Quiz 2

Monday, January 21, 2019 2:33 PM

quiz

Design and Analysis of Algorithms
Massachusetts Institute of Technology
Profs. Erik Demaine, Srini Devadas, and Nancy Lynch

March 12, 2015
6.046J/18.410J
Quiz 1

Quiz 1 62%

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- The quiz contains 7 problems, with multiple parts. You have 120 minutes to earn 120 points.
- This quiz booklet contains 12 pages, including this one.
- This quiz is closed book. You may use one double-sided letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted. Cell phones must be put away.
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to "give an algorithm" in this quiz, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how many minutes to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Please be neat.
- Good luck!

Problem	Title	Points	Parts	Grade	Initials
1	True or False	40	10		60
2	Fast Fourier Transform	5	1		60
3	Yellow Brick Road	10	1		
4	Amortized Analysis	15	1		
5	Verifying Polynomial Multiplication	15	4		
6	Dynamic Programming	15	2		
7	Median of Sorted Arrays	20	3		60
Total		120		74	

Name: _____

Problem 1. True or False. [40 points] (10 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false and briefly explain why.

(a) (T) F [4 points] With all equal-sized intervals, a greedy algorithm based on earliest start time will always select the maximum number of compatible intervals.

EFT will be optimal
This can be proven inductively on size of set of intervals. BC: EFT selects only one is optimal IS: Given optimal selection of set size n+1, consider set of size n+1, we [illegible] select one task, return remaining can set to at most size n

(b) (T) F [4 points] The problem of weighted interval scheduling can be solved in $O(n \log n)$ time using dynamic programming.

1) sort by EST O(n lg n)
2) DP: subproblems: DP[i..] = max(DP[i+1], w(i)+DP[next task])
recursion lgn cost to find next task

(c) (T) F [4 points] If we divide an array into groups of 3, find the median of each group, recursively find the median of those medians, partition, and recurse, then we can obtain a linear-time median-finding algorithm.

mass is conserved. believe it or not

$T(n) = T(a/b) + O(1)$, $b > 1$

By IH, Alg proceeds optim
If we ex
no eles, r
is opt. S
we exclud
eles... a
such ele
are NE
clear

(d) (T) F [4 points] If we used the obvious $\Theta(n^2)$ merge algorithm in the divide-and-conquer convex-hull algorithm, the overall time complexity would be $\Theta(n^2 \log n)$.

Master's Thm
$T(n) = 2T(n/2) + O(n^2)$
n lg n
O(1)
2T(n/2)
O(n^2)
0) Sort on x coord
1) Divide problem in half, based off x
2) Conquer
3) Combine

(e) (T) F [4 points] Van Emde Boas sort (where we insert all numbers, find the min, and then repeatedly call SUCCESSOR) can be used to sort $n = \lg u$ numbers in $O(\lg u \cdot \lg\lg\lg u)$ time.

Trick question, has an extra lg in there you bastard

2

n · lg lg n

? Low log to scale a VEB tree

MT ca
c-crit
f(n) =

s echanges from f(n) = Theta($n^{c\text{-}crit}$) to it dominating.
here is 1
Theta(n^2), therefore, we simply have T(n) = O(f(n)) = O(n^2)

(f) T **F** [4 points] Van Emde Boas on n integers between 0 and $u-1$ supports successor queries in $O(\lg \lg u)$ worst-case time using $O(n)$ space.

need $O(n)$ space to store bit vector

(g) T **F** [4 points] In the potential method for amortized analysis, the potential energy should never go negative.

otherwise ~~[illegible]~~ or assessment fee ~~[illegible]~~ is no longer granted

$\sum \hat{c}_i \geq \sum c_i$

(h) T **F** [4 points] The quicksort algorithm that uses linear-time median finding to run in worst-case $O(n \log n)$ time requires $\Theta(n)$ auxiliary space.

quicksort is $O(n^2)$

(i) T **F** [4 points] Searching in a skip list takes $\Theta(\log n)$ time with high probability, but could take $\Omega(2^n)$ time with nonzero probability.

hard-cap height, not of "right moves"

