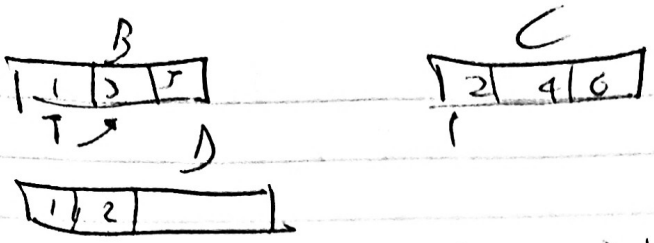
L 2

To do that, we'll Piggyback on Merge Sort.

for every recursive call, we don't only count the inversions but also order the array.
We can do that because it doesn't change the number of split inversions

2 merges in one n, every time ~~that~~ ~~moves to the right~~

every time you compare i and j, if i < j you move i to the right. if i > j you count 3 and move j to the



when copying 2 from the right array, you are exposing 2 different splits! (3,2) and (5,2).

Later when 8 gets copied it only exposes 1.

Q General Asin:

The split mergings involving an element y of the 2nd (or bell) array C are precisely the numbers left in the first array (sorted) A when y is copied to the output d

This is because they are ordered.

Proof: let ~~x be~~ $x \in B$,

① if x is copied to D before y, $x < y$, no mergings

② if y is copied to D before x, then $y < x$, ~~not all remain~~ since $x \le x_{n} \le x_{n-1} \le \dots \le x_{n}$ additional splits are exposed

Run time is $O(n) + O(n) = O(n)$

L3

We'll be talking about $n \times n$ matrices, but applies to non-square ones as well

$$\boxed{x} \cdot \boxed{y} = \boxed{z}$$

$$z_{ij} = \begin{pmatrix} \text{ith row} \\ \text{of } x \end{pmatrix} \cdot \begin{pmatrix} \text{jth column} \\ \text{of } y \end{pmatrix} = \sum_{k=1} x_{ik} \cdot y_{kj}$$

~~Thus~~ We are led to believe that the best algorithm for this will run in $\geq O(n^2)$.

~~BUT~~

~~O~~ $\Theta(n^3)$,

n elements by n elements to compute, each requiring n operations.

DIV n CONQ:

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

$$X \cdot Y = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

(1) 8 products

(2) additions ($O(n^2)$ time)

Fact: Run time is still $\Theta(n^3)$

DIFFERENT APPROACH? ~~IS NEEDED~~

(1) Step 1: 7 products ~~is also good~~, with Strassen

Step 2: find all products still $O(n^2)$ time (more adds and subs but)

Ind better than cubic time!

The seven Products are:

$P_1 = A(F-H)$

$P_2 = (A+B)H$

$P_3 = (C+D)E$

$P_4 = D(G-E)$

$P_5 = (A+D)(E+H)$

$P_6 = (B-D)(G+H)$

$P_7 = (A-C)(E+F)$

$$X \cdot Y = \left(\begin{array}{c|c} AE+BG & AF+BH \\ \hline CE+DG & CF+DH \end{array} \right) = \left(\begin{array}{c|c} P_5+P_4-P_2+P_6 & P_1+P_2 \\ \hline P_3+P_4 & P_1+P_5-P_3-P_7 \end{array} \right)$$

$$AE + AH + DE + DH + DG - DE - AH - BH + BG + DH - BG - DH$$

$$= AE + BG$$

$$AF - AH + AE + AH + DE + DH - CE - DE - AE + AF + CE + CF$$

L6

L6

The master method is a general tool to determine the running time of D&C Algorithms.

A Recursive Algorithm

Express $T(n)$ in terms of running time of recursive calls.

Base case: $T(1) \leq$ a constant

For all $n > 1$: $T(n) \leq 4\,T\left(\frac{n}{2}\right) + O(n)$

(Work done by rec. calls)

A Better Recursive Algorithm

Algorithm #2 (Gauss): recursively compute ac, bd, $(a+b)(c+d)$

Base Case: $T(1) \leq k$

For all $n > 1$ $T(n) \leq 3\,T\left(\frac{n}{2}\right) + O(n)$

L7

General Master Method

Assumption: all of subproblems have same size

Recurrence format:

① Base Case: $T(n) \leq$ a constant for all sufficiently small n

② for all larger n:

$$T(n) \leq a\,T\left(\frac{n}{b}\right) + O\left(n^d\right)$$

a = number of recursive calls

b = factor by which input problem shrinks

d = exponent in the running time outside of recursive calls.

("combine step" (and split) of the original problem if it takes time)

(independent of n)

$T(n)$ is upper bounded by 1 of 3 cases.

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \quad \text{1st case} \\ O(n^d) & \text{if } a < b^d \quad \text{2nd case} \\ O(n^{\log_b(a)}) & \text{if } a > b^d \quad \text{3rd case} \end{cases}$$

L7

1) Example 1: Merge Sort

$a = 2$ (2 calls)

$b = 2$ (size of subinput = size input / 2)

$d = 1$ (Linear time)

$a = b^d \longrightarrow O(n^d \log(n)) = O(n \log(n))$

2) For binary search of sorted array:

$(a, b, d) = 1, 2, 0$ ← one comparison, not dependent on n

$a = 1$ call, half the size $\quad b^d = 2^0 = 1 = a$

$O(n^d \log(n)) \xrightarrow{d=0} O(\log(n))$

3) Multiplication of integers:

Without Gauss's trick:

$a = 4$

$b = 2$

$c = 1$

$b^c < a$, $T(n) = O(n^{\log_b(a)}) = O(n^{\log_2 4}) = O(n^2)$

With:

