Midterm Exam 2
( 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 midterm exam 1 and midterm exam 2. The higher of the two scores 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 twenty-two (22) pages, including nine (9) blank pages and one (1) page of appendices. Please use the blank pages if you need additional space for your answers.

• The exam is open slides and open notes (including scribe notes). But no books and no computers are allowed.

Good Luck!

<table>
<thead>
<tr>
<th>Question</th>
<th>Pages</th>
<th>Parts</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. “Probabilistic” Staircase Numbers</td>
<td>2 – 6</td>
<td>(a) – (c)</td>
<td>5 + 10 + 10 = 25</td>
<td></td>
</tr>
<tr>
<td>2. Parallel Recursive Selection Sort</td>
<td>8 – 11</td>
<td>(a) – (b)</td>
<td>10 + 15 = 25</td>
<td></td>
</tr>
<tr>
<td>3. Store-Retrieve Lockers</td>
<td>14 – 19</td>
<td>(a) – (d)</td>
<td>5 + 5 + 3 + 12 = 25</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>

Name: ____________________________________________

SBU ID: ________________________________________
Figure 1: When called with an integer $n \geq 0$ as a parameter, `Prob-Staircase(n)` will return what we will call the $n$-th “probabilistic” staircase number.

**Question 1. [25 Points] “Probabilistic” Staircase Numbers.** The function given in Figure 1 computes what we will call “probabilistic” staircase numbers. When supplied with an integer $n \geq 0$ as a parameter, it will return the $n$-th probabilistic staircase number $s_n$. Clearly, $s_0 = 0$ and $s_1 = 1$, but for $n > 1$, $s_n$ does not have a fixed value.

This question asks you to compute the expected running time of `Prob-Staircase(n)` for $n \geq 0$.

(a) [5 Points] Let $t_n$ be the expected running time of `Prob-Staircase(n)` for $n \geq 0$. We claim that $t_n$ can be described by the following recurrence relation, where $c_1$ and $c_2$ are positive constants:

$$t_n \leq \begin{cases} c_1, & \text{if } n \leq 1, \\ \frac{1}{2} t_{n-1} + \frac{1}{2} t_{n-2} + c_2, & \text{otherwise}, \end{cases}$$

Justify this recurrence.

---

1Let us not confuse these with “Polite Numbers” which are also called “Staircase Numbers.”
(b) [10 Points] Let us simplify the recurrence from part (a) to the following (by choosing \( c_1 = c_2 = 1 \) and replacing the ‘\( \leq \)’ with an ‘\( = \)’).

\[
 t_n = \begin{cases} 
 1, & \text{if } n \leq 1, \\
 \frac{1}{2} t_{n-1} + \frac{1}{2} t_{n-2} + 1, & \text{otherwise,}
\end{cases}
\]

Let \( T(z) \) be the ordinary generating function for \( t_n \), i.e.,

\[
 T(z) = t_0 + t_1 z + t_2 z^2 + \ldots + t_n z^n + \ldots
\]

Show that \( T(z) = \frac{z^2 - z + 2}{(z-1)^2(z+2)} \).
(c) [10 Points] We observe the following (you do not need to prove it):

\[ \frac{z^2 - z + 2}{(z - 1)^2(z + 2)} = \frac{2}{3(1 - z)^2} - \frac{1}{9(1 - z)} + \frac{4}{9(1 + \frac{z}{2})}. \]

Use the above and part (b) to show that

\[ t_n = \frac{1}{9} \left( 6n + 5 + 4 \left( -\frac{1}{2} \right)^n \right). \]
**Partition( A, B, n )**

**Input:** Two non-overlapping arrays A and B containing n numbers each, where n is a power of two.

**Output:** Rearrange the numbers in A and B such that no number in A is larger than any number in B.

1. if $n = 1$ then
2. if the number in A is larger than the one in B then swap the two numbers
3. else
4. let $A_L$ (resp. $B_L$) denote the left half of A (resp. B) and let $A_R$ (resp. $B_R$) denote its right half
5. $\text{Partition}( A_L, B_L, \frac{n}{2} )$
6. $\text{Partition}( A_R, B_R, \frac{n}{2} )$
7. $\text{Partition}( A_L, B_R, \frac{n}{2} )$
8. $\text{Partition}( A_R, B_L, \frac{n}{2} )$
9. return

**Rec-Selection-Sort( A, n )**

**Input:** An array A containing n numbers, where n is a power of two.

**Output:** The numbers in A rearranged in nondecreasing order of value.

1. if $n > 1$ then
2. let $A_L$ denote the left half of A and let $A_R$ denote its right half
3. $\text{Partition}( A_L, A_R, \frac{n}{2} )$
4. $\text{Rec-Selection-Sort}( A_L, \frac{n}{2} )$
5. $\text{Rec-Selection-Sort}( A_R, \frac{n}{2} )$
6. return

---

**Question 2. [ 25 Points ] Parallel Recursive Selection Sort.** When Pramod² was a student, he designed a recursive version of the selection sort algorithm with improved I/O-complexity. Figure 2 shows the high-level structure of the serial version of the algorithm. This question asks you to parallelize it and derive its parallel performance bounds.

