In-Class Midterm

(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 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*. But *no books* and *no computers*.

Good Luck!

Question	Pages	Score	Maximum
1. Lining up LEGO bricks	2–4		30
2. LCS in Subquadratic Space	6–10		35
3. Searching in a Random BST	12		10
Total			75

Name: ______________________________

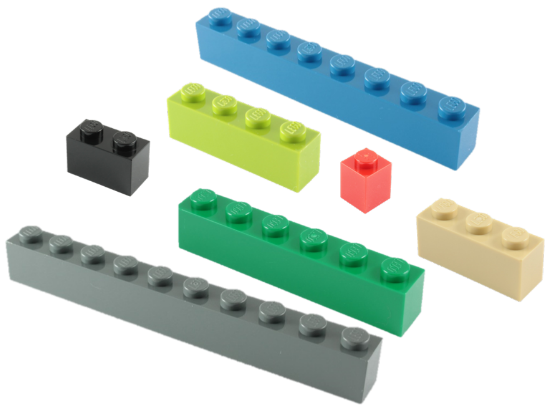

Figure 1: Few $1 \times r$ LEGO bricks, where r is a positive integer.

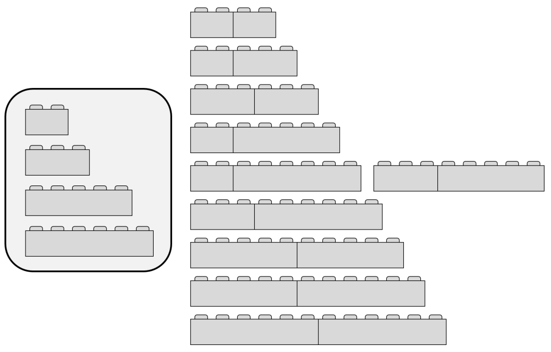

Figure 2: If you are allowed to line up exactly 2 LEGO bricks chosen from 1×2, 1×3, 1×5 and 1×6 bricks (as shown inside the box), then only rows of 9 distinct lengths (4–12) can be created (as shown on the right).

Question 1. [**30 Points**] **Lining up LEGO bricks.** A $1 \times r$ LEGO brick is one that has a row of r knobs on its top (see Figure 1 for examples), where r is a positive integer. We will say that the *length* of such a brick is r.

Suppose you have a bag of $1 \times r$ LEGO bricks. There are bricks of exactly n distinct lengths[1], and at least k bricks of each length, where both n and k are positive integers. Also assume that $r \in [1, cn]$ for some constant $c \geq 1$.

Now I ask you to take exactly k bricks from the bag, and line them up so that all knobs form a single row. If you chose bricks $1 \times r_1, 1 \times r_2, \ldots, 1 \times r_k$, the length of the row will be $\sum_{i=1}^{k} r_i$. Then put the bricks back to the bag, and take another set of k bricks (not necessarily completely disjoint from the pervious set) and line them up so that they form a row of a length different from the previous one. How many times can you do this so that everytime you get a completely new row length[2]? What are the distinct row lengths you get? Figure 2 shows an example.

1(a) [**5 Points**] Show that for any given k you can output all distinct row lengths in $\mathcal{O}\left(n^2k^2\right)$ time. For example, for the example in Figure 2 you will have to output: 4, 5, 6, 7, 8, 9, 10, 11, 12.

[1] i.e., r takes exactly n distinct values

[2] different from each of the previous lengths

1(b) [**15 Points**] Explain how you will output all distinct row lengths in $\mathcal{O}(n \log n)$ time when $k = 2$.

[*Hint: Construct a polynomial for the* $1 \times r$ *bricks of different lengths you have, i.e., coefficient of* x^r *will be 1 if you have a* $1 \times r$ *brick and 0 otherwise.*]

1(c) [**10 Points**] Explain how you will extend your algorithm from part 1(b) to output all distinct row lengths in $\mathcal{O}\left(nk(\log n + \log k)\right)$ time for any given k.