(j) T **F** [3 points] The following collection $\mathcal{H} = \{h_1, h_2, h_3\}$ of hash functions is ~~universal~~, where each hash function maps the universe $U = \{A, B, C, D\}$ of keys into the range $\{0, 1, 2\}$ according to the following table:

x	A	B	C	D
$h_1(x)$	1	0	1	1
$h_2(x)$	0	1	0	1
$h_3(x)$	2	2	1	0

$Pr(h(A) == h'(A)) \leq 1/3$

not universal

not mapping for keys C and D

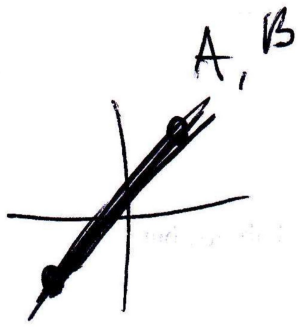
H| = 3, m = 3
r any pair of keys, hash functions that have a collision must be bounded by |H| / m = . bounded by 1
B: collide on h3
C: collide on h1, h2
D: coollide on h1
C: 0
D: 1 collide
D: 1 collide

Problem 2. Fast Fourier Transform (FFT). [5 points] (1 part)

Ben Bitdiddle is trying to ~~multiply~~ compute two polynomials using the FFT. In his trivial example, Ben sets $a = (0, 1)$ and $b = (0, 1)$, both representing $0 + x$, and calculates:

$$A = \mathcal{F}(a) = B = \mathcal{F}(b) = (1, -1),$$
$$C = A * B = (1, 1),$$
$$c = \mathcal{F}^{-1}(C) = (1, 0).$$

So c represents $1 + 0 \cdot x$, which is clearly wrong. Point out Ben's mistake in one sentence; no calculation needed. (Ben swears he has calculated FFT $\mathcal{F}$ and inverse FFT $\mathcal{F}^{-1}$ correctly.)



We need enough samples to recover a 2*N degree polynomial, so for polynomials of d=1, we need to have 4 samples each (regardless of the actual value),

For a basic FFT package, this can be achieved by simpling padding trailing zeros in the coeff vector
It will detect d'=3 for A' and B'
and generate
sample points at
X = {1, -1, i, -1}}

(1, 1)

(-1, 1)

polynomial of size

FFT is expected to take

In 2 samples!

Fourier Transform Expected to generate two samples each

Problem 3. Yellow Brick Road. [10 points] (1 part)

Prof. Gale is developing a new Facebook app called "Yellow Brick Road" for maintaining a user's timeline, here represented as a time-ordered list $e_0, e_1, \ldots, e_{n-1}$ of n (unchanging) events. (In Facebook, events can never be deleted, and for the purposes of this problem, don't worry about insertions either.) The app allows the user to mark an event e_i as ***yellow*** (important) or ***grey*** (unimportant); initially all events are grey. The app also allows the user to jump to the next yellow event that comes after the event e_i currently on the screen (which may be yellow or grey). More formally, you must support the following operations:

1. MARK-YELLOW(i): Mark e_i yellow.
2. MARK-GREY(i): Mark e_i grey.
3. NEXT-YELLOW(i): Find the smallest $j > i$ such that e_j is yellow.

Give the fastest data structure you can for this problem, measured according to *worst-case time*. The faster your data structure, the better.

Hint: Use a data structure you have seen in either 6.006 or 6.046 as a building block.

use a deterministic skip list.

skip list

van emde boas tree's, we did it baby!

veb tree can do all ops in lglgn time using a bit vector to represent yellowness.

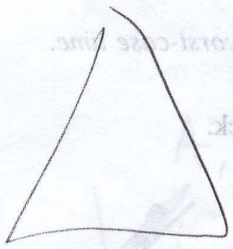
regular bit vector can work but our cost on next-yellow

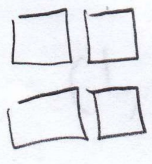
Problem 4. Amortized Analysis. [15 points] (1 part)

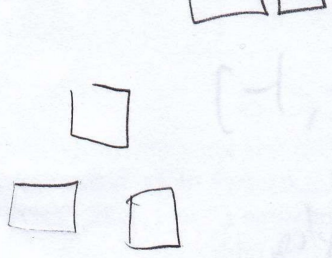
Design a data structure to maintain a set S of n distinct integers that supports the following two operations:

1. INSERT(x, S): insert integer x into S.
2. REMOVE-BOTTOM-HALF(S): remove the smallest $\lceil \frac{n}{2} \rceil$ integers from S.

Describe your algorithm and give the worse-case time complexity of the two operations. Then carry out an amortized analysis to make INSERT(x, S) run in amortized $O(1)$ time, and REMOVE-BOTTOM-HALF(S) run in amortized 0 time.







