Final In-Class Exam
(7:05 PM – 8:20 PM : 75 Minutes)

• This exam will account for either 15% or 30% of your overall grade depending on your relative performance in the midterm and the final. The higher of the two scores (midterm and final) will be worth 30% of your grade, and the lower one 15%.

• There are three (3) questions, worth 75 points in total. Please answer all of them in the spaces provided.

• There are 18 pages including four (4) blank pages and one (1) page of appendix. Please use the blank pages if you need additional space for your answers.

• The exam is open slides and open notes. But no books and no computers (no laptops, tablets, capsules, cell phones, etc.).

Good Luck!

<table>
<thead>
<tr>
<th>Question</th>
<th>Pages</th>
<th>Score</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A Queue from Two Stacks</td>
<td>2–6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2. Average Deviation from a Running Max</td>
<td>8–11</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>3. Largest Divisor</td>
<td>13–15</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>75</td>
</tr>
</tbody>
</table>

Name: __________________________________________
Figure 1: A FIFO (First In, First Out) queue $Q$ is implemented using two LIFO (Last In, First Out) stacks $Q.S_1$ and $Q.S_2$.

**Question 1. [ 20 Points ] A Queue from Two Stacks.** A queue $Q$ is a dynamic collection of items that supports two operations: **Enqueue** and **Dequeue**. An **Enqueue**($Q$)($x$) operation inserts the item $x$ into $Q$, while a **Dequeue**($Q$)() operation removes the oldest item (i.e., the item that has been in the set for the longest time) from the collection. Thus a queue works on a “First In, First Out” or FIFO principle.

A stack $S$, on the other hand, is a collection of items that also supports two operations: **Push** and **Pop**. An **Push**($S$)($x$) operation inserts the item $x$ into $S$, while a **Pop**($S$)() operation removes the newest item (i.e., the item that has been in the set for the shortest time) from the collection. Thus a stack works on a “Last In, First Out” or LIFO principle.

Figure 1 shows how to implement a queue $Q$ using two stacks $Q.S_1$ and $Q.S_2$. Figure 2 shows the state of the data structure after each of a sequence of fourteen **Enqueue** and **Dequeue** operations performed on it.

Assuming that both **Push** and **Pop** operations can be performed on $Q.S_1$ and $Q.S_2$ in $\Theta(1)$ worst-case cost per operation, this task asks you to determine the worst-case and amortized costs of **Enqueue** and **Dequeue** operations on $Q$ as implemented in Figure 1. You can assume that if a **Pop** operation on an empty stack returns NIL.
Figure 2: State of $Q$ after each of a sequence of fourteen Enqueue and Dequeue operations performed on it.
1(a) [ 4 Points ] What is the worst-case cost of each of the following operations when $Q$ contains $n$ items: (i) $\text{ENQUEUE}(Q)(x)$ and (ii) $\text{DEQUEUE}(Q)()$? Justify your answers.
1(b) [ **12 Points** ] Use either the accounting method or the potential method to show that the amortized cost of each of the following operations is \( \Theta(1) \): (i) \( \text{ENQUEUE}(Q)(x) \) and (ii) \( \text{DEQUEUE}(Q)() \).
1(c) [4 Points] Suppose after executing \textsc{Init}^{(Q)} we perform an intermixed sequence of \( n \) \textsc{Enqueue} and \textsc{Dequeue} operations on \( Q \) as implemented in Figure 1. What is the total worst-case cost of performing these \( n \) operations on \( Q \) based on your results from part 1(a)? What is the total worst-case cost based on your results from part 1(b)?
**Question 2.** [30 Points] Average Deviation from a Running Max. The **Print-Max-Stat** \((A, n)\) function in Figure 3 scans the input array \(A[1:n]\) of \(n\) distinct positive numbers in order to find the largest number in the array. Every time it finds a new maximum it prints the average deviation of all the numbers seen so far from the new maximum. To reduce the number of times the print statement is executed the function first randomly permutes the numbers in \(A\).

