In-Class Final Exam

(2:35 PM – 3:50 PM : 75 Minutes)

- This exam will account for 25% of your overall grade.
- There are four (4) 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 two (2) pages of appendix. Please use the blank pages if you need additional space for your answers.
- The exam is *open slides*. So you may consult the lecture slides (hard copies only) during the exam. No additional cheatsheets are allowed.
- Please assume that the span of a parallel ***for*** loop is $\mathcal{O}(1+t)$, where t is the maximum span of an iteration.

Good Luck!

Question	Pages	Score	Maximum
1. Schröder Numbers	2–3		10
2. Doubly Logarithmic-Depth Tree	5–8		30
3. ϵ-Approximate Median	10–12		15
4. Matrix Transposition	14–15		20
Total			75

Name: ______________________________

Question 1. [**10 Points**] **Schröder Numbers.** For $k \geq 2$, *Schröder Number* S_k is the number of lattice paths in the Cartesian plane that go from $(1,1)$ to (k,k) without ever crossing the line $y = x$, and from any given point (x,y) moving only to one of the following three points: $(x, y+1)$, $(x+1, y)$ and $(x+1, y+1)$.

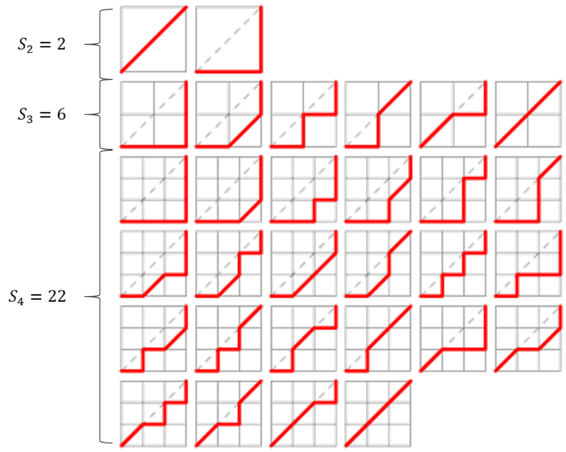

Figure 1: Schröder Numbers (Figure adapted from *Wolfram Mathworld*)

Starting from S_2 the first 20 Schröder numbers are as follows: 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098, 1037718, 5293446, 27297738, 142078746, 745387038, 3937603038, 20927156706, 111818026018, 600318853926, 3236724317174 and 17518619320890.

1(a) [**10 Points**] Schröder numbers can be computed from the following recurrence relation:

$$S_k = \begin{cases} 1 & \text{if } k < 2, \\ 3\left(2 - \frac{3}{k}\right) S_{k-1} - \left(1 - \frac{3}{k}\right) S_{k-2} & \text{otherwise.} \end{cases}$$

Describe a parallel algorithm that computes the first n Schröder numbers in $\mathcal{O}\left(\frac{n}{p} + \log n\right)$ parallel time using $\Theta(n)$ space, where p is the number of processing elements.

Hint: The recurrence relation can be rewritten as follows:

$$\left[\begin{array}{cc} S_1 & S_0 \end{array}\right] = \left[\begin{array}{cc} 1 & 1 \end{array}\right], \text{ and for } k \geq 2,\ \left[\begin{array}{cc} S_k & S_{k-1} \end{array}\right] = \left[\begin{array}{cc} S_{k-1} & S_{k-2} \end{array}\right] \left[\begin{array}{cc} 3\left(2 - \frac{3}{k}\right) & 1 \\ & \\ -\left(1 - \frac{3}{k}\right) & 0 \end{array}\right].$$

Use this page if you need additional space for your answers.

Question 2. [**30 Points**] **Doubly Logarithmic-Depth Tree.** Let $n = 2^{2^h}$ for some integer $h \geq 0$. Then a *doubly logarithmic-depth tree* ($\mathcal{DLDT}$) with n leaves has exactly $h + 2$ levels. Assuming that the root node is at level 0, each node at level $k \in [0, h-1]$ has degree $n^{\frac{1}{2^{k+1}}}$, and each node at level h has degree 2. The leaves are at level $h + 1$. See Figure 2. Looks familiar[1]?