$a = 3$

$b = 2$

$c = 1$

$O(n^{\log_b(a)}) = O(n^{\log_2 3}) = O(n^{1.5})$

5) Strassen's Matrix Multiplication

$a = 7$
$b = 2$
$d = 2$

$b^d < a$ since $4 < 7$

Case 3

$T(n) = O(n^{\log_2 7}) = O(n^{2.81})$ instead of $O(n^3)$

6) Fictitious Recurrence

$T(n) \geq 2T\left(\frac{n}{2}\right) + O(n^2)$

$a = 2$
$b = 2$
$d = 2$

$b^d = 4 > 2 \Rightarrow$

Case 2

$T(n) = O(n^d) = O(n^2)$

Proof of the master method

Proof of the master method

$$T(n) \leq a\,T\left(\frac{n}{b}\right) + O(n^d)$$

Preamble:

Assume recurrence is

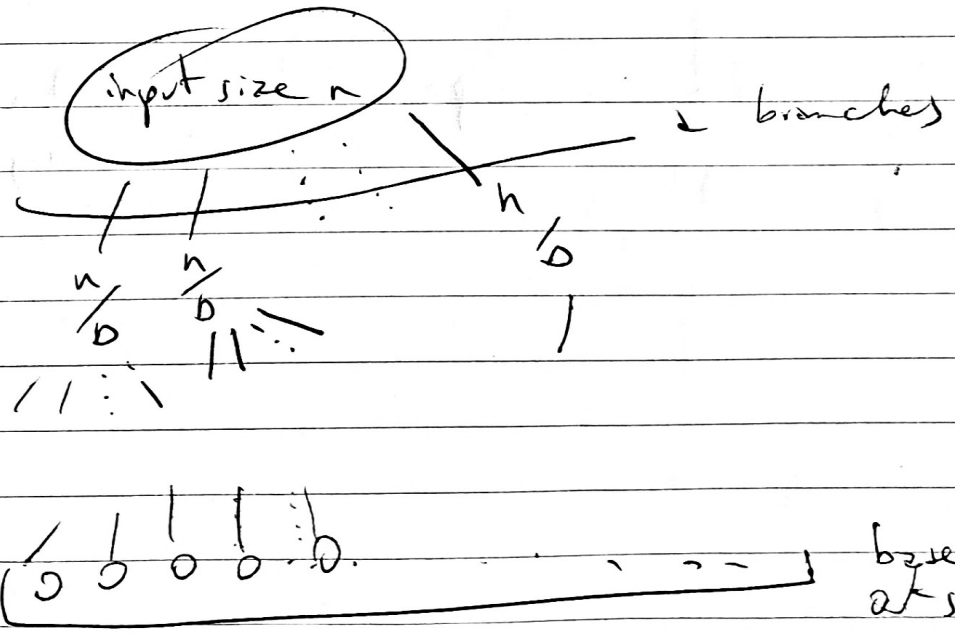
i) $T(1) \leq c$

ii) $T(n) \leq a\,T\left(\frac{n}{b}\right) + cn^d$ (for some constant c which we previously ignored)

and n is a power of b (general case more tedious)

Idea: generalize analysis used with merge sort (a recursion tree)

The Rec Tree:



Total work at level j (ignoring work in recursive calls)

is $\leq a^j \cdot c\left(\left\lceil \frac{n}{b^j} \right\rceil\right)^d$ (work per level j subproblem)

# of level j subproblems (size of each level j subproblem)

$$= cn^d \cdot \left(\frac{a}{b^d}\right)^j$$

Sum over all levels:

$$\text{total work} \le c n^d \cdot \sum_{j=0}^{\log_b n} \left[\frac{a}{b^d}\right]^j$$

TBC

Upper bound on work at level j:

$$c n^d \cdot \left(\frac{a}{b^d}\right)^j$$

Interpretation $\frac{a}{b^d}$

Tug of war between forces of good (b, d) and of evil (

a = rate at which subproblems proliferate as you go deeper in the recursive tree (RSP)

b^d = rate of work shrinkage (n^d time for a problem of size n, size shrinks by b work $\propto n^d$, $n \propto \frac{1}{b}$, work $\propto \left(\frac{1}{b}\right)^d$; work shrinkage is $\propto b^d$

Intuition for 3 Cases

1. RSP = RWS same amount of work at each level $O(n^d \log n)$ (same work at root n^d + $\log n$ levels)
2. RSP < RWS less work each level (most work at root) $O(n^d)$ because n^d dominates over later work done in subproblems
3. RSP > RWS, the amount of work increases at each sublevel, leaves of tree dominate. We might expect $O(\#\text{ leaves})$, $\propto$ [illegible] (levels from root to bottom) (???)

Completion of the proof of the master method. L10

Total work $\leq \boxed{cn^d} \cdot \boxed{\sum_{j=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^j}$ (*)

1) If $a = b^d$, then $= 1$, $= \log_b(n)$, (*) $= cn^d \log_b(n) = O(n^d \log n)$

Remember: $1 + r + r^2 + \dots + r^K = \frac{r^{K+1} - 1}{r - 1}$

For sums, if $r < 1$ is constant, $\sum r^K \leq \left(\frac{1}{1-r}\right)$ = a constant.
if $r > 1$ is constant, $\leq r^K \cdot \left(1 + \frac{1}{r-1}\right)$ — {independent of K, levels}

2) $a < b^d$, so $r < 1$, (*) $\leq cn^d \cdot \sum_{k=1}^{\text{whatever}} (r)^k \leq cn^d \cdot \text{constant}$
$r < 1$
$= O(n^d)$

3) (*) $\leq cn^d \cdot \sum_{j=0}^{\log_b(n)} (r)^j \leq cn^d \cdot \left(\frac{a}{b^d}\right)^{\log_b(n)} \cdot \text{constant} =$
$r > 1$

$c_2 = c \cdot \text{constant}$

$\leq c_2 n^d \cdot (a)^{\log_b(n)} \cdot b^{-d \log_b n} = cn^d \cdot (a)^{\log_b(n)} \cdot n^{-d}$

$= O\left(a^{\log_b(n)}\right) = O\left(n^{\log_b(a)}\right)$

$a^{\log_b(n)} = b^{\log_b(a) \log_b(n)} = n^{\log_b(a)}$

Problem Scribblings for WEEK 2.

