CSE548, AMS542: Analysis of Algorithms, Fall 2023 **Date:** Nov 30, 2023

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!

Question	Pages	Parts	Points	Score
1. "Probabilistic" Staircase Numbers	2 – 6	$(a) - (c)$	$5 + 10 + 10 = 25$	
2. Parallel Recursive Selection Sort	8 – 11	$(a) - (b)$	$10 + 15 = 25$	
3. Store-Retrieve Lockers	14 – 19	$(a) - (d)$	$5 + 5 + 3 + 12 = 25$	
Total			75	

NAME: ______________________________

SBU ID: ______________________________

```
PROB-STAIRCASE( n )
if n < 1 then x ← 0
elseif n = 1 then x ← 1
else
    r ← RANDOM( )          {choose a real number
                            uniformly at random from (0, 1]}
    if r ≤ 0.5 then
        x ← PROB-STAIRCASE( n − 1 )
    else
        x ← PROB-STAIRCASE( n − 2 )
    endif
endif
return x + 1
```

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[1]. 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.

[1] Let us not confuse these with "Polite Numbers" which are also called "Staircase Numbers."

page intentionally left blank (use for your answers, if needed)

(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 $T(z) = \frac{z^2 - z + 2}{(z-1)^2(z+2)}$.

page intentionally left blank (use for your answers, if needed)

(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

$$t_n = \frac{1}{9}\left(6n + 5 + 4\left(-\frac{1}{2}\right)^n\right)$$

.

page intentionally left blank (use for your answers, if needed)

```
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.             Partition( A_L, B_L, n/2 )
6.             Partition( A_R, B_R, n/2 )
7.             Partition( A_L, B_R, n/2 )
8.             Partition( A_R, B_L, 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.             Partition( A_L, A_R, n/2 )
4.             Rec-Selection-Sort( A_L, n/2 )
5.             Rec-Selection-Sort( A_R, n/2 )
6. return
```

Figure 2: The recursive version of the selection sort algorithm.

Question 2. [25 Points] Parallel Recursive Selection Sort. When Pramod[2] 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.

[2] Pramod 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).

$\boxed{\textbf{page intentionally left blank (use for your answers, if needed)}}$

(*b*) [**15 Points**] Parallelize the Rec-Selection-Sort function. As in part (*a*), you can simply put the ***spawn*** and ***sync*** keywords at appropriate locations inside the funtion in Figure 2 to show how to parallelize it. Analyze its work, span, parallelism, and parallel running time (under a greedy scheduler).

page intentionally left blank (use for your answers, if needed)

$\boxed{\textbf{page intentionally left blank (use for your answers, if needed)}}$

```
Resize-Locker( L )
m ← L.numItems
if m > 0 then
    allocate array newslots of size 2m
    copy the m items from L.slots to the first m slots of newslots and set the remaining m slots to NIL
else
    newslots ← NIL
endif
free L.slots
L.slots ← newslots
L.numSlots ← 2m
```

```
Locker-Store( L, x )
if L.numSlots = 0 then
    allocate L.slots with 2 slots
    L.numSlots ← 2,  L.numItems ← 0
    L.slots[1] ← NIL,  L.slots[2] ← NIL
endif
f ← False ,  n ← L.numSlots
while f = False  do
    k ← Random( 1, n )
    if L.slots[k] = NIL then
        f ← True
        L.slots[k] ← x
        L.numItems ← L.numItems + 1
    endif
endwhile
if L.numItems ≥ 2/3 × L.numSlots then
    Resize-Locker( L )
endif
```

```
Locker-Retrieve( L )
if L.numItems = 0 then x ← NIL
else
    f ← False ,  n ← L.numSlots
    while f = False  do
        k ← Random( 1, n )
        if L.slots[k] ≠ NIL then
            f ← True
            x ← L.slots[k]
            L.slots[k] ← NIL
            L.numItems ← L.numItems − 1
        endif
    endwhile
endif
if L.numItems ≤ 1/3 × L.numSlots then
    Resize-Locker( L )
endif
return x
```

Figure 3: The Locker data structure.

Question 3. [25 Points] Store-Retrieve Lockers. Figure 3 shows the locker data structure L that maintains a resizable array $L.slots$ and supports the following two operations.

- Locker-Store(L, x) stores an item x in a random empty slot of $L.slots$, and
- Locker-Retrieve(L) removes an item from a random nonempty slot of $L.slots$.

Each slot stores at most one item. The total number of slots in $L.slots$ is given by $L.numSlots$, 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

$$L.numItems \geq \frac{2}{3} \times L.numSlots \quad \text{(see Line 15)}$$

- LOCKER-RETRIEVE(L) detects immediately after removing an item that

$$L.numItems \leq \frac{1}{3} \times L.numSlots \quad \text{(see Line 14)}$$

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 $\mathcal{O}(\log n)$ iterations w.h.p. in n.

page intentionally left blank (use for your answers, if needed)

(*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 $\mathcal{O}(\log n)$ 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:

$$\Phi\,(\;L_i\;) = c \times \mid 2 \times L.numItems - L.numSlots \mid,$$

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 $\mathcal{O}(\log n)$ w.h.p. in n, and
- Locker-retrieve is $\mathcal{O}(\log n)$ w.h.p. in n,

where, $n = L.numSlots$ at the time of execution.

page intentionally left blank (use for your answers, if needed)

page intentionally left blank (use for your answers, if needed)

Appendix I: Useful Tail Bounds

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

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

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

Upper Tail:

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

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

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

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 $f(n) = \mathcal{O}\left(n^{\log_b a - \epsilon}\right)$ 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 $T(n) = \Theta\left(f(n)\right)$.