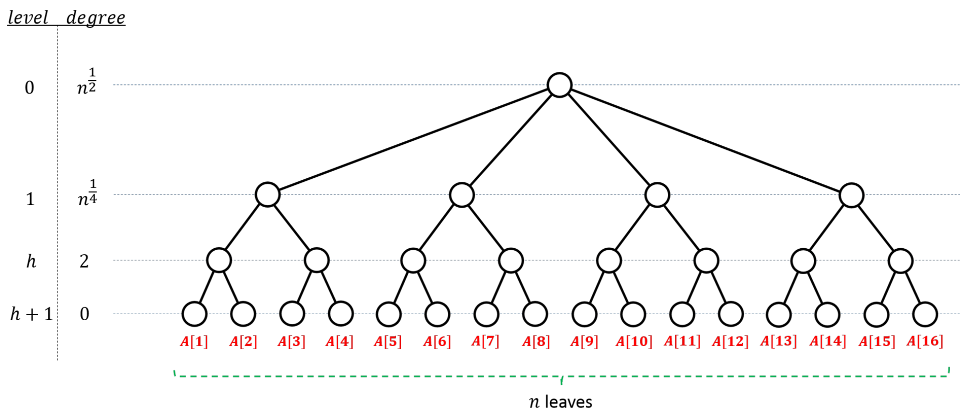

Figure 2: A doubly logarithmic-depth tree ($\mathcal{DLDT}$)

Recall that we saw in the class how to compute the maximum of n numbers in $\Theta(n)$ work and $\Theta(\log n)$ span (e.g., use the parallel prefix sums algorithm given in the first 5–6 slides of lecture 8 with **max** as the binary associative operator $\oplus$). In this problem, we will use a $\mathcal{DLDT}$ to design a parallel algorithm with a shorter span for finding the maximum in an array $A[1:n]$ of n numbers.

[1]If not, no worries. But later compare it with the structure of an n-Funnel (see slide 24 of lectures 18–19)

2(a) [**5 Points**] Given $p = n^2$ processing elements design an algorithm to find the maximum number in $A[1:n]$ in $\mathcal{O}(1)$ parallel time using $\mathcal{O}(n)$ space (without using a $\mathcal{DLDT}$).

2(b) [**5 Points**] Prove that a $\mathcal{DLDT}$ with n leaves has $n^{1-\frac{1}{2^k}}$ internal nodes at level $k \in [0, h]$.

2(c) [**10 Points**] Suppose we build a $\mathcal{DLDT}$ with n leaves, where the i-th leaf from the left contains the number $A[i]$, $1 \leq i \leq n$ (see Figure 2). Now given $p = n$ processing elements design a parallel algorithm that terminates in $\Theta\left(\log \log n\right)$ parallel time, and for each internal node in the $\mathcal{DLDT}$ computes the maximum number stored among the leaves of the subtree rooted at that node. Thus the root of the $\mathcal{DLDT}$ will hold the maximum number in $A[1:n]$.

Hint: Use your results from parts 2(a) and 2(b).

2(d) [**10 Points**] Is your algorithm from part 2(c) work-optimal? If not, how do you make it work-optimal?

Hint: Use a $\Theta(n)$ work and $\Theta(\log n)$ span algorithm we saw in the class (e.g., the prefix sums algorithm) to reduce the size of the input array first.

Use this page if you need additional space for your answers.

Question 3. [15 Points] ϵ-Approximate Median. Let $A[\ 1:n\ ]$ be an array of n distinct numbers. For any number x, we define $rank(\ x\ )$ to be the number of items in A that are not larger than x, i.e., $rank(\ x\ ) = |\ \{\ A[\ i\ ] \mid 1 \le i \le n\ \wedge\ A[\ i\ ] \le x\}\ |$.

For any $\epsilon \in (0, \frac{1}{2}]$, an *$\epsilon$-approximate median* of A is a number x with $rank(\ x\ ) \in \left(\ \frac{n}{2} - \epsilon n,\ \frac{n}{2} + \epsilon n\ \right)$.