$a = 7$
$b = 3$
$d = 2$
$b^d = 9 > 7 = a$, case 2, $O(n^2)$

22

$a = 9$
$b = 3$
$d = 2$

$a = b^d = 9$ so case 1 $O(n^d \log(n)) = (n^2 \log(n))$

$a = 5$
$b = 3$
$d = 1$

~~$a < b^d$, case 2, $O(n^d)$~~

$5 = a > b^d = 3$, case 3, $O(n^{\log_b(a)}) = O(n^{\log_3(5)})$

$a = 1$
$b = 2$
$c = 0$

$n^d \log(n) = \log(n)$

$T(n) < T(\frac{n}{2})$

for $n = \log m$ $m = \log n$ $n = 2^{\log n}$ $T(2^{\log n}) < T(2^{(\log n)/2})$

$T(m)$

$a = 1$
$b = 2$
$d = 0$

WEEK 3

L1 | L1 | L1 | L1

QuickSort

- Definitely a "greatest hit" algorithm
- Prevalent in practice
- Beautiful Analysis
- O(n log n) time "on average", works in place (not much extra memory needed)

Back at the Sorting Problem.

Input: | 3 | 8 | 2 | 5 | 1 | 4 | 7 | 6 |

Output: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Assume all array elements are distinct
(extend to handle duplicate as exercise)

Key idea: partition around a pivot element.

- pick first element of array(3)(for now) | 3 | 8 | 2 | 5 | 1 | 4 | 7 | 6 |
- rearrange array s.t. all elements left of the pivot are smaller than the pivot, all elements to the right are greater than the pivot

| 2 | 1 | 3 | 6 | 7 | 4 | 5 | 8 |

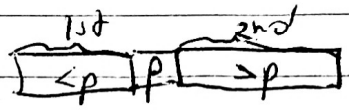
Then Note: pivot is in the right position now
1) This all happens in (O(n)) linear time, and requires no extra memory
2) It reduces the problem size

Quicksort High Level Description

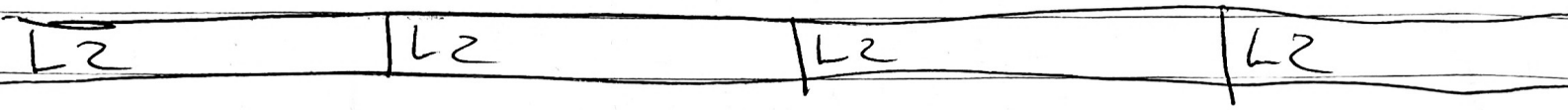
D&C

Quicksort (Array A of length n)

- if n=1 return
- p = ChoosePivot(A, n) (implemented later)
- Partition A around p



- recursively sort 1st & 2nd parts



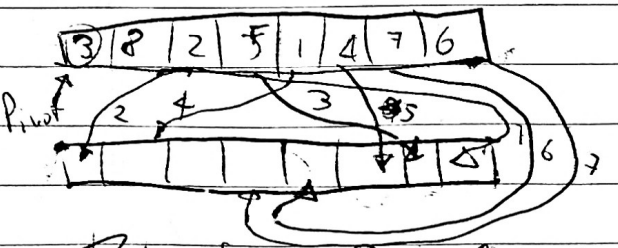
Partitioning around a pivot in detail

Two cool facts about partition:

① Linear time, no extra memory

② Reduces problem size

Easy way out (not in place), using O(n) extra memory



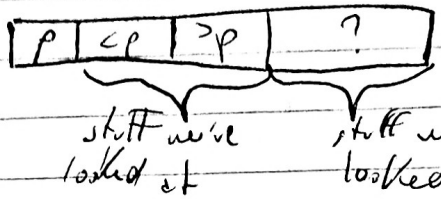
Scan LTR, ~~check~~ if x < Pivot {fill from left}, else fill from right

We can do it without additional memory!

Assume: pivot = 1st element of array

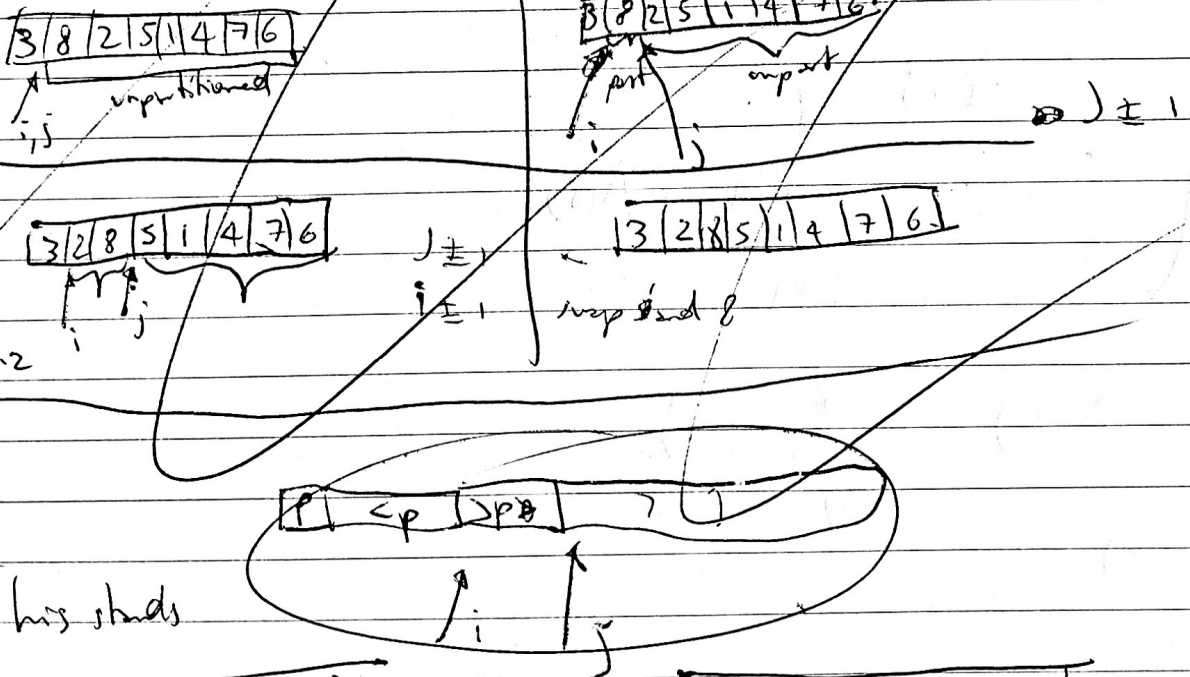
[if not, swap pivot with 1st element of preprocessing step]

