In-Class Midterm

(1:05 PM – 2:20 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 three (3) 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* and *open notes*.

Good Luck!

Question	Pages	Score	Maximum
1. Can You See the Statue of Liberty from Manhattan?	2–7		35
2. Approximate Triangulations Counting and More	9–10		20
3. Multiplying Matrices with Polynomial Entries	12		20
Total			75

Name: _____

Question 1. [35 Points] Can You See the Statue of Liberty from Manhattan? Yes, you can. But not from everywhere in Manhattan. The problem is Manhattan is full of tall buildings, and even if you are on the top of a very tall building, often another building that is even taller will obstruct your view. In this task we will consider every building on an imaginary straight line[1] L between an observer and the statue, and for each such building find out if other buildings block its view of the statue.

We make the following simplifying assumptions.

(i) Only heights of buildings (and not their widths) have any impact on our visibility calculations.

(ii) No two buildings on L have the same height, and all buildings are taller than the statue.

(iii) If buliding y lies between building x and the statue on L, and y is taller than x then no one in x can see the statue.

(iv) If x and y are two buildings on L, then y is visible from x iff all buildings on L between x and y are shorter than y.

Suppose there are $n > 0$ buildings on L numbered from 1 to n with building i being the i-th closest building to the statue. For $i \in [1, n]$, the height of building i is stored in $B[i].h$.

We say that a building x is "Lucky" provided no other building obstructs its view of the statue, that is, all buildings on L between x and the statue are shorter than x.

```
MARK-LUCKY-BUILDINGS( B[1 : n] )      {For i ∈ [1, n], height of building i is given in B[i].h, where building 1 is
                                       the closest to the statue and building n is the farthest from it on L.
                                       This algorithm sets B[i].p = 0 provided no other building is obstructing
                                       building i's view of the statue, otherwise B[i].p is set to the index of
                                       the building closest to i obstructing its view.}
1. B[1].p ← 0                         {no other bulding is obstructing building 1's view of the statue}
2. for i ← 2 to n do
3.     B[i].p ← FIND-CLOSEST-TALLER-BUILDING ( B[1 : n], i )    {now B[i].p = 0 if i's view is not obstructed,
                                       otherwise B[i].p contains the index of the building closest to i obstructing its view of the statue.}
```

```
FIND-CLOSEST-TALLER-BUILDING( B[1 : n], i )    {Given i ∈ [2, n], returns the largest j ∈ [1, i − 1] with
                                                B[j].h > B[i].h if such a j exists, otherwise returns 0.}
1. j ← i − 1                          {building closest to building i in the direction of the statue}
2. while (B[j].h < B[i].h) or (B[j].p > 0) do    {skip shorter buildings that cannot see the statue}
3.     j ← B[j].p                     {jump to the building blocking building j's view}
4. endwhile
5. if B[j].h > B[i].h then return j   {j is the building closest to i blocking i's view of the statue}
6. else return 0                      {building i can see the statue}
```

[1] line of sight

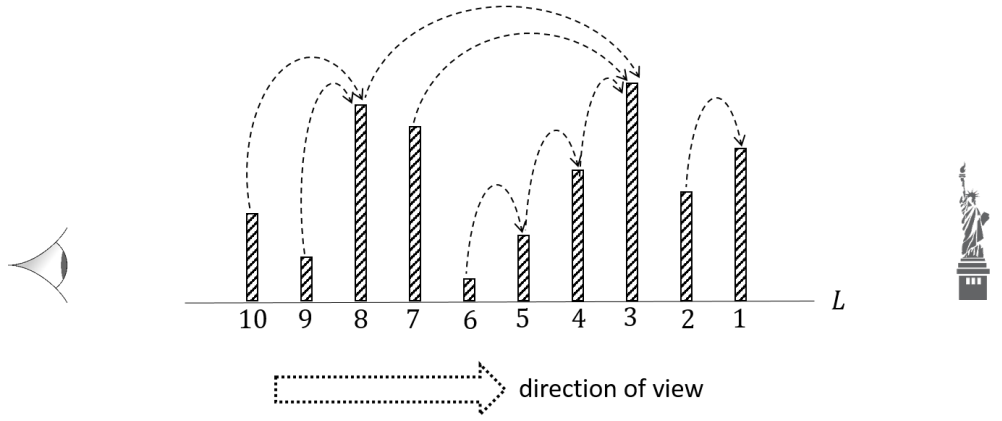

Figure 1: After executing MARK-LUCKY-BUILDINGS, every building points to the taller building closest to it (if one exists) in the direction of the statue.

