Midterm Exam 1

(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 four (4) questions worth 75 points in total. Please answer all of them in the spaces provided.
- There are twenty-two (22) pages, including eight (8) 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. Miles and Miles of Tiles	2 – 6	$(a) - (c)$	$4 + 15 + 6 = 25$	
2. Multiply Multiple Multipliers	8 – 10	$(a) - (b)$	$6 + 9 = 15$	
3. Ugly Recurrences	12 – 17	$(a) - (e)$	$2 + 2 + 6 + 6 + 9 = 25$	
4. $(\vee, \wedge)$ Bitwise Matrix Multiplication	19 – 20	$(a) - (b)$	$3 + 7 = 10$	
Total			75	

Name: ____

SBU ID: ________________________________

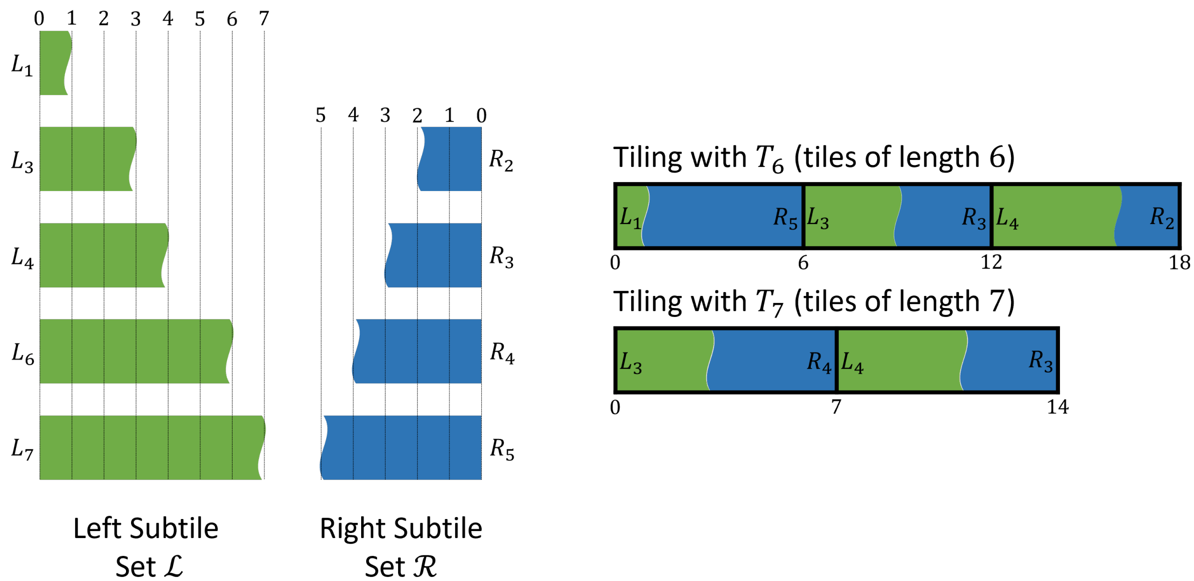

Figure 1: [Question 1] A subtile L_i of length i from the left subtile set $\mathcal{L}$ can be combined with a subtile R_j of length j from the right subtile set $\mathcal{R}$ to form a full tile T_{i+j} of length $i+j$. Using the subtiles given in this example, one can create exactly three T_6 tiles covering a length of $3 \times 6 = 18$, but only two T_7 tiles covering a length of only $2 \times 7 = 14$.

Question 1. [25 Points] Miles and Miles of Tiles. You are given subtiles or tile fragments of two specific types – left subtiles and right subtiles. All subtiles have the same width but not necessarily the same length. A left (resp. right) subtile of length k is denoted by L_k (resp. R_k). An L_i can be combined with an R_j to form a full tile T_{i+j} of length $i+j$. We assume that all subtiles have integral lengths.