²Pramod Ganapathi – currently a faculty member of SBUCS.
(a) [10 Points] Parallelize the PARTITION function. You can simply put the `spawn` and `sync` keywords at appropriate locations inside the function in Figure 2 to show how to parallelize it. Analyze its work, span, parallelism, and parallel running time (under a greedy scheduler).
(b) [15 Points] Parallelize the \texttt{REC-SELECTION-SORT} function. As in part (a), you can simply put the \texttt{spawn} and \texttt{sync} keywords at appropriate locations inside the function in Figure 2 to show how to parallelize it. Analyze its work, span, parallelism, and parallel running time (under a greedy scheduler).
Question 3. [ 25 Points ] Store-Retrieve Lockers. Figure 3 shows the locker data structure $L$ that maintains a resizable array $L\text{.slots}$ and supports the following two operations.

- **LOCKER-STORE( $L$, $x$ )** stores an item $x$ in a random empty slot of $L\text{.slots}$, and
- **LOCKER-RETRIEVE( $L$ )** removes an item from a random nonempty slot of $L\text{.slots}$.

Each slot stores at most one item. The total number of slots in $L\text{.slots}$ is given by $L\text{.numSlots}$,
and the number of items currently stored in the data structure is given by $L.numItems$.
The Resize-Locker ($L$) function resizes $L.slots$ as soon as one of the following two events occurs.

- **Locker-Store ($L, x$)** detects immediately after inserting $x$ that
  \[
  L.numItems \geq \frac{2}{3} \times L.numSlots
  \] (see Line 15)

- **Locker-Retrieve ($L$)** detects immediately after removing an item that
  \[
  L.numItems \leq \frac{1}{3} \times L.numSlots
  \] (see Line 14)

In both cases, $L.slots$ is resized to $L.numSlots = 2 \times L.numItems$. Observe that the smallest non-zero size $L.slots$ can have is 2 (see Lines 1–5 of Locker-Store).

To insert an item into $L$, **Locker-Store** repeatedly chooses a slot in $L.slot$ uniformly at random until it finds an empty slot and stores the item in that slot (see Lines 6–14).

To retrieve an item from $L$, **Locker-Retrieve** repeatedly chooses a slot in $L.slot$ uniformly at random until it finds a nonempty slot and removes the item from that slot (see Lines 3–12).

(a) **[5 Points]** Show that the expected number of times the while loop in Lines 7–14 of Locker-Store needs to execute to find an empty spot in $L.slots$ is \( \frac{n}{n-m} \), where \( n = L.numSlots \) and \( m = L.numItems \) at the time of execution. Also, show that the loop finds an empty spot in $O(\log n)$ iterations w.h.p. in $n$.  

15
(b) [5 Points] Show that the expected number of times the while loop in Lines 4–12 of LOCKER-RETRIEVE needs to execute to find a nonempty spot in $L.slots$ is $\frac{n}{m}$, where $n = L.numSlots$ and $m = L.numItems$ at the time of execution. Also, show that the loop finds a nonempty spot in $O(\log n)$ iterations w.h.p. in $n$. 
(c) [3 Points] In order to find the amortized costs of the operations performed on $L$ we will use the following potential function:

$$
\Phi \left( L_i \right) = c \times \left| 2 \times L.numItems - L.numSlots \right|
$$

where, $L_i$ is the state of $L$ after the $i$-th ($i \geq 0$) operation is performed on it assuming that $L$ was initially empty, and $c$ is a positive constant.

Argue that this potential function guarantees that the total amortized cost will always be an upper bound on the total actual cost.
(d) [ **12 Points** ] Use the potential function from part (c) and your results from parts (a) and (b) to show that the amortized costs of

- **Resize-Locker** is 0,
- **Locker-Store** is $O(\log n)$ w.h.p. in $n$, and
- **Locker-retrieve** is $O(\log n)$ w.h.p. in $n$,

where, $n = L.numSlots$ at the time of execution.
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 $0 < \delta < 1$, $Pr \{X \leq (1 - \delta)\mu\} \leq e^{-\frac{\mu \delta^2}{2}}$
  - 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 $0 < \delta < 1$, $Pr \{X \geq (1 + \delta)\mu\} \leq e^{-\frac{\mu \delta^2}{2}}$
  - 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 $\lfloor \frac{n}{b} \rfloor$ or $\lceil \frac{n}{b} \rceil$. 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 $T(n) = \Theta\left(f(n)\right)$. 