This task asks you to find a high probability (w.r.t. \(n\)) upper bound on the running time of **Print-Max-Stat**.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>for (i \leftarrow 1) to (n-1) do</td>
</tr>
<tr>
<td>2.</td>
<td>(k \leftarrow \text{Random}(i, n))</td>
</tr>
<tr>
<td>3.</td>
<td>(A[i] \leftrightarrow A[k])</td>
</tr>
<tr>
<td>4.</td>
<td>(\max \leftarrow -\infty)</td>
</tr>
<tr>
<td>5.</td>
<td>for (i \leftarrow 1) to (n) do</td>
</tr>
<tr>
<td>6.</td>
<td>if (A[i] &gt; \max) then (\max \leftarrow A[i])</td>
</tr>
<tr>
<td>7.</td>
<td>(s \leftarrow 0) ({\text{s will store the average deviation of the numbers in } A[1:i] \text{ from } \max})</td>
</tr>
<tr>
<td>8.</td>
<td>for (j \leftarrow 1) to (i) do (s \leftarrow s + (\max - A[j])) ({\text{add to } s \text{ the deviation of } A[j] \text{ from } \max})</td>
</tr>
<tr>
<td>9.</td>
<td>(\text{print} \ \frac{s}{i}) ({\text{print average deviation}})</td>
</tr>
</tbody>
</table>

**Figure 3:** Printing average deviations from a running maximum.

2(a) [4 Points] How many times the **print** statement in line 11 will be executed in the worst case? What is the worst-case running time of **Print-Max-Stat** \((A, n)\)? What random permutation of the input numbers leads to this worst-case performance? Assume that each call to **Random** in line 2 takes only \(\Theta(1)\) time to execute.
2(b) [10 Points] Show that the expected number of times the `print` statement in line 11 will be executed during a call to `PRINT-MAX-STAT(A, n)` is $\approx \ln n$. Also show that the expected number of times the assignment statement in line 10 will be executed is $n$. 
2(c) [10 Points] Show that during a call to PRINT-MAX-STAT( A, n ) the following statements hold w.h.p. in n:

(i) the print statement in line 11 executes $\Theta(\log n)$ times, and 
(ii) the assignment statement in line 10 executes $\Theta(n)$ times.
2(d) [6 Points] Use your results from part 2(c) to give an upper bound on the running time of \textsc{Print-Max-Stat}(A, n) which holds w.h.p. in $n$. Assume that each call to \textsc{Random} in line 2 takes only $\Theta(1)$ time to execute.
Use this page if you need additional space for your answers.
**Question 3. [ 25 Points ] Largest Divisor.** Suppose $A[1:n]$ and $B[1:n]$ are two arrays containing only positive integers larger than 1, where $n > 0$. These two arrays are not necessarily sorted. Also, suppose $D[1:n]$ is another array each entry of which is initialized to 0.

In this task, for every $A[i]$ we will find the largest number $B[j]$ (if exists) such that $B[j] \neq A[i]$ but $B[j]$ divides $A[i]$, and we will store that $B[j]$ in $D[i]$, where $1 \leq i, j \leq n$. If no such entry exists in $B$ then $D[i]$ will retain its initial value 0.

**Largest-Divisor-Iterative**

(Large Inputs are two arrays $A[1:n]$ and $B[1:n]$ containing only positive integers larger than 1, where $n > 0$. Output will be written to another array $D[1:n]$ such that for $1 \leq i \leq n$, $D[i]$ will contain the largest number in $B[1:n]$ that is not equal to $A[i]$ but divides $A[i]$. If no such entry exists in $B$ then $D[i]$ will contain 0. We assume that each entry of $D$ has already been initialized to 0.)

1. for $i \leftarrow 1$ to $n$ do
2.  for $j \leftarrow 1$ to $n$ do
4.     $D[i] \leftarrow \max\{D[i], B[j]\}$

Figure 4: Finds the largest divisor of each entry of $A[1:n]$ from a given array $B[1:n]$.

3(a) [ 7 Points ] Parallelize LARGEST-DIVISOR-ITERATIVE shown in Figure 4 by replacing as many (serial) for loops with parallel for loops as possible so that it still produces correct results. You do not need to write the entire pseudocode for this parallel version. Simply modify the pseudocode in Figure 4 by (clearly and unambiguously) writing parallel to the left of the line number of each for loop you want to parallelize.

For each for loop justify your decision (in a sentence or two), i.e., if you parallelized a for loop explain why that parallel for loop will still produce correct results, and if you chose not to parallelize a for loop explain how parallelizing it will produce incorrect results.

Find the work and span of your parallel version of LARGEST-DIVISOR-ITERATIVE. Justify your answers.