Recall that deterministically finding the exact median of A requires time linear in n. In this problem we will see that an ϵ-approximate median (w.h.p. in n) of A can be found in time logarithmic in n.

Approx-Median($A[\ 1:n\ ],\ \epsilon$)

(Inputs are an array $A[\ 1:n\ ]$ of n distinct numbers, and a floating point parameter $\epsilon \in (0, \frac{1}{4}]$. This routine chooses a sample of size $\lceil \frac{14}{\epsilon^2} \log n \rceil$ from A (with replacement), and returns the median of that sample.)

$m \leftarrow \lceil \frac{14}{\epsilon^2} \log n \rceil$ {*size of the sample*}
array $B[\ 1:m\ ]$ {*array to store the sample*}
for $i \leftarrow 1$ ***to*** m ***do*** {*sample m items (with replacement) from A*}
    $j \leftarrow$ Random($1, n$) {*choose an integer uniformly at random from $[\ 1, n\ ]$*}
    $B[\ i\ ] \leftarrow A[\ j\ ]$ {*choose $A[\ j\ ]$ as the next sample from A*}
$x \leftarrow$ Median($B[\ 1:m\ ]$) {*find the median of $B[\ 1:m\ ]$ using a linear time selection algorithm*}
return x

Figure 3: Find an ϵ-approximate median of $A[\ 1:n\ ]$.

3(a) [**10 Points**] Consider the function APPROX-MEDIAN given in Figure 3 which runs in $\Theta\left(\frac{1}{\epsilon^2}\log n\right)$ worst-case time.

$$\text{Let } \mathcal{L} = \left\{ A[\ i\ ] \mid i \in [\ 1, n\ ]\ \wedge\ rank(\ A[\ i\ ]\ ) \leq \frac{n}{2} - \epsilon n \right\}$$

$$\text{and } \mathcal{H} = \left\{ A[\ i\ ] \mid i \in [\ 1, n\ ]\ \wedge\ rank(\ A[\ i\ ]\ ) \geq \frac{n}{2} + \epsilon n \right\}.$$

Suppose $B[\ 1 : m\ ]$ contains l elements from $\mathcal{L}$, and h elements from $\mathcal{H}$. Now assuming $\epsilon \in \left(\ 0,\ \frac{1}{4}\ \right]$, prove that

$$\Pr\left[\ l < \frac{m}{2}\ \right] > 1 - \frac{1}{n^{\frac{7}{6}}} \quad \text{and} \quad \Pr\left[\ h < \frac{m}{2}\ \right] > 1 - \frac{1}{n^{\frac{7}{6}}}.$$

Hint: $\Pr\left[\ l \geq \frac{m}{2}\ \right] \leq \Pr\left[\ l \geq (1+\epsilon)\mu\ \right]$, where $\mu = m\left(\frac{1}{2} - \epsilon\right)$.

3(b) [**5 Points**] Use your results from part 3(a) to argue that for $\epsilon \in \left(0, \ \frac{1}{4} \right]$, Approx-Median returns an ϵ-approximate median of $A[\ 1 : n\ ]$ w.h.p. in n.

Use this page if you need additional space for your answers.

QUESTION 4. [20 Points] Matrix Transposition. The *transpose* of a matrix X is another matrix X^T obtained by writing the rows of X as the columns of X^T. An example is given below.

$$X = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix} \qquad \Rightarrow \qquad X^T = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_4 & b_4 & c_4 & d_4 \end{bmatrix}$$

In this problem we will analyze the cache complexity of a couple of algorithms for transposing square matrices.

$4(a)$ [**5 Points**] Analyze the cache complexity of ITER-MATRIX-TRANSPOSE given in Figure 4.

ITER-MATRIX-TRANSPOSE(X, Y, n)
(Input is an $n \times n$ square matrix $X[\,1:n,\ 1:n\,]$. This function generates the transpose of X in Y.)