Insert: Theta(n, S): append n to singly linked List S
Remove-Bottom-Half(S):
m = linear-median-finder(S)
for x in S:
    if x < m: delete x from S

RBH is Theta(n) = c*n
Ins is Theta(1) = d

let c'(Insert) = c+d
let c'(RBH) = 0

Show: summation(operaitons to build up n elements and then operation cost to build up n elements and dlete it)
d*n+c*n <= (c+d)*n + 0

Therefore, our amortized cost analysis remains true.

a more robust but esoteric proof
For all operations, show that
c' = c' + change in potential
for all ops.

this can be used to generate the implicaiton that
summation(c' for all c') >= summation(c for all c)

delete it) <= summation(amortized

Problem 5. Verifying Polynomial Multiplication. [15 points] (4 parts)

This problem will explore how to check the product of two polynomials. Specifically, we are given three polynomials:

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0,$$
$$q(x) = b_n x^n + b_{n-1}x^{n-1} + \cdots + b_0,$$
$$r(x) = c_{2n}x^{2n} + c_{2n-1}x^{2n-1} + \cdots + c_0.$$

We want to check whether $p(x) \cdot q(x) = r(x)$ (for all values x). Via FFT, we could simply compute $p(x) \cdot q(x)$ and check in $O(n \log n)$ time. Instead, we aim to achieve $O(n)$ time via randomization.

(a) [5 points] Describe an $O(n)$-time randomized algorithm for testing whether $p(x) \cdot q(x) = r(x)$ that satisfies the following properties:

1. If the two sides are equal, the algorithm outputs YES.
2. If the two sides are unequal, the algorithm outputs NO with probability at least $\frac{1}{2}$.

Choose × at random among the n roots of unity
We evaluate p(x), q(x), r(x)
Generate $\sqrt{n}$ samples for each fn.
at cost $O(\sqrt{n}\log n)$

Pick a single sample. randomly, from a set of values between 1 and 4*n.
Evaluate for all polynomials, and then check the products are equal.
O(n) to eval poly's individual. Convolution and checks are O(1) after that.

(b) [2 points] Prove that your algorithm satisfies Property 1.

case condition 1: $p(x)q(x) = r(x)$
$p(x_0)q(x_0) = r(x_0) \mid x_0 \in X$
all checks will pass!

-3

(c) [3 points] Prove that your algorithm satisfies Property 2.

Hint: Recall the Fundamental Theorem of Algebra: A degree-d polynomial has (at most) d roots.

~~Any polynomial is diff.~~

b/c clarity

d-degree poly can be defined on $d+1$ points.

Case 2 contrapos $\Rightarrow$ $\exists$ at least one of the ~~points~~ n points is n^{th} roots of unity disagree.

Very birthday paradox, by sampling $\sqrt{n}$ of n^{th} roots of unity, we have at least a 0.5 chance to "discover" the bad pt.

(d) [5 points] Design a randomized algorithm to check whether $p(x) \cdot q(x) = r(x)$ that is correct with probability at least $1 - \varepsilon$. Analyze your algorithm in terms of n and $1/\varepsilon$.

Bf: ovs the no-deletion Lost.

-5

ey Idea: if they don't match and you check a random sample, it's almost impossible that you don’t
etect it. worst case in the universe is d/|range of values you're checking|

ey insight: an equality check can be "re-organized" to a new fn that evalutes to zero for a "pass"
*q == r, can be rephrased to s = p*q - r ==0.

s some degree 2*d polynomial with at most 2*d roots. i.e., 2*d potential values it can be zero.

hat means at BEST, if all roots are within that weird range, Pr(s(a) == 0) <= 0.5

- 2 sides are unequal
- however, the thing is some polynomial, so worst case we might hit a bunch fo zero's by sheer luck.

Ve have at least a 50% detection rate every unique sample.
or correctness up to $1-\epsilon, we\ require\ 1-\left(\frac{1}{2}\right)^{m} checks$
psilon = (1/2)^m = 2(-m)
= - lg(epsilon)

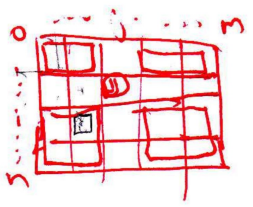
e require lg(epsilon) iterations at cost O(n)
r total cost of O(n * lg(epsilon)

Problem 6. Dynamic Programming. [15 points] (2 parts)

Prof. Child is cooking from her garden, which is arranged in grid with n rows and m columns. Each cell (i, j) $(1 \le i \le n, 1 \le j \le m)$ has an ingredient growing in it, with ***tastiness*** given by a positive value $T_{i,j}$. Prof. Child doesn't like cooking "by the book". To prepare dinner, she will stand at a cell (i, j) and pick one ingredient from each quadrant relative to that cell. The tastiness of her dish is the product of the tastiness of the four ingredients she chooses. Help Prof. Child find an $O(nm)$ dynamic programming algorithm to maximize the tastiness of her dish.