[*Hint: Use repeated squaring. For example,* x^{25} *can be computed using only 6 multiplications (instead of 24) as follows:* $x^{25} = \left(x^{12}\right)^2 \cdot x$*,* $x^{12} = \left(x^6\right)^2$*,* $x^6 = \left(x^3\right)^2$ *and* $x^3 = (x)^2 \cdot x$*.*]

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

```
FIND-LCS( $X$, $Y$, $k$ )  {Given two sequences $X = x_1x_2\ldots x_n$ and $Y = y_1y_2\ldots y_n$ this algorithm returns the longest
                            common subsequence (LCS) of $X$ and $Y$. Here, $k \in [2, n]$ is a parameter that determines
                            the space usage and running time of the algorithm. We assume $n$ to be a power of $k$.}

 1. let $Z[1..n]$ be an array for storing the LCS in reverse order
 2. $\mathcal{L} \leftarrow 0$                {global variable: length of the LCS segment found so far}
 3. let $c[0..n]$ be a new integer array
 4. for $j \leftarrow 0$ to $n$ do $c[j] \leftarrow 0$
 5. $\mathcal{H} \leftarrow n$   {global variable: the LCS segment found so far have considered all symbols of $Y$ to the right of $y_{\mathcal{H}}$}
 6. RECURSIVE-LCS( $X$, $Y$, $c$, $k$ )
 7. let $z_1z_2\ldots z_{\mathcal{L}}$ be the symbols stored in $Z[1..\mathcal{L}]$ in reverse order
 8. return $z_1z_2\ldots z_{\mathcal{L}}$
```

```
RECURSIVE-LCS( $X$, $Y$, $d$, $k$ ) {Given two sequences $X = x_1x_2\ldots x_m$ and $Y = y_1y_2\ldots y_n$ this algorithm returns the
                                     LCS of $X$ and $y_1y_2\ldots y_{\mathcal{H}}$. Here, $k \in [2, n]$ is a parameter that determines the
                                     space usage and running time of the algorithm. We assume $m$ to be a power of $k$.}

 1. $m \leftarrow X.length$, $n \leftarrow Y.length$, $r \leftarrow 0$, $t \leftarrow \frac{m}{k}$
 2. let $c[0..k-1, 0..n]$ be a new table
 3. for $j \leftarrow 0$ to $n$ do $c[r, j] \leftarrow d[j]$
 4. for $i \leftarrow 1$ to $m$ do
 5.     if $(i-1) \bmod t = 0$ then
 6.         $r \leftarrow r + 1$
 7.         for $j \leftarrow 0$ to $n$ do $c[r, j] \leftarrow c[r-1, j]$
 8.     $v \leftarrow c[r, 0]$
 9.     for $j \leftarrow 1$ to $n$ do
10.         if $x_i = y_j$ then $u \leftarrow c[r, j-1] + 1$
11.         else if $c[r, j] \geq c[r, j-1]$ then $u \leftarrow c[r, j]$
12.             else $u \leftarrow c[r, j-1]$
13.         $c[r, j-1] \leftarrow v$, $v \leftarrow u$
14.     $c[r, n] \leftarrow v$
15. if $m \leq k$ then
16.     $i \leftarrow m$
17.     while $i > 0$ and $\mathcal{H} > 0$ do
18.         if $x_i = y_{\mathcal{H}}$ then $\mathcal{L} \leftarrow \mathcal{L} + 1$, $Z[\mathcal{L}] \leftarrow x_i$, $i \leftarrow i - 1$, $\mathcal{H} \leftarrow \mathcal{H} - 1$
19.         else if $c[i-1, \mathcal{H}] \geq c[i, \mathcal{H}-1]$ then $i \leftarrow i - 1$
20.             else $\mathcal{H} \leftarrow \mathcal{H} - 1$
21. else
22.     for $r \leftarrow k-1$ downto $0$ do
23.         RECURSIVE-LCS( $X[rt+1..(r+1)t]$, $Y$, $c[r, 0..n]$, $k$ )
```

Figure 3: FIND-LCS(X, Y, k) finds an LCS of sequences X and Y using space parameterized by k.

Question 2. [**35 Points**] **LCS in Subquadratic Space.** A sequence $Z = \langle z_1, z_2, \ldots z_p \rangle$ is called a *subsequence* of another sequence $X = \langle x_1, x_2, \ldots x_m \rangle$ provided Z can be obtained from X by removing zero or more symbols from X. For example, $Z = \langle b, c, a \rangle$ is a subsequence of $X = \langle a, b, c, b, a \rangle$ obtained by removing the first 'a' and the second 'b' from X. A sequence Z is a *common subsequence* of sequences X and Y if Z is a subsequence of both X and Y. In the *Longest Common Subsequence* (LCS) problem we are given two sequences X and Y, and we need to find a maximum-length common subsequence of X and Y. For example, $Z = \langle b, c, b, a \rangle$ is an LCS of $X = \langle a, b, c, b, d, a, b \rangle$ and $Y = \langle b, d, c, a, b, a \rangle$.