High-Level Idea:

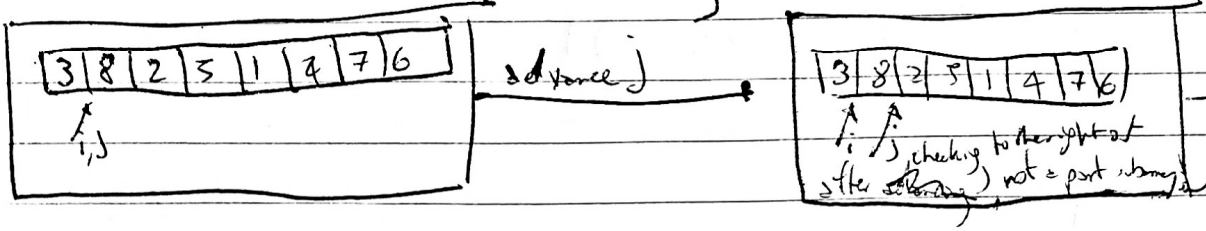


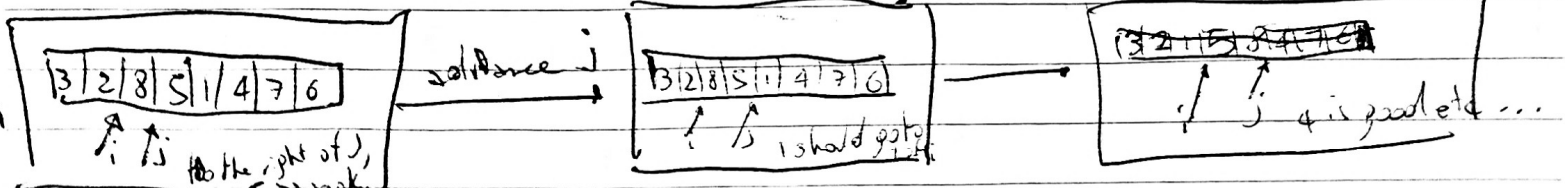
- single scan through array
- invariant: everything looked at so far is partitioned

Partition Example

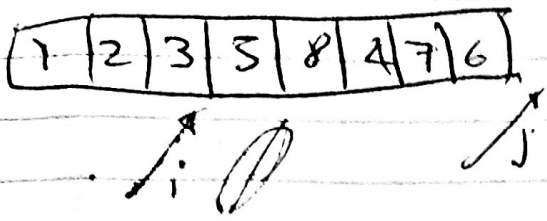


This stinks





Swap with left pivot to the right of i:



Pseudocode for this:

Partition (A, ℓ, r) input = $A[\ell \ldots r]$

- pivot = $A(\ell)$
- $i = \ell + 1$
- for $j = \ell + 1$ to r:
  - if $A[j] < p$:
    - swap $A[j]$, $A[j]$
    - $i = i + 1$
- Swap $A[\ell]$ and $A[i-1]$

$O(n)$ (wow) $n = r - \ell + 1$

$O(1)$ work per array entry (len(array) = n)

Also works in place!

Also correct

Choosing a good Pivot.

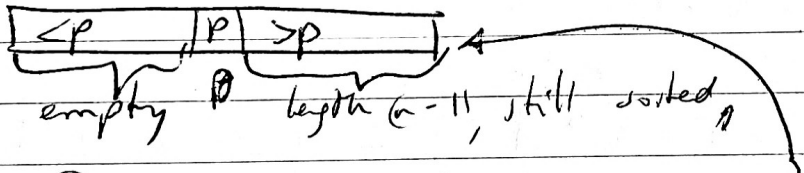
ChoosePivot(A, n) (Array A of length n)

The importance of the Pivot.

Q: running time of Quicksort? Can't discuss now, depends crucially on quality of the pivot chosen

For a Choosepivot = 1st elem, $O(n^2)$ for sorted array

Why?



: QuickSort calls itself again on array of len (n-1), then (n-2) etc $n(n-1) < O(n^2)$

If ChoosePivot found Median element at each subarray, $O(n \log(n))$ (like merge sort), split in two etc)

BIG IDEA: Random Pivots!

Randomised algorithm, will run differently even with the same input because randomness is intrinsic to it

Why could it possibly work?