You are given an integer $n > 0$, a left subtile set $\mathcal{L}$, and a right subtile set $\mathcal{R}$. For every integer $k \in [1, n]$, $\mathcal{L}$ (resp. $\mathcal{R}$) includes at most one L_k (resp. R_k). Your task is to find for every $k \in [2, 2n]$, the total length d_k you can tile by using only the full tiles of length k (i.e., T_k's) that you can form by combining the left subtiles of $\mathcal{L}$ with the right subtiles of $\mathcal{R}$. Figure 1 shows an example. Using the sets given in the example, one can create exactly three tiles of length 6 (i.e.,, T_6) covering a total length of $d_6 = 3 \times 6 = 18$. But one can create only two tiles of length 7 (i.e., T_7) that cover a total length of only $d_7 = 2 \times 7 = 14$.

Now answer the following questions.

(a) [**4 Points**] Given integer $n > 0$ and sets $\mathcal{L}$ and $\mathcal{R}$, give an algorithm that can compute all d_k values for $2 \leq k \leq 2n$ in $\Theta(n + n_l n_r)$ time, where n_l and n_r denote the number of subtiles in $\mathcal{L}$ and $\mathcal{R}$, respectively.

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

(*b*) [**15 Points**] Give an algorithm that computes all d_k values for $2 \le k \le 2n$, in $\Theta(n \log n)$ time.

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

(c) [**6 Points**] Now suppose that $\mathcal{L}$ and $\mathcal{R}$ can have at most $m \geq 1$ copies of each subtile[1], and the number of copies of each subtile appearing in $\mathcal{L}$ / $\mathcal{R}$ is a power of 2. For this case, give an algorithm that can compute all d_k values for $2 \leq k \leq 2n$, in $\mathcal{O}\left(n \log n (\log(m+1))^2\right)$ time.

[1] So, $\mathcal{L}$ and $\mathcal{R}$ are multisets in which no item appears more than m times.

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

Question 2. [**15 Points**] **Multiply Multiple Multipliers.** Karatsuba's algorithm multiplies two n-bit integers in $\Theta\left(n^{\log_2 3}\right)$ time. This problem asks you to use Karatsuba's algorithm to multiply $m \geq 2$ binary integers each of which is n bits long.

Input: binary integers $x_1, x_2, \ldots, x_m$ each exactly n bits long
Output: product $x_1 x_2 \ldots x_m$

Algorithm:

1. $z \leftarrow x_1$
2. **for** $i \leftarrow 2$ **to** m **do**
3. $\quad t \leftarrow z \times x_i$ computed using Karatsuba's algorithm
4. $\quad z \leftarrow t$
5. **return** z

Figure 2: [Question 2] Naïvely computing the product of m binary numbers of length n each.

(a) [**6 Points**] Show that the algorithm given in Figure 2 takes $\Theta\left(m(mn)^{\log_2 3}\right)$ time to multiply $m \geq 2$ binary integers containing n bits each.

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

(*b*) [**9 Points**] Give a recursive divide-and-conquer algorithm that runs in $\Theta\left((mn)^{\log_2 3}\right)$ time to multiply $m \geq 2$ binary integers each of which is n bits long. You must use Karatsuba's algorithm whenever multiplying two integers. Write down the recurrence relation describing the running time of the algorithm and solve it.

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

Question 3. [25 Points] Ugly Recurrences. This problem asks you to use Akra-Bazzi to solve ugly recurrences of the following form[2].

$$T(n) = \begin{cases} \Theta(1), & \text{if } 2 \le n \le n_0, \\ \sum_{i=1}^{k} a_i n^{\alpha(1-b_i)} T\left(n^{b_i}\right) + \Theta\left(n^{\alpha}(\log n)^{\beta}\right), & \text{otherwise,} \end{cases}$$

where, $k \ge 1$ is an integer constant; $a_i > 0$ and $b_i \in (0,1)$ are constants for $1 \le i \le k$; $n \ge 2$ is a real number; α and β are real constants; $n_0 \ge 2$ is a constant and $n_0 \ge 2^{\max\left\{\frac{1}{b_i}, \frac{1}{1-b_i}\right\}}$ for $1 \le i \le k$. Let p be the unique real number for which $\sum_{i=1}^{k} a_i b_i^p = 1$.

