Quiz 1

- 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		
2	Fast Fourier Transform	5	1		
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		
Total		120			

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.

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

(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.

(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 $\Theta(n^2 \log n)$.

(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.

(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.

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

(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.

(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.

(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

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

Ben Bitdiddle is trying to multiply 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.)

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.

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.

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 $\frac{1}{2}$.

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

(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.

(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 $1/\varepsilon$.

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) ($1 \leq i \leq n, 1 \leq j \leq m$) 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): $TL_{i,j} = \max\{T_{a,b} \mid 1 \leq a \leq i, 1 \leq b \leq j\}$. Find a dynamic programming algorithm to compute $TL_{i,j}$, for all $1 < i < n$ and $1 < j < m$, in $O(nm)$ time.

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

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.

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

(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!*

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2015