① Hopefully a random pivot is "pretty good" often enough

② Intuition: approximately balanced split will give good res too. If always get 25-75 ~~%~~, still $O(\lg n)$

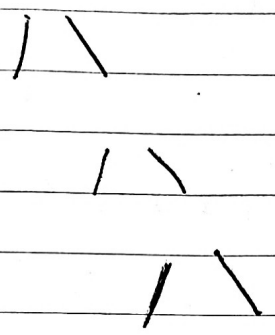
1st + 3rd
$a = 1$
$b = 1$
$d = 1$

→

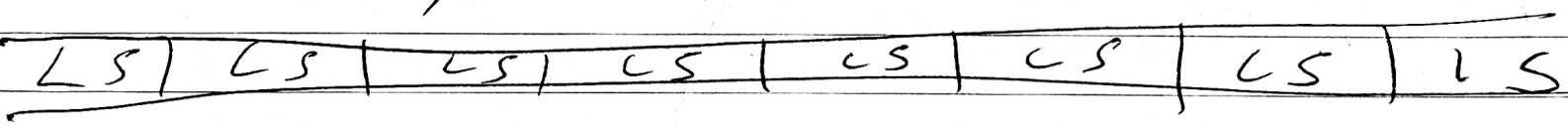
$a = 2$
$b = \left(\frac{1}{0,25} + \frac{1}{0,75}\right)/2 =$
$d = 1$

$0,75^5 = 0,25$, after 5 levels, $5+2$, $b = 1,5$

$b \geq 1,4$ for $O(n \log(n))$



? does it actually work?



Proof that Θ of quicksort is $\Theta(n \log(n))$

Fix input Array of length n

* Sample space Ω = all possible outcomes of random choice in Quicksort

Key notion: for $\sigma \in \Omega$, $C(\sigma)$ = # of comparisons made by quicksort between two input elements

Lemma: running time of Quicksort = # comparisons

Note: Can't apply master method (random, unbalanced subp b/ten

BUILDING BLOCKS

Notation: z_i = ith smallest element of A

z_i is not the element in the ith position of the array (usually)

For $\sigma \in \Omega$, indices $i < j$, let $X_{ij}(\sigma)$ = # of times z_i, z_j get compared with pivot separ

Can be 0 or 1, if ever pivot then 1, one's if not, then 0.

$$C(\sigma) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}(\sigma), \text{ by linearity of expectation}$$

$$E[C] = \sum_{i=1}^{n-1} \sum_{j=i+1} E[X_{ij}] =$$

Complicated! Simple!

$$E[X_{ij}] = 0 \cdot \text{blah blah blah} + 1 \cdot P[X_{ij} = 1] = P[X_{ij} = 1]$$

General Decomposition Principle (might I say)

1 Identify RVAR Y
2 Express Y in terms of indicator RVARS
3 Apply linearity of expectation

3e

LG | LG | LG | LG

Quicksort Theorem: $O(n \log(n))$ for every input.

Key claim: $\forall i < j$, $Pr(z_i, z_j \text{ get compared}) = x_{ij} = \frac{2}{(j-i+1)}$

Proof of Key claim:

fix z_i, z_j, $i < j$

consider $z_i, z_{i+1}, z_{i+2} \ldots\ldots z_{j-1}, z_j$

Inductively: as long as none of these misled:

if pivot smaller than all of these, will end up on the right

" " bigger " " left

⇒ do no comparisons

comparison happens when pivot is picked within these $(j-i+1)$ elements

Consider pivot is z_i or z_j, they will get compared

IF not, no comparison. EVER.

$\frac{2}{j-i+1}$ chances of this happening.

$E[C]$

So $= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(\frac{2}{j-i+1}\right) =$

$= \left(2\sum_{i=1}^{n-1}\right) \cdot \left(\sum_{j=i+1}^{n} \frac{1}{(j-i+1)} = \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots\right) \leq$

$= So. E[C] \leq 2 \cdot n(\text{choices for } i) \cdot \sum_{k=2}^{n} \frac{1}{k} \quad \int \frac{1}{x} = \log(x), \text{ smaller than integral} \leq 2n \log n$

Scribbles from Problems WK3

$\frac{\alpha}{0.5}$ $\frac{0.5}{\alpha}$

$\frac{\alpha}{0.5} = \frac{1}{2}$

$2\alpha = 1 \Rightarrow p$

$1 - 2\alpha$

$n(1-\alpha)$ will take longer for $\alpha < 0.5$

$n(1-\alpha)^d = 1$

$\log(n) + d\log(1-\alpha) = 0$

$d = \frac{-\log n^b}{\log(1-\alpha)} = -\log_{(1-\alpha)}(n)$

$\alpha = 0.5$,

$X_{ij} = \begin{cases} 0 \end{cases}$

$E[X] = \sum_{i=1}^{n-1} \sum_{j=i}^{n}$ Pr (i and) have same birthday)

$= \left(\frac{1}{365} \sum_{i=1}^{n-1}\right) \sum_{j=i+1}^{n}$ $= b$ i+1 + i+2 + ... + i+k

$\frac{2 \pm \sqrt{2900}}{2}$

34

X product of x_1 and x_2
$\geq$ x_1 and x_3

not independent (consider $x_1 = 1$)

$E[YZ] = E[Y] \cdot E[Z]$

1
2 · 2
3 · 2
4 · 2
5 · 2
6 · 2
8

Fixing x_1, yes, but no!

| 3 | 8 | 2 | 5 | 1 | 4 | 7 | 6 |

i j

| 3 | 2 | 8 | 5 | 1 | 4 | 7 | 6 |

i j

WEEK 3 PROBABILITY REVIEW 1

Sample Spaces

Sample space Ω is a collection of all the things that could happen, all possible outcomes

will be dealing with this as finite set in algos

Also each outcome $i \in \Omega$ has a probability $p(i) \geq 0$

Constraint $\sum_{i \in \Omega} p(i) = 1$ Example: 2 dice rolls $\Omega \{(1,1),(1,2) \dots$

EVENTS

An event is a subset $S \subseteq \Omega$

Example: sum of rolls is 11, $S \subseteq \Omega$, $S = \{(5,6)(6,5)\}$

RANDOM VARIABLES

A random variable X is a real-valued function $X: \Omega \to \mathbb{R}$

Example: sum of the two dice

EXPECTATION

Average value, weighted by probability

$E[X]$ of $X = \sum_{i \in \Omega} X(i) \cdot p(i)$

5) LINEARITY OF EXPECTATIONS

Let $X_1, \ldots, X_n$ be RANDVARS, defined on Ω.

$$E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

Sum of two slice ✓

Product X

Also
Simple

LOADING PROBLEM

Sample space Ω = all n^n possible assignments of processes to servers, each equally likely

Let Y = # of processes assigned to the first server.