Here the four ***quadrants*** relative to a cell (i, j) are defined as follows:

$$\begin{aligned}
\text{top-left} &= \{\text{all cells } (a,b) \mid a < i, b < j\},\\
\text{bottom-left} &= \{\text{all cells } (a,b) \mid a > i, b < j\},\\
\text{top-right} &= \{\text{all cells } (a,b) \mid a < i, b > j\},\\
\text{bottom-right} &= \{\text{all cells } (a,b) \mid a > i, b > j\}.
\end{aligned}$$



Because Prof. Child needs all four quadrants to be non-empty, she can only stand on cells (i, j) where $1 < i < n$ and $1 < j < m$.

(a) [10 points] Define $TL_{i,j}$ to be maximum tastiness value in the top-left quadrant of cell (i, j): $TL_{i,j} = \max\{T_{a,b} \mid 1 \le a \le i, 1 \le b \le j\}$. Find a dynamic programming algorithm to compute $TL_{i,j}$, for all $1 < i < n$ and $1 < j < m$, in $O(nm)$ time.

we use a bottom up DP

Algorithm with subproblems: $TL_{i,j}$ corresponding to max value of tastiness in TL quadrant.

recursion: $TL_{i,j} = \max(TL_{i-1,j}, TL_{i,j-1}, T_{i-1,j-1})$ | $i>1, j>1$

$i=1$ or $j=1$: 0

Correct: subproblems are already computed when invoked.

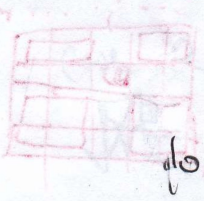
run time: nested for loop for $O(nm)$ time

```
for i = 1 to n:
    j = 1, TL_{i,j} = 0
for j = 1 to n:
    i = 1, TL_{i,j} = 0
```

```
for i = 2 to n:
    for j = 2 to n:
        TL_{i,j} = max(T_{i-1,j-1}, TL_{i-1,j}, TL_{i,j-1})
```

(b) [5 points] Use the idea in part (a) to obtain an $O(nm)$ algorithm to find the tastiest dish.

PC:
- Use DP to compute & store quadrant tastiness values for TL, BL, TR, and BR.



Use this information to get score for each location. Take max.

Time Analysis. DP: 4.O(nm) = O(nm)

calculate score: O(nm) · O(1)
- O(nm): locations
- O(1): calculation cost

retrieve answer: search O(nm) values

O(nm)

Problem 7. Median of two sorted arrays. [20 points] (3 parts)

Finding the median of a sorted array is easy: return the middle element. But what if you are given two sorted arrays A and B, of size m and n respectively, and you want to find the median of all the numbers in A and B? You may assume that A and B are disjoint.

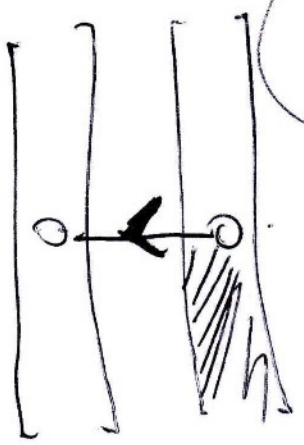
(a) [3 points] Give a naïve algorithm running in $\Theta(m+n)$ time.

target rank = $(m+n)/2$
current_rank = 0, i=0 j=0
while
if A[i] < B[j] i++ last = A[i]
else: j++ last = B[j]
return last

Merge the two arrays by popping off the next smallest element between the A and B into array C, return the $\lfloor \frac{m+n}{2} \rfloor$ element in C

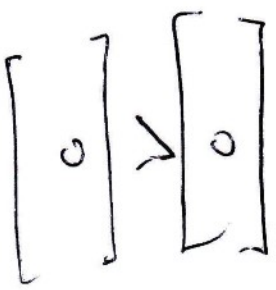
(b) [10 points] If $m = n$, give an algorithm that runs in $\Theta(\lg n)$ time.

i, j = 1
k, l = n
while j-1 > 1,
if A[k] median([A[i:j]]) > media



If I can reduce problem size by some fraction, while only doing O(1) overhead, this is a O(lg n) algo

biggest 4th
1
3
5



(c) [7 points] Give an algorithm that runs in $O(\lg(\min\{m, n\}))$ time, for any m and n.
Don't spend too much time on this question!

Some algorithm,
except when one
array reduces to empty array,
the median of remaining
non empty array

The first element of the big array is smaller than m-1 elements
however, the median element must be smaller than exactly
(m+n)/2 - 1 elements and no more.

in the case n < m, this element is defiinitely fucked.

In fact, the (m-n)/2 is fucked because it's smaller than m/2+n/2 elements. which it shoudlnt be.