(a) [**2 Points**] Suppose $T'(n) = \frac{T(n)}{n^{\alpha}}$. Show that

$$T'(n) = \begin{cases} \Theta(1), & \text{if } 2 \le n \le n_0, \\ \sum_{i=1}^{k} a_i T'\left(n^{b_i}\right) + \Theta\left((\log n)^{\beta}\right), & \text{otherwise.} \end{cases}$$

[2]Recurrences of this form appear in the analysis of running times of several FFT variants, column sort, etc.

(b) [**2 Points**] Suppose $n = 2^x$, $n_0 = 2^{x_0}$, and $T''(x) = T'(2^x)$. Show that

$$T''(x) = \begin{cases} \Theta\left(1\right), & \text{if } 1 \le x \le x_0, \\ \sum_{i=1}^{k} a_i T''\left(b_i x\right) + \Theta\left(x^{\beta}\right), & \text{otherwise.} \end{cases}$$

(c) [**6 Points**] Use the Akra-Bazzi formula to show that the recurrence from part (b) has the following solutions:

$$T''(x) = \begin{cases} \Theta\left(x^{\beta}\log x\right), & \text{if } p = \beta, \\ \Theta\left(\left(1 - \frac{1}{\beta - p}\right)x^{p} + \left(\frac{1}{\beta - p}\right)x^{\beta}\right), & \text{if } p \neq \beta. \end{cases}$$

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

(d) [**6 Points**] Use the solution from part (c) to show that

$$T(n) = \begin{cases} \Theta\left(n^{\alpha}(\log n)^{\beta}\right), & \text{if } p < \beta, \\ \Theta\left(n^{\alpha}(\log n)^{\beta}\log\log n\right), & \text{if } p = \beta, \\ \Theta\left(n^{\alpha}(\log n)^{p}\right), & \text{if } p > \beta. \end{cases}$$

(*e*) [**9 Points**] Now use the solution from part (*d*) to solve the following recurrences:

$$T_1(n) = \begin{cases} \Theta\left(1\right), & \text{if } 2 \leq n \leq n_0, \\ n^{\frac{1}{2}} T_1\left(n^{\frac{1}{2}}\right) + \Theta\left(n\right), & \text{otherwise,} \end{cases}$$

$$T_2(n) = \begin{cases} \Theta\left(1\right), & \text{if } 2 \leq n \leq n_0, \\ n^{\frac{1}{3}} T_2\left(n^{\frac{2}{3}}\right) + n^{\frac{2}{3}} T_2\left(n^{\frac{1}{3}}\right) + \Theta\left(n \log n\right), & \text{otherwise,} \end{cases}$$

$$T_3(n) = \begin{cases} \Theta\left(1\right), & \text{if } 2 \leq n \leq n_0, \\ 2n^{\frac{15}{8}} T_3\left(n^{\frac{1}{16}}\right) + n^{\frac{3}{2}} T_3\left(n^{\frac{1}{4}}\right) + \Theta\left(n^2 \log n\right), & \text{otherwise,} \end{cases}$$

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

$$\begin{vmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{vmatrix} \otimes \begin{vmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

Figure 3: [Question 4] Bitwise product of two 3×3 bit matrices.