The classic dynamic programming algorithm for finding an LCS of two sequences of lengths m and n runs in $\Theta(mn)$ time and uses $\Theta(mn)$ space. The algorithm in Figure 3, on the other hand, provides a tradeoff between time and space. The smaller the value of parameter k the smaller the space usage and the larger the running time, and vice versa. This task asks you to analyze the running time and space requirement of that algorithm.

The algorithm Find-LCS accepts two sequences X and Y of the same length n, and calls Recursive-LCS to find an LCS of X and Y through recursive divide-and-conquer.

2(a) [**8 Points**] Let $T(m, n)$ be the running time of Recursive-LCS when $X.length = m$ and $Y.length = n$. Argue that $T(m, n)$ can be described using the following recurrence relation assuming m to be a power of k.

$$T(m,n) = \begin{cases} \Theta(mn) & \text{if } m \le k, \\ kT\left(\frac{m}{k}, n\right) + \Theta(mn) & \text{otherwise.} \end{cases}$$

Solve the recurrence to obtain an asymptotic bound on $T(m, n)$.

[*Hint: You do not need to understand the detailed working of the algorithm in order to write down the recurrence relation for its running time. So please do not waste time trying to do that. All you need to do is to find bounds on the running times of various parts of the algorithm. For example, since each of lines 10–13 takes $\Theta(1)$ time, the **for** loop in lines 9–13 takes $\Theta(n)$ time. Also the **while** loop in lines 17–20 takes only $\mathcal{O}(m+n)$ time.*]

2(b) [**12 Points**] Let $S(m, n)$ be the space complexity of Recursive-LCS when $X.length = m$ and $Y.length = n$. Write down a recurrence relation describing $S(m, n)$ assuming m to be a power of k. Solve the recurrence.

[*Hint: Please keep in mind that when you return from a recursive call in line 23 all temporary space allocated inside that function are already freed, and so the freed space will be reallocated/reused by the next recursive call (if any).*]

2(c) [**5 Points**] Do your solutions from parts 2(a) and 2(b) hold when $k = n^{\epsilon}$ for some given constant $\epsilon \in (0,1]$ (more specifically, $k = \min\{\lceil n^{\epsilon} \rceil, n\}$, but we will assume $k = n^{\epsilon}$ for simplicity)? Why or why not?

2(d) [**10 Points**] What are the bounds on $T(m, n)$ and $S(m, n)$ for the value of k given in part 2(c)?

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

Question 3. [**10 Points**] **Searching in a Random BST.** Consider constructing a binary search tree (BST) from n numbers as follows. Pick a number uniformly at random from the n input numbers as a pivot, put the pivot in the root node, put all numbers smaller than the pivot in the left subtree and the rest in the right subtree. Apply the same partitioning step recursively on the two subtrees.

Now the expected time $T(n)$ needed to search for a number in such a tree can be described by the following recurrence relation.

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n < 30, \\ \frac{1}{3}T\left(\frac{2n}{3}\right) + \frac{1}{6}T\left(\frac{3n}{4}\right) + \frac{1}{10}T\left(\frac{4n}{5}\right) + \frac{1}{15}T\left(\frac{5n}{6}\right) + \frac{1}{3}T(n) + \Theta(1) & \text{otherwise.} \end{cases}$$

The recurrence above simply says the following. If at any point you are at the root of a (sub-)tree containing n nodes, you spend $\Theta(1)$ time comparing the item you are searching for with the item stored at that node, and if that node does not contain the item you are looking for then based on the outcome of the comparison you either move to the left subtree or the right subtree. But with probability at least $\frac{1}{3}$ the size of the subtree you will move to will be $\frac{2n}{3}$ or lower, otherwise[3] with probability at least $\frac{1}{6}$ the subtree size will be $\frac{3n}{4}$ or lower, and so on. Finally, with probability at most $\frac{1}{3}$ the subtree size is larger than $\frac{5n}{6}$, but smaller than n.

Solve the recurrence for finding an asymptotic upper bound on $T(n)$.

[*Hint: The main problem is $T(n)$ also appears on the right hand side of the recurrence. Recall how you usually simplify a equation that has the same variable on both sides of it, e.g.,* $3x = 5 + \frac{x}{2}$.]

[3] if the subtree size is greater than $\frac{2n}{3}$

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).