Goal: compute $E(Y)$

Let $X_j = \begin{cases} 1 & \text{if } j\text{th process assigned to first server.} \\ 0 & \text{otherwise} \end{cases}$

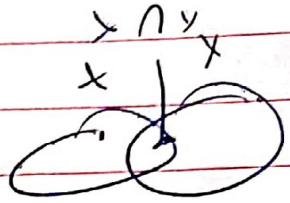
$E[Y] = \sum_{j=1}^{n} X_j$, since linearity, we just have to find X_j, $\frac{1}{n}$

$$\therefore \frac{1}{n}\left(\frac{(n+1)n}{2}\right) = \frac{(n+1)}{2} \sum_{j=1}^{n} \frac{1}{n} = \boxed{1}$$

WEEK 3 PROBABILITY REVIEW 2

6) CONDITIONAL PROBABILITY

Let $X, Y \subseteq \Omega$ be events

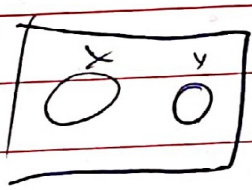


$$Pr[X \mid Y] = \frac{P[X \cap Y]}{P[Y]}$$

Given Y happened, what's the probability of X?

7) INDEPENDENCE OF EVENTS

X, Y are independent if and only if $Pr[X \cap Y] = Pr[X] \cdot Pr[Y]$



8) INDEPENDENCE OF RANDVARS

Definition: for A, B RANDVARS, independent if $Pr[A = a] \cdot Pr[B = b]$

are independent for all a, b

Do not do between sum n product of dice, product has prob of being 1 of 1 if sum = 12

$$E[A \cdot B] = E[A] \cdot E[B]$$

WEEK 8 L1

The Problem:

Input: Array A with n numbers, number i, $1 \leq i \leq n$

Output: ith order statistic (ith smallest element of A)

($i = \frac{n+1}{2}$ for n odd)

($i = n/2$ for n even)

Reduction to Sorting Approach

① Apply a Merge Sort, after sorted return ith element of array

Can we do better?

Fact: can't sort any faster than n log n (optional video)

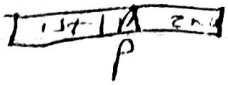
Next: $O(n)$ time (randomized ok) by modifying Quicksort

You can do it without randomization (deterministic) it's just a little more complicated, uses median of medians

D&C, find pivot, partition, look to right or left for jth statistic depending on what statistic you want, & end when the pivot is

Example: $i=10$, pivot ended up in 3 and we're looking for 8th statistic. We look right of the pivot for 2nd statistic.

RSelect (A (len(A) = n) , 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 (1st, i)
5. if j < i return RSelect (2nd, i-j)

Claim: it's correct
Proof: by induction

Running Time? Depends. ~~Best case O(n log n), worst case O(n^2)~~ Worst case

The best pivot is the median, but this is circular. If we did that,

Recurrence $T(n) \leq T\left(\frac{n}{2}\right) + O(n)$ (case 2, $T(n) = O(n)$)

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

WEEK 4 L2

Analysis of RSelect

Proof that it's O(n):

Lemma: RSelect uses $\leq cn$ operations outside the recursive call

Notation: RSelect is in phase j if current array size between $\left(\frac{3}{4}\right)^{j+1} n$ and $\left(\frac{3}{4}\right)^{j} n$

- X_j: for a phase j, it measures the number of recursive calls during phase j

Note: running time of RSelect $\leq \sum_{\text{phase } j} X_j \cdot c \cdot \left(\frac{3}{4}\right)^j n$ (*)

(X_j: RANDOM!; $\sum$: this over all phases; c: amount of work on each subproblem; $\left(\frac{3}{4}\right)^j n$: $\leq$ array size at phase j)

X_j = # of recalls in phase j

Probability of 25-75 split or better is 50%, "average" case!

$$E[N_j] = 1 + \frac{1}{2} \cdot E[N_j] \text{ etc } \ldots$$

Average number of coin flips to get "heads" is 2

(*) becomes $\leq 2cn \sum_{\text{phase } j} (3/4)^j = 2cn \cdot \frac{1}{1-\frac{3}{4}} = 8cn$, $O(n)$! :)

WEEK 4 LESSON 7, Graphs & contr

Minimum cuts problems (solutions) in Graphs are elegant. Nice.

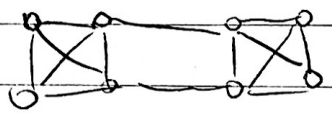
Vertices or Nodes (V)
Edges = (E)
can be directed or undirected

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

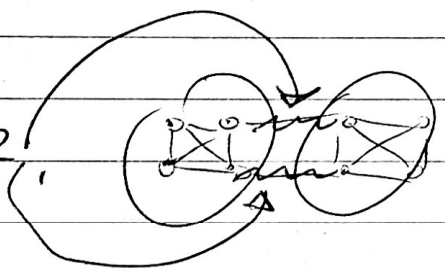
Definition: The crossing edges of a cut (A, B) are those with
- one endpoint in each of (A, B) [undirected]
- tail in A, head in B (directed)

The Minimum Cut problem: For undirected G(V,E
← parallel edges allowed in repr

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



min cut #s CEs are 2.



