In-Class Midterm
( 2:35 PM – 3:50 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 four (4) questions, worth 75 points in total. Please answer all of them in the spaces provided.

- There are 16 pages including four (4) blank pages and two (2) pages of appendices. Please use the blank pages if you need additional space for your answers.

- The exam is open slides.

Good Luck!

<table>
<thead>
<tr>
<th>Question</th>
<th>Pages</th>
<th>Score</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Counting Paths</td>
<td>2–4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2. A Schönhage-Strassen-like Recurrence</td>
<td>6–8</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>3. Closest Pair of Points</td>
<td>10–11</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4. An Impossible Priority Queue</td>
<td>13</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>
**Question 1. [ 20 Points ] Counting Paths.** Suppose you are given two directed graphs $G_1$ and $G_2$ containing $n + 2$ nodes each for some $n \geq 0$. For $i \in \{1, 2\}$, $G_i$ includes two special nodes — a source node $s_i$ with no incoming edges and a target node $t_i$ with no outgoing edges. These two nodes are called **external nodes** while the rest are called **internal nodes**. The figure below shows an example with $n = 5$ in which the internal nodes are colored grey and the external nodes are white. Let $g_i(k)$ denote the number of paths in $G_i$ that go from $s_i$ to $t_i$ and pass through exactly $k$ internal (i.e., grey) nodes. For example, in the figure below $g_1(3) = 4$ which represents the following 4 paths:

\[
s_1 \rightarrow a_1 \rightarrow b_1 \rightarrow e_1 \rightarrow t_1, \\
s_1 \rightarrow a_1 \rightarrow c_1 \rightarrow b_1 \rightarrow t_1, \\
s_1 \rightarrow c_1 \rightarrow b_1 \rightarrow e_1 \rightarrow t_1 \\
and s_1 \rightarrow c_1 \rightarrow d_1 \rightarrow e_1 \rightarrow t_1.
\]

Suppose for $0 \leq k \leq n$, all $g_1(k)$ and $g_2(k)$ values are known to you.

Now suppose you connect $G_1$ and $G_2$ by putting an edge directed from $t_1$ to $s_2$. For $0 \leq k \leq 2n$, let $g_{12}(k)$ denote the number of paths from $s_1$ to $t_2$ that pass through exactly $k$ internal (i.e., grey) nodes. The figure above shows an example in which $g_{12}(3) = 5$ representing the following 5 paths:

\[
(s_1 \rightarrow c_1 \rightarrow t_1) \rightarrow (s_2 \rightarrow c_2 \rightarrow b_2 \rightarrow t_2), \\
(s_1 \rightarrow c_1 \rightarrow t_1) \rightarrow (s_2 \rightarrow d_2 \rightarrow c_2 \rightarrow t_2), \\
(s_1 \rightarrow a_1 \rightarrow b_1 \rightarrow t_1) \rightarrow (s_2 \rightarrow d_2 \rightarrow t_2), \\
(s_1 \rightarrow a_1 \rightarrow c_1 \rightarrow t_1) \rightarrow (s_2 \rightarrow d_2 \rightarrow t_2) \\
and (s_1 \rightarrow c_1 \rightarrow b_1 \rightarrow t_1) \rightarrow (s_2 \rightarrow d_2 \rightarrow t_2).
\]

---

1 e.g., road networks with one-way roads
2 e.g., incoming roads
3 e.g., outgoing roads
1(a) [ 5 Points ] For any given integer $k \in [0, 2n]$, show that $g_{12}(k)$ can be computed from $g_1$‘s and $g_2$‘s in $O(n)$ time.
1(b) [ 15 Points ] Show that for $0 \leq k \leq 2n$, one can compute all $g_{12}(k)$ values simultaneously in $\mathcal{O}(n \log n)$ time.
Use this page if you need additional space for your answers.
**Question 2. [25 Points] A Schönhage-Strassen-like Recurrence.** Consider the following recurrence (for $n \geq 2$) which is similar to the recurrence that arises during the analysis of the Schönhage-Strassen algorithm for multiplying large integers.

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

2(a) [4 Points] Show that the recurrence above can be rewritten as follows, where $T(n) = nS(n)$.

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

2(b) [4 Points] Show that the recurrence in 2(a) can be rewritten as follows, where $P(x) = S(2^x)$.

$$P(x) = \begin{cases} 
\Theta(1) & \text{if } 1 \leq x \leq 3, \\
 P \left( \frac{x}{3} \right) + P \left( \frac{2x}{3} \right) + \Theta(x) & \text{otherwise}.
\end{cases}$$
2(c) [9 Points] Solve the recurrence from part 2(b) to show that $P(x) = \Theta(x \log x)$. 
Use your results from part 2(c) to show that $T(n) = \Theta(n \log n \log \log n)$. 
Use this page if you need additional space for your answers.
**Question 3. [ 20 Points ] Closest Pair of Points.** Consider the algorithm **Closest-Pair** given below that finds the closest pair of points among a given set of points in the plane.