(Inputs are two arrays A[1 : n] and B[1 : n] containing only positive integers larger than 1, where n > 0 and n is a power of 2. Output will be written to another array D[1 : n] such that for 1 ≤ i ≤ n, D[i] will be overwritten with the largest number in B[1 : n] that is not equal to A[i] but divides A[i], provided that number is larger than the value D[i] already contains, otherwise D[i] will remain unchanged)

1. if n = 1 then {base case}
3. \[ D[1] \leftarrow \max \{ D[1], B[1] \} \]
4. else
5. let X_L and X_R denote X[1 : \frac{n}{2}] and X[\frac{n}{2} + 1 : n], respectively, where X ∈ { A, B, D }
6. LARGEST-DIVISOR-RECURSIVE( A_L, B_L, D_L )
7. LARGEST-DIVISOR-RECURSIVE( A_R, B_L, D_R )
8. LARGEST-DIVISOR-RECURSIVE( A_L, B_R, D_L )
9. LARGEST-DIVISOR-RECURSIVE( A_R, B_R, D_R )

Figure 5: Finds the largest divisor of each entry of A[1 : n] from a given array B[1 : n]. Assumes that each entry of D[1 : n] has already been initialized to 0 before making the initial function call.

3(b) [ 10 Points ] Function LARGEST-DIVISOR-RECURSIVE shown in Figure 5 is a recursive version of LARGEST-DIVISOR-ITERATIVE.

Show how you will parallelize LARGEST-DIVISOR-RECURSIVE by writing down spawn and sync keywords at the right places in Figure 5. Justify your choices.

Write down the recurrences for the work and span of your parallel version of LARGEST-DIVISOR-RECURSIVE. Solve them to show that it performs \( \Theta(n^2) \) work and has \( \Theta(n) \) span. What is its parallelism?
3(c) [8 Points] Explain how you will improve the parallelism of LARGEST-DIVISOR-RECURSIVE. Write down the pseudocode for your improved algorithm.

Write down the recurrences for the work and span of your new algorithm and solve them. What is its parallelism?
Use this page if you need additional space for your answers.
Use this page if you need additional space for your answers.
APPENDIX I: USEFUL TAIL BOUNDS

Markov’s Inequality. Let $X$ be a random variable that assumes only nonnegative values. Then for all $\delta > 0$, $Pr \{ X \geq \delta \} \leq \frac{E[X]}{\delta}$.

Chebyshev’s Inequality. Let $X$ be a random variable with a finite mean $E[X]$ and a finite variance $Var[X]$. Then for any $\delta > 0$, $Pr \{ |X - E[X]| \geq \delta \} \leq \frac{Var[X]}{\delta^2}$.

Chernoff Bounds. Let $X_1, \ldots, X_n$ be independent Poisson trials, that is, each $X_i$ is a 0-1 random variable with $Pr[X_i = 1] = p_i$ for some $p_i$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$. Following bounds hold:

Lower Tail:
- for $0 < \delta < 1$, $Pr \{ X \leq (1 - \delta)\mu \} \leq \left( \frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}} \right)^\mu$
- for $0 < \delta < 1$, $Pr \{ X \leq (1 - \delta)\mu \} \leq e^{-\mu \frac{\delta^2}{2}}$
- for $0 < \gamma < \mu$, $Pr \{ X \leq \mu - \gamma \} \leq e^{-\frac{\gamma^2}{2\mu}}$

Upper Tail:
- for any $\delta > 0$, $Pr \{ X \geq (1 + \delta)\mu \} \leq \left( \frac{e^{\delta}}{(1+\delta)^{(1+\delta)}} \right)^\mu$
- for $0 < \delta < 1$, $Pr \{ X \geq (1 + \delta)\mu \} \leq e^{-\frac{\mu \delta^2}{3}}$
- for $0 < \gamma < \mu$, $Pr \{ X \geq \mu + \gamma \} \leq e^{-\frac{\gamma^2}{2\mu}}$

APPENDIX II: THE MASTER THEOREM

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = \begin{cases} \Theta(1), & \text{if } n \leq 1, \\ aT \left( \frac{n}{b} \right) + f(n), & \text{otherwise}, \end{cases}$$

where, $\frac{n}{b}$ is interpreted to mean either $\left\lfloor \frac{n}{b} \right\rfloor$ or $\left\lceil \frac{n}{b} \right\rceil$. Then $T(n)$ has the following bounds:

Case 1: If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.

Case 2: If $f(n) = \Theta\left(n^{\log_b a \log^k n}\right)$ for some constant $k \geq 0$, then $T(n) = \Theta\left(n^{\log_b a \log^{k+1} n}\right)$.

Case 3: If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and $af\left( \frac{n}{b} \right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta\left(f(n)\right)$.