1. ***for*** $i \leftarrow 1$ ***to*** n ***do***
2. ***for*** $j \leftarrow 1$ ***to*** n ***do***
3. $Y[\,i,\ j\,] \leftarrow X[\,j,\ i\,]$

Figure 4: Iterative matrix transposition.

4(b) [**10 Points**] Complete the recursive divide-and-conquer algorithm (Rec-Matrix-Transpose) for transposing a square matrix given in Figure 5. Analyze its cache complexity assuming a *tall* cache (i.e., $M = \Omega\left(B^2\right)$, where M is the cache size and B is the cache block size).

Rec-Matrix-Transpose(X, Y, n)

(Input is an $n \times n$ square matrix $X[\ 1 : n,\ 1 : n\ ]$. This function recursively generates the transpose of X in Y. We assume $n = 2^k$ for some integer $k \geq 0$. If $n > 1$, let X_{11}, X_{12}, X_{21} and X_{22} denote the top-left, top-right, bottom-left and bottom-right quadrants of X, respectively. Similarly for Y.)

1. ***if*** $n = 1$ ***then*** $Y \leftarrow X$ {*base case: the transpose of a* 1×1 *matrix is the matrix itself*}
2. ***else*** {*divide* X *and* Y *into quadrants, and generate the transpose of* X *recursively.*}
3. Rec-Matrix-Transpose(, ,) {*fill out*}
4. Rec-Matrix-Transpose(, ,) {*fill out*}
5. Rec-Matrix-Transpose(, ,) {*fill out*}
6. Rec-Matrix-Transpose(, ,) {*fill out*}

Figure 5: Iterative matrix transposition.

4(c) [**5 Points**] Is the cache complexity result of part 4(b) optimal? Why or why not?

Use this page if you need additional space for your answers.

Appendix: Prefix Sums

Input. A sequence of n elements $x_1, x_2, \ldots x_n$ drawn from a set S with a binary associative operation (e.g., addition, multiplication, maximum, matrix product, union, etc.), denoted by $\oplus$.

Output. A sequence of n partial sums $s_1, s_2, \ldots s_n$, where $s_i = x_1 \oplus x_2 \oplus \ldots x_i$ for $1 \leq i \leq n$.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
5	3	7	1	3	6	2	4

$\oplus$ = binary addition

5	8	15	16	19	25	27	31
s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8

```
Prefix-Sum ( ⟨x_1, x_2, …, x_n⟩, ⊕ )    { n = 2^k for some k ≥ 0.
                                          Return prefix sums
                                          ⟨s_1, s_2, …, s_n⟩ }
if n = 1 then
    s_1 ← x_1
else
    parallel for i ← 1 to n/2 do
        y_i ← x_{2i−1} ⊕ x_{2i}
    ⟨z_1, z_2, …, z_{n/2}⟩ ← Prefix-Sum( ⟨y_1, y_2, …, y_{n/2}⟩, ⊕ )
    parallel for i ← 1 to n do
        if i = 1 then s_1 ← x_1
        else if i = even then s_i ← z_{i/2}
             else s_i ← z_{(i−1)/2} ⊕ x_i
return ⟨s_1, s_2, …, s_n⟩
```

Figure 6: A parallel prefix sums algorithm with $\Theta(n)$ work and $\Theta(\log n)$ span (from lecture 8).

Appendix: 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 $X = \displaystyle\sum_{i=1}^{n} X_i$ and $\mu = E[X]$. Then the following bounds hold.

(1) For any $\delta > 0$, $Pr\left[X \geq (1+\delta)\mu\right] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}$.

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

(3) For $0 < \gamma < \mu$, $Pr\left[X \geq \mu + \gamma\right] \leq e^{-\frac{\gamma^2}{3\mu}}$.

(4) For $0 < \delta < 1$, $Pr\left[X \leq (1-\delta)\mu\right] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mu}$.

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

(6) For $0 < \gamma < \mu$, $Pr\left[X \leq \mu - \gamma\right] \leq e^{-\frac{\gamma^2}{2\mu}}$.