Note: The homework is due electronically on the course website on Thursday, February 04 by 10:30 am, the beginning of the class. For late submissions, please refer to the Late Submission Policy on the course webpage: http://www.seas.upenn.edu/~cis121

You must use the hw121.cls template provided on the course website.

Please write concise and clear solutions; you will get only a partial credit for correct solutions that are either unnecessarily long or not clear.

You are allowed to discuss ideas for solving homework problems in groups of up to 3 people but you must write your solutions independently. Also, you must write on your homework the names of the people with whom you discussed.

Finally, you are not allowed to use any material outside of the class notes and the textbook. Any violation of this policy may seriously affect your grade in the class.

Note the following.

1. \( \lg n \) means \( \log_2 n \).

2. You may find Steve Seiden’s Theoretical Computer Science Cheat Sheet (posted on the class page under “Resources”) useful.

[14] 1. (a) For each pair of expressions \((A, B)\) in the table below, indicate whether \(A\) is \(O, o, \Omega, \omega, \) or \(\Theta\) of \(B\). Assume that \(k \geq 1, \epsilon > 0,\) and \(c > 1\) are constants. Your answer should be in the form of the table with “yes” or “no” written in each box. No justification is required.

<table>
<thead>
<tr>
<th>(A)</th>
<th>(B)</th>
<th>(O)</th>
<th>(o)</th>
<th>(\Omega)</th>
<th>(\omega)</th>
<th>(\Theta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\lg^k n)</td>
<td>(n^\epsilon)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n^k)</td>
<td>(e^n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2^n)</td>
<td>(2^{n/2})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n^{\lg c})</td>
<td>(c^{\lg n})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\lg(n!)</td>
<td>(\lg(n^n))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) Rank the following functions by order of their growth; i.e., find an arrangement \(g_1, g_2, \ldots\) of the functions satisfying \(g_1 = \Omega(g_2), g_2 = \Omega(g_3), \ldots\) No justification is required.
(c) Let \( f(n) = 2^{n+1} \) and \( g(n) = 3^n/n \). Prove that \( f(n) = O(g(n)) \) by giving constants \( c \) and \( n_0 \).

[10] 2. Consider the following \texttt{IsPrime} algorithm for determining if the input positive integer is a prime.

\begin{verbatim}
IsPrime(n)
for (i = 2; i ≤ \( \sqrt{n} \); i = i + 1) do
    if n mod i = 0 then
        return \texttt{NO}
    else
        return \texttt{YES}

\end{verbatim}

a. Prove that the algorithm correctly determines if the input integer \( n > 1 \) is a prime or not.

b. Let \( T(n) \) be the worst-case running time of the algorithm in terms of the input integer \( n \). Determine \( T(n) \).

c. Let \( b \) denote the \textit{size} of the input to \texttt{IsPrime}, i.e., \( b \) is the number of bits needed to represent \( n \). Express the worst-case running time of the algorithm as a function of \( b \).

d. Is \texttt{IsPrime} a polynomial-time algorithm in the size of the input \( b \)?

[28] 3. For each of the code fragments given below, give a bound of the form \( \Theta(f(n)) \) on its running time on an input of size \( n \). Justify your answer.

\begin{verbatim}
(a) for (i = 1; i ≤ n; i = i * 2) do
    for j = i to 8 * i + 1 do
        print(‘CIS 121 is fun!’)
\end{verbatim}
(b) \begin{align*}
\text{for } & (i = 2; \ i \leq n; \ i = i^2) \ \text{do} \\
\text{for } & (j = \frac{n}{1024}; \ j \leq n^2; \ j = j \times 2) \ \text{do} \\
\text{print } & \left( \text{“hi”} \right)
\end{align*}

(c) \begin{align*}
\text{for } & i = 1 \ \text{to } n \ \text{do} \\
\text{for } & (j = 1; \ j \leq i; \ j = j \times 2) \ \text{do} \\
\text{print } & \left( \text{“hi”} \right)
\end{align*}