A few Applications:
- Identify the network bottlenecks / weaknesses
- Community detection in Social Networks
- Image segmentation
  - group of pixels
  - use edge weights (similar color has more weight)

WEEK 4 LESSON 8

What is input size for a graph? We have Vertices and Edges.

Edges are at ~~most~~ least $n-1$, at most $\frac{n(n-1)}{2}$

Sparse vs dense graphs

For n = # of V, m = # of E.

In most (but not all) cases, m is $\Omega(n)$ and $O(n^2)$

In a "sparse graph", m is $O(n)$ or close
In a "dense graph", m is closer to $O(n^2)$

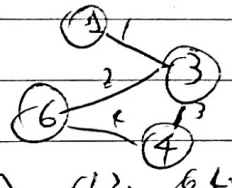
Adjacency Matrixes

$n \times n$ 0-1 matrix, directed or undirected or upper triangular.
OR use -1 to denote direction

If weighted, weight instead of (0 or 1)

Space needed for matrix is $\Theta(n^2)$

Adjacency Lists



- Array of vertices
- Array of edges
- Each edge points to its endpoints (3,6), (3,4), (1,3), (1,5) and can be directed or
- Each vertex points to edges incident on it [3, [1, 2, 3]]

Space needed for Adj List is $\Theta(m+n)$

Which is better then?

Answer: depends on graph density and operations needed

This course will focus on adjacency lists.
They are the best kind for search.

WEEK 4 LECTION 8

MCUT Problem:

Input: undirected G (V, E)

Goal: out of 2^n cuts, which is best?

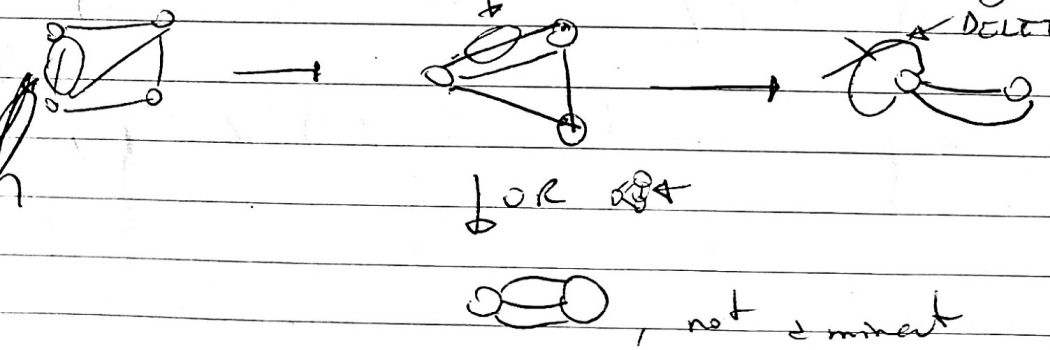
RCA:

while there are more than 2 vertices:

- Pick a remaining edge at random (u, v)
- merge (or "contract") u and v into single vertex
- remove self-loops (edges to itself and edges at same vertex)

return cut represented by final 2 vertices

Example

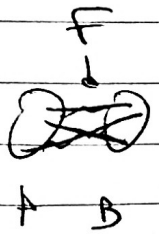


RCA sometimes identifies the min cut, sometimes doesn't.
Depends on choices it makes

The obvious question is then: Is this even a useful algorithm?

What is probability of success?

Fix $G=(V,E)$, n Vs, m Es. min cut (A,B), with k CE (call them F)



IF some edge of F is contracted (by sheer luck), RCA won't output A, B

IF RCA never contracts any edge of F, vertices of A and B stick together and we will get (A,B)

The correct output.

Thus $\Pr[\text{output is } (A,B)] = \Pr[\text{never contracts an edge of } F]$

Let S_i = event that edge of F is contracted in ith (ctr of $n-2$) iteration

In first iteration, we have probability k/m of screwing up.

Key observation: degree of each edge is at least k

$\text{total degree} = 2m \geq k \cdot n \rightarrow m \geq \frac{kn}{2}$

(every edge has 2 vertexes; # of edges)

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

$$Pr\left(\overline{S_1} \cap \overline{S_2}\right) = \underbrace{Pr\left[\overline{S_2} \mid \overline{S_1}\right]}_{1 - \frac{k}{\text{remaining edges}}} \cdot \underbrace{Pr\left[\overline{S_1}\right]}_{\geq \left(1 - \frac{2}{n}\right)} =$$

↑ resulting in terms of vertices, $m > \frac{k(n-1)}{2}$

$$\geq \left(1 - \frac{2}{(n-1)}\right)\left(1 - \frac{2}{n}\right)$$

$$P = \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \cdots \left(1 - \frac{2}{n-(n-3)}\right) \cdot$$

$$\frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{2}{4} \cdot \frac{1}{3} =$$

$$= \frac{2}{n(n-1)} \geq \frac{1}{n^2}$$

↑ very low success rate, but better than random selecting between 2^n cuts

Solution: run RCA a large number of times, until you're sure you have found it.

How many times do we need to do this?

T_i = event that (A, B) is found on the ith try.

By definition different T_is are independent

So $P_r[\text{all } N \text{ trials fail}] = \prod_{i=1}^{N} P_r[\overline{T_i}] \leq \left(1 - \frac{1}{n^2}\right)^N$.