1(a) **[3 Points]** The MARK-LUCKY-BUILDINGS routine given on the other page marks all "Lucky" buildings. For each $i \in [1, n]$, it calls another function named FIND-CLOSEST-TALLER-BUILDING that sets $B[i].p$ to 0 provided no other building is obstructing building i's view of the statue, otherwise $B[i].p$ is set to the index of the building closest to i obstructing its view (i.e., $B[i].p$ points to the taller building closest to i in the direction of the statue). Figure 1 shows an example.

What is the worst-case running time of FIND-CLOSEST-TALLER-BUILDING?

1(b) [**15 Points**] Show that the amortized running time of FIND-CLOSEST-TALLER-BUILDING is $\mathcal{O}(1)$.

```
COUNT-TALLER-BUILDINGS( B[1 : n] )    {For i ∈ [1, n], height of building i is given in B[i].h, where building 1 is
                                       the closest to the statue and building n is the farthest from it on L.
                                       This algorithm returns the number of taller buildings an observer on the
                                       top of building n can see if he/she looks in the direction of the statue.}
1. j ← n,  k ← 0
2. for i ← n − 1 downto 1 do
3.     if B[i].h > B[j].h then
4.         j ← i,  k ← k + 1
5. return k
```

1(*c*) **[5 Points]** The COUNT-TALLER-BUILDINGS routine given above returns the number k of taller buildings an observer can see from the top of building n when looking in the direction of the statue. We want to compute the average value of k returned by this routine, when the average is taken over all $n!$ possible permutations of the building heights.

Let $f_{n,k}$ be the fraction of all possible permutations of building heights for which COUNT-TALLER-BUILDINGS returns k as the count. Then the average value C_n of k is given by the following expression.

$$C_n = \sum_k k f_{n,k}$$

Consider the following generating function for $f_{n,k}$'s.

$$F_n(z) = f_{n,0} + f_{n,1}z + f_{n,2}z^2 + \ldots + f_{n,n}z^n$$

Show that $C_n = F_n'(1)$.

1(d) [**5 Points**] One can show that for $n > 0$, $f_{n,k}$ can be described by the following recurrence.

$$nf_{n,k} = \begin{cases} 1 & \text{if } (n = 1 \ \wedge \ k = 0), \\ 0 & \text{if } (k < 0) \ \vee \ (n = 1 \ \wedge \ k \neq 0), \\ (n-1)f_{n-1,k} + f_{n-1,k-1} & \text{otherwise.} \end{cases}$$

Use this recurrence to show that $F_n(z) = \frac{z+n-1}{n} F_{n-1}(z)$.

1(e) [**7 Points**] Use results from parts (c) and (d) to show that $C_n \approx \ln n$.

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

Question 2. [**20 Points**] **Approximate Triangulations Counting and More.** This question is on recurrences.

2(*a*) [**10 Points**] The following recurrence[2] describes the running time of an approximation algorithm for counting triangulations.

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

Use the Akra-Bazzi method to show that $T(n) < 2^{\Theta\left(\sqrt{n}\log n\right)}$.

[2] with a different base case, but never mind

2(*b*) [**10 Points**] Consider the following recurrence for $n = 2^k$, where $k \geq 0$ is an integer.

$$S(n) = \begin{cases} \Theta(1) & \text{if } 1 \leq n \leq 4, \\ 3\sum_{i=2}^{\log_2 n} S\left(\frac{n}{2^i}\right) + 2n & \text{otherwise.} \end{cases}$$

Use the Akra-Bazzi method to show that $S(n) = \Theta(n^\alpha)$, where $\alpha = \log_2\left(\frac{\sqrt{13}-1}{6}\right) \approx 1.2$.

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

Question 3. [20 Points] Multiplying Matrices with Polynomial Entries. You are given two $n \times n$ matrices A and B, where $n > 0$. For $1 \leq i, j \leq n$, the cell at the intersection of row i and column j of matrix A contains a polynomial $A_{i,j}(x)$ of degree bound n. Similarly, the entry at cell (i, j) of matrix B is a polynomial $B_{i,j}(x)$ of degree bound n. Show that one can compute the product of A and B in $\mathcal{O}\left(n^{1+\mu}\right)$ time, where for some $\mu > 2$, $\mathcal{O}\left(n^{\mu}\right)$ is the complexity of computing the product of two standard $n \times n$ matrices[3].

[3] e.g., for Strassen's algorithm $\mu = \log_2 7 \approx 2.81$

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

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 $\left\lfloor \frac{n}{b} \right\rfloor$ 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 $n \times n$ 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 $\Theta\left(n \log n\right)$ time using the FFT (Fast Fourier Transform) algorithm (improving over the standard $\Theta\left(n^2\right)$ time algorithm).