(d) \begin{align*}
\text{sum} = 0 \\
\text{for } & (i = 1; \ i < n; \ i = i \times 2) \ \text{do} \\
\text{for } & (j = i; \ j < n; \ j++) \ \text{do} \\
\text{sum} = & \text{sum} + 1
\end{align*}

[12] 4. Consider an array $A$ containing $n$ binary numbers 0 or 1. Furthermore, $A$ is such that one of the two numbers occurs at least 95% of the times in $A$. Given such an array $A$, our objective is to determine if $A$ contains mostly 0s or mostly 1s.

a. Give a deterministic algorithm for the problem that examines only a small fraction of elements in $A$ (strictly less than $n$). Prove the correctness of your algorithm. What is its running time? Use the $\Theta$ notation to express your answer and provide a justification.

b. Consider the following randomized algorithm for the problem, in which the function `random(a, b)` returns a random integer between $a$ and $b$, with all numbers equally likely. What is the running time of the algorithm? You may assume that the function `random` takes $O(1)$ time.

\begin{align*}
& i \leftarrow \text{random}(1, n) \\
& j \leftarrow \text{random}(1, n) \\
& k \leftarrow \text{random}(1, n) \\
& \text{if } A[i] + A[j] + A[k] \leq 1 \ \text{then} \\
& \quad \text{return } \left( \text{“mostly 0s”} \right) \\
& \text{else} \\
& \quad \text{return } \left( \text{“mostly 1s”} \right)
\end{align*}

c. The algorithm can output an incorrect answer. Explain how this can happen.

d. Suppose $A$ contains mostly 0s. Give a non-trivial upper-bound on the probability of the bad event, i.e., give an upper-bound on the probability that the randomized algorithm will output “mostly 1s.” Justify your answer.
5. To check whether an input has a certain property $P$, suppose there is a randomized algorithm $A$ that works as follows:

- if $x$ has the property $P$, $A(x)$ always outputs YES.
- if $x$ does not have property $P$, then $A(x)$ outputs NO with a probability $2/3$, otherwise it says YES.

Your goal is to find out if a certain input $x$ has the property $P$ or not. However, you want the probability of getting the wrong answer to be at most $1/10^{12}$. Design an algorithm to do this? Justify your answer.

6. Consider the following basic problem. You’re given an array $A$ consisting of $n$ integers $A[1], A[2], \ldots, A[n]$. You’d like to output a two-dimensional $n$-by-$n$ array $B$ in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$ – that is, the sum $A[i] + A[i+1] + \cdots + A[j]$. (The value of the array entry $B[i, j]$ is left unspecified whenever $i \geq j$, so it doesn’t matter what is the output of these values.)

Here is a simple algorithm to solve this problem.

```plaintext
for (i=1; i <= n; i++)
    for (j=i+1; j <= n; j++)
        Add up array entries A[i] through A[j]
        Store the result in B[i,j]
```

a. For the above code fragment give a bound of the form $O(f(n))$ on its running time on an input of size $n$. Justify your answer.

b. For this same function $f$, show that the running time of the algorithm on an input of size $n$ is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

c. Although the algorithm you analyzed in parts (a) and (b) is the most natural way to solve the problem – after all, it just iterates through the relevant entries of the array $B$, filling in a value for each – it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n \to \infty} g(n)/f(n) = 0$.

7. The Pigeonhole Principle Players of Philadelphia needs some help with bus logistics. They are planning to have a huge closing out finale for their world tour. They are providing buses to bring people from towns in the area to the show. The other people on the logistics team are planning on sending one bus to every town, but as a studious student, you know you can save the PPPP money on buses by reducing how many buses they use.

You are provided with a list of $n$ towns and the corresponding number of fans coming from each town. This list is sorted in non-decreasing order of the number of fans at each
of the towns. You know the capacity of the bus is $C$. Give an algorithm to find two towns, if they exist, that the bus can visit such that the bus after picking up all fans from the two towns will be at exactly full capacity, or state if it is not possible to do so. That is, if there exists a possible combination of two towns such that the number of fans from these two towns is exactly equal to $C$, your algorithm will output the towns. Otherwise, it will output that no such pair of towns exist. Your algorithm must run in $O(n)$ time. Justify the running time of your algorithm and prove its correctness.