For all numbers $x \in \mathbb{R}$, $1 + x \leq e^x$

$\leq \left(e^{-\frac{1}{n^2}}\right)^N$ so for $N = n^2 \ln(n)$ $= \frac{1}{n}$

Running time is now polynomial! ($\Omega(n^2 m)$)

But can get big speedups to roughly $O(n^2)$ with more ideas

LESSON 10 WEEK 4

Number of minimum cuts for a graph.

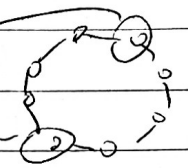
Question: what's the largest number of min cuts a graph with n vertices can have?

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

Why?

Lower Bound for max

Consider n cycle whatever you take the min cut



$\binom{n}{2} = \frac{n(n-1)}{2}$ possible min cuts

$T \geq \binom{n}{2}$

Upper Bound for max

Success probability: $\Pr[\text{output}(A;B)] \geq \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}}$

output $\subseteq$ all

$\frac{T}{\binom{n}{2}} \leq 1$ at most all cuts are min cuts!

$T \leq \binom{n}{2}$

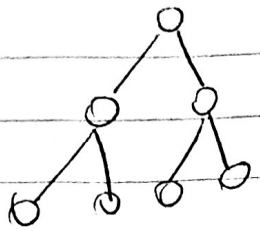
$T = \binom{n}{2} \Leftrightarrow \binom{n}{2} \leq T \leq \binom{n}{2}$

50

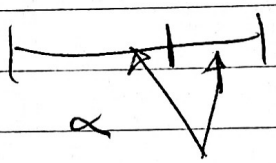
Problem Set Blocks

Controlling $\alpha \to 0$

does not n, does not $\binom{n}{2}$ or $(2^n - 2)$

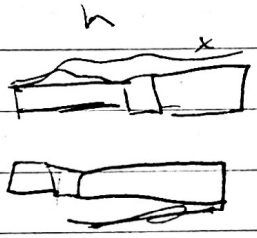


$$\frac{1}{n^2} \leq \frac{2}{n(n-1)}$$

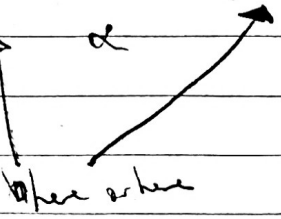


median is there or here

α



α



where or here

$(h-1-x)$ $\quad$ x $\quad$ $(1-x)$

$\alpha = 0.5$ $\quad$ $p = 1$ $\quad$ $p = ?$

$p(h-1-x) \leq \alpha n$

$n - n \times 0 - 1$ [illegible] $x^2 \leq \alpha$

every time

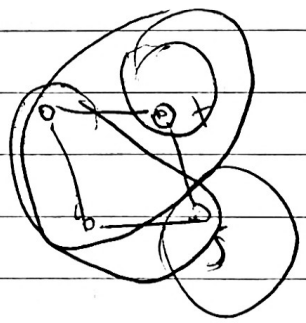
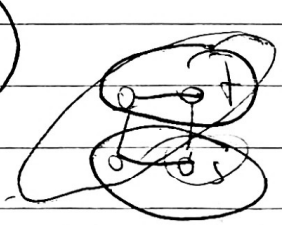
$$n\left(\alpha\right)^{d} = 1$$

size 1 = n

size 2 = $n - (1-\alpha)n = \alpha n$

$$n\left(\alpha\right)^{d} = 1$$

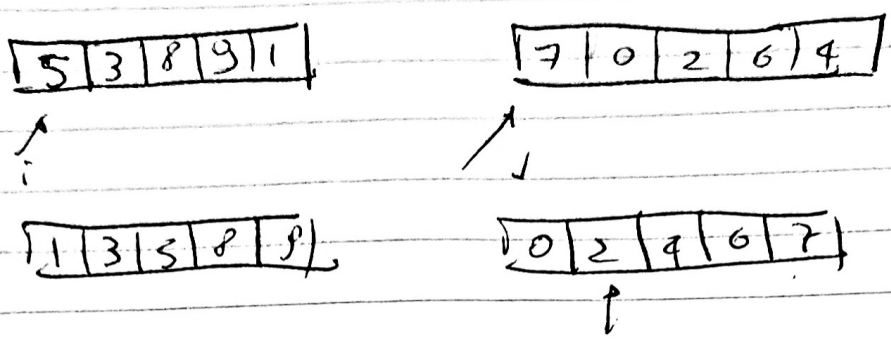
$$\log_{\alpha}(n) = d$$

graph  there are $\binom{n}{2}$ mincuts, of which 

$\frac{\binom{n}{2}}{2^n}$

52

Exam



$f(1)$ $g(1) \geq 1$

increasing

$f_n = O(g(n))$

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

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

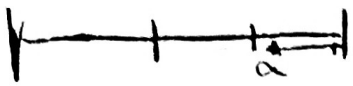
$2n = O(n)$

$2^{f(n)} =$

$2^{2n} = 4^n = O(2^n)$

$4^n \leq$

$0 < \alpha < .5$



$0,5 - \alpha$

p

(1-p) prob of fucking up,

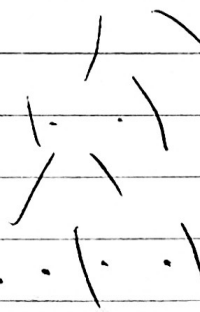
$$(1-p)^d = \epsilon$$

$$d \log(1-p) = \log \epsilon$$

$$d = \frac{\log \epsilon}{\log(1-p)} = \log_{1-p}(\epsilon)$$

$$\frac{\log_\epsilon(b)}{\log_\epsilon(a)} = \log_a(b)$$

$$\log_2(7) = \frac{\log 7}{\log 2}$$



$T_{(A)} \leq$

$a = 2$
$= 2$
$d =$

$a = 7$
$b = 2$
$d = 2$

$b^d = 4 < 2$
$(2^x) \; x \log_2(7)$