Question 4. [10 Points] $(\vee, \wedge)$ **Bitwise Matrix Multiplication.** Suppose for some positive integer n, you are given two $n \times n$ matrices X and Y in which every entry is a single bit (either 0 or 1). Therefore, each matrix occupies exactly n^2 bits. You multiply X and Y using bitwise OR $(\vee)$ and bitwise AND $(\wedge)$ operators only. You end up with an $n \times n$ product matrix Z in which each entry is a single bit, and for $1 \leq i, j \leq n$, entry $z_{i,j}$ of Z is defined as follows, where $x_{i,k}$'s and $y_{k,j}$'s are entries of X and Y, respectively.

$$\begin{aligned} z_{i,j} &= \bigvee_{k=1}^{n} (x_{i,k} \wedge y_{k,j}) \\ &= (x_{i,1} \wedge y_{1,j}) \vee (x_{i,2} \wedge y_{2,j}) \vee (x_{i,3} \wedge y_{3,j}) \vee \ldots \vee (x_{i,n} \wedge y_{n,j}) \end{aligned}$$

This product is similar to the product in the standard matrix multiplication algorithm we saw in the class, except that we have replaced the '$\times$' and '$+$' operators with '$\wedge$' and '$\vee$' operators, respectively. Clearly, all entries of Z can be computed in $\Theta\left(n^3\right)$ time using a naïve looping code.

Figure 3 shows an example, where we use the $\otimes$ operator to indicate that this is not standard matrix multiplication.

Now, answer the following questions.

(*a*) **[3 Points]** It turns out that the standard $\Theta\left(n^3\right)$ time recursive matrix multiplication algorithm that we saw in the class can be easily modified (by replacing '$\times$' and '$+$' with '$\wedge$' and '$\vee$', respectively) to correctly compute the bitwise product of X and Y as defined above using only $\Theta\left(n^2\right)$ bits of space. However, Strassen's algorithm cannot be used to compute Z in $\Theta\left(n^2\right)$ bits of space using those bitwise operators. Why?

(b) [**7 Points**] Suppose I allow you to use $\Theta\left(n^2 \log n\right)$ bits of space. Now, can you use Strassen's algorithm without replacing the standard '$\times$' and '$+$' operators to compute Z correctly? How?

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

Appendix: Recurrences

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

Akra-Bazzi Recurrences. Consider the following recurrence:

$$T(x) = \begin{cases} \Theta(1), & \text{if } 1 \leq x \leq x_0, \\ \sum_{i=1}^{k} a_i T(b_i x) + g(x), & \text{otherwise}, \end{cases}$$

where,

1. $k \geq 1$ is an integer constant,
2. $a_i > 0$ is a constant for $1 \leq i \leq k$,
3. $b_i \in (0, 1)$ is a constant for $1 \leq i \leq k$,
4. $x \geq 1$ is a real number,
5. x_0 is a constant and $\geq \max\left\{\frac{1}{b_i}, \frac{1}{1-b_i}\right\}$ for $1 \leq i \leq k$, and
6. $g(x)$ is a nonnegative function that satisfies a polynomial growth condition (e.g., $g(x) = x^\alpha \log^\beta x$ satisfies the polynomial growth condition for any constants $\alpha, \beta \in \Re$).

Let p be the unique real number for which $\sum_{i=1}^{k} a_i b_i^p = 1$. Then $T(x) = \Theta\left(x^p \left(1 + \int_1^x \frac{g(u)}{u^{p+1}} du\right)\right)$.

Appendix: Computing Products

Integer Multiplication. Karatsuba's algorithm can multiply two n-bit integers in $\Theta\left(n^{\log_2 3}\right) = \mathcal{O}\left(n^{1.6}\right)$ time (improving over the standard $\Theta\left(n^2\right)$ time algorithm).

Matrix Multiplication. Strassen's algorithm can multiply two 2×2 matrices using 7 multiplications, and two $n \times n$ matrices in $\Theta\left(n^{\log_2 7}\right) = \mathcal{O}\left(n^{2.81}\right)$ time (improving over the standard $\Theta\left(n^3\right)$ time algorithm).

Polynomial Multiplication. One can multiply two n-degree polynomials in $\Theta(n \log n)$ time using the FFT (Fast Fourier Transform) algorithm (improving over the standard $\Theta\left(n^2\right)$ time algorithm).