**Closest-Pair( \( P, n \) )**

**Input:** A set \( P = \{ p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n) \} \) of \( n \) points in the plane. Assume for simplicity that (a) \( n = 2^k \) for some integer \( k > 0 \), (b) all \( x_i \)'s are distinct, and (c) all \( y_i \)'s are distinct.

**Output:** Two distinct points \( p_i, p_j \in P \) such that the distance between \( p_i \) and \( p_j \) is the smallest among all pairs of points in \( P \).

**Algorithm:**

1. if \( n = 2 \) then return \( \langle p_1, p_2 \rangle \)
2. else
3. Find a value \( x \) such that exactly \( \frac{n}{2} \) points in \( P \) have \( x_i < x \), and the other \( \frac{n}{2} \) points have \( x_i > x \)
4. Let \( L \) be the subset of \( P \) containing all points with \( x_i < x \)
5. Let \( R \) be the subset of \( P \) containing all points with \( x_i > x \)
6. \( \langle p_L, q_L \rangle \leftarrow \text{Closest-Pair}( L, \frac{n}{2} ) \)
7. \( \langle p_R, q_R \rangle \leftarrow \text{Closest-Pair}( R, \frac{n}{2} ) \)
8. \( d_L \leftarrow \text{distance between } p_L \text{ and } q_L \)
9. \( d_R \leftarrow \text{distance between } p_R \text{ and } q_R \)
10. \( d \leftarrow \min \{ d_L, d_R \} \)
11. Scan \( P \) and remove each \( p_i = (x_i, y_i) \in P \) with \( x_i < x - d \) or \( x_i > x + d \)
12. Sort the remaining points of \( P \) in increasing order of \( y \)-coordinates
13. Scan the sorted list, and for each point compute its distance to the 7 subsequent points in the list. Let \( \langle p_M, q_M \rangle \) be the closest pair of points found in this way.
14. Let \( \langle p, q \rangle \) be the closest pair among \( \langle p_L, q_L \rangle, \langle p_R, q_R \rangle \) and \( \langle p_M, q_M \rangle \)
15. return \( \langle p, q \rangle \)

3(a) [ 10 Points ] Argue that for a set of \( n \) points, steps 3–5 take \( O(n) \) time while steps 8–15 take \( O(n \log n) \) time.
3(b) [10 Points] Let $T(n)$ be the running time of CLOSEST-PAIR on a set of $n$ points. Write a recurrence relation for $T(n)$ and solve it.
Use this page if you need additional space for your answers.
Question 4. [10 Points] An Impossible Priority Queue. Consider a (comparison-based) priority queue $Q$ (for real numbers) that supports the following operations.

- **MAKE-QUEUE($Q$):** Create an empty queue $Q$.
- **INSERT($Q, x$):** Insert item $x$ into $Q$.
- **INCREASE-KEY($Q, x, k$):** Increase the key of item $x$ to $k$ assuming $k \geq$ current key of $x$.
- **FIND-MIN($Q$):** Return a pointer to an item in $Q$ containing the smallest key.
- **DELETE-MIN($Q$):** Delete an item with the smallest key from $Q$ and return a pointer to it.

4(a) [10 Points] Suppose $Q$ supports **INSERT** and **INCREASE-KEY** operations in $O(1)$ amortized time each, and **DELETE-MIN** operations in $O(\log n)$ worst-case time each, where $n$ is the number of items in $Q$. It also supports the **MAKE-QUEUE** operation and every **FIND-MIN** operation in $O(1)$ worst-case time.

Argue that such a priority queue cannot exist.
Use this page if you need additional space for your answers.
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 $\lfloor \frac{n}{b} \rfloor$ 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 $T(n) = \Theta\left(n^{\log_b a \log k + 1}\right)$.

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 $T(n) = \Theta\left(f(n)\right)$.

Akra-Bazzi Recurrences. Consider the following recurrence:

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

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 $\alpha, \beta \in \mathbb{R}$).

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 $\Theta(n^{\log_2 3}) = \mathcal{O}(n^{1.6})$ time (improving over the standard $\Theta(n^2)$ time algorithm).

**Matrix Multiplication.** Strassen’s algorithm can multiply two $n \times n$ matrices in $\Theta(n^{\log_2 7}) = \mathcal{O}(n^{2.81})$ time (improving over the standard $\Theta(n^3)$ 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 $\Theta(n^2)$ time algorithm).