0. **(1 pt.)** Have you thoroughly read the course policies on the webpage?

[**We are expecting:** The answer “yes.”]

1. **(1 pt.)** Which of the following functions are \( O(n^2) \)?

No explanation is required, but you might want to prove your answer to yourself to convince yourself that you are correct.

(a) \( f_1(n) = 5n + 3 \log n \)
(b) \( f_2(n) = 5n^2 + 3n \)
(c) \( f_3(n) = \frac{1}{100}n^3 + 3n^2 \)
(d) \( f_4(n) = n \log(n) \)
(e) \( f_5(n) = \sin(n) + 5 \)
(f) \( f_6(n) = 2^n \)
(g) \( f_7(n) = 2^{100} \)

[**We are expecting:** A list of which functions are \( O(n^2) \). No explanation is required and no partial credit will be given.]

2. **(1 pt.)** Which of the following functions are \( \Omega(n^2) \)?

No explanation is required, but you might want to prove your answer to yourself to convince yourself that you are correct.

(a) \( f_1(n) = 5n + 3 \log n \)
(b) \( f_2(n) = 5n^2 + 3n \)
(c) \( f_3(n) = \frac{1}{100}n^3 + n^2 \)
(d) \( f_4(n) = n^2 \log(n) \)
(e) \( f_5(n) = n^2(\sin(n) + 5) \)
(f) \( f_6(n) = n^2\left(\frac{1}{\log n}\right) \)
(g) \( f_7(n) = n! \)

[**We are expecting:** A list of which functions are \( \Omega(n^2) \). No explanation is required and no partial credit will be given.]

More exercises on next page...
3. (4 pt.) In this problem, we’ll utilize two different applications of the definition of big-O to prove that $f(n) = O(g(n))$, where $f(n) = 4n$, $g(n) = 2n^2 - 2n$.

(a) In the definition of big-O, suppose we have to use $c = 1$. Give a proof that $f(n) = O(g(n))$ in which “$c$” is chosen to be 1.
(b) Now instead, suppose we don’t care what $c$ is, but would like to take $n_0 = 2$. Give a different proof that $f(n) = O(g(n))$ in which “$n_0$” is chosen to be 2.

[We are expecting: For both parts, a short but formal proof.]

4. (2 pt.) Prove that $8^n$ is not $O(2^n)$.

[We are expecting: A formal proof.]

5. (3 pt.) Consider the following algorithm (pseudocode shown), which recursively computes the maximum of an array (this is by no means the most elegant algorithm for computing the maximum):

```plaintext
RecursiveMaximum( array A of length n ):
    if n == 1:
        return A[0]
    max_1 = RecursiveMaximum( A[:n/3] )
    max_2 = RecursiveMaximum( A[n/3 : 2n/3] )
    return max(max_1, max_2, max_3)
```

(a) Complete the following Recursion Tree table for `RecursiveMaximum`.

For full credit, and to help you with part (b), avoid using Big-O notation in any entries of the table. For example, if you wanted to put down $O(n^2)$, put down $c \cdot n^2$ instead.

Some tips for formatting the table in LaTeX are provided at the end of this assignment.

<table>
<thead>
<tr>
<th>Level</th>
<th># of problems</th>
<th>Problem size</th>
<th>Work per problem</th>
<th>Total Work in this Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>$n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

[We are expecting: The completed Recursion Tree table for `RecursiveMaximum`. If you aren’t using the Homework 1 Template, please label your entries clearly.]

(b) Using your completed recursion tree table, compute the overall Big-O runtime of `RecursiveMaximum`.

Show your work by summing up all the values in the last column (“Total Work in this Level”).

[HINT: When simplifying your summation, the geometric series formula may be helpful.]

[We are expecting: The overall Big-O runtime of `RecursiveMaximum`, and your work that shows how you used your recursion tree table to arrive at your answer.]

(c) Now, use the Master Theorem to compute the overall Big-O runtime of `RecursiveMaximum` on an array of length $n$. Start by writing the recurrence relation, and briefly show your work.

[We are expecting: The recurrence relation for `RecursiveMaximum`, and your work that shows how you applied the Master Theorem to compute the overall Big-O runtime.]

More problems on next page.
Problems

You may talk with your fellow CS161-ers about the problems. However:

- Try the problems on your own before collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

6. **[Fibonacci] (6 pt.)**

The Fibonacci sequence $F_0, F_1, \ldots$, is defined by $F_0 = 0, F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. For example, the first several Fibonacci numbers are 0, 1, 1, 2, 3, 5, 8, \ldots. Show by induction that

(a) $F_n = O(2^n)$

(b) $F_{2n} = \Omega(2^n)$.

Together, parts (a) and (b) imply that $F_n = 2^{\Theta(n)}$.

**HINT:** We want you to combine what we’ve learned about induction proofs with what we’ve learned about Big-O proofs. It will help to structure each of your overall proofs as an induction proof within a Big-O proof. Begin by carefully setting up your Big-O proof, using the format discussed in lecture. Then, after you’ve declared what you’re setting out to prove, prove it (using induction)!

**We are expecting:** For each part, a formal proof that involves induction. Make sure to clearly label your inductive hypothesis, base case, inductive step, and conclusion.

More problems on next page...
7. **[Skyline] (13 pt.)** You are handed a scenic black-and-white photo of the skyline of a city. The photo is \( n \)-pixels tall and \( m \)-pixels wide, and in the photo, buildings appear as black (pixel value 0) and sky background appears as white (pixel value 1). In any column, all the black pixels are below all the white pixels. In this problem, you will design an efficient algorithm that finds the location of a tallest building in the photo.\(^1\)

The input is an \( n \times m \) matrix, where the buildings are represented with 0s, and the sky is represented by 1s. The output is an integer representing the location of a tallest building. For example, for the input 6 \( \times \) 5 matrix below, a tallest building has height 5 and is in location 1 (assuming we are 0-indexing). Thus the output is 1.

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

(a) **(5 pt.)** Find an algorithm that finds a tallest building in time \( O(m \log n) \).

**[We are expecting: Pseudocode, a clear English description of what your algorithm is doing, and a brief justification of the runtime. No proof of correctness is required.]**

(b) **(5 pt.)** Find an algorithm that finds a tallest building in time \( O(m + n) \).

**[We are expecting: Pseudocode, a clear English description of what your algorithm is doing, and a brief justification of the runtime. No proof of correctness is required.]**

(c) **(3 pt.)** For some values of \( (n, m) \) the algorithm from part (a) is more efficient, while for others, the algorithm from part (b) is more efficient. For each of the values of \( n \) in terms of \( m \) below, determine which of the above algorithms is more efficient (or that they are equally efficient) in terms of big-O notation. The case \( n = m \) is filled in as an example.

If it is helpful, some tips for formatting the table in \LaTeX are provided at the end of this assignment.

<table>
<thead>
<tr>
<th>( n = ? )</th>
<th>100</th>
<th>( \sqrt{m} )</th>
<th>( m/\log m )</th>
<th>( m \log m )</th>
<th>( m^2 )</th>
<th>( 2^m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime for (a)</td>
<td>( O(m \log m) )</td>
<td>( O(m) )</td>
<td>( \text{(b)} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runtime for (b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Which is better?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**[We are expecting: For each value of \( n \) in terms of \( m \), the best big-O runtime you can guarantee for each of your algorithms from (a) and (b), and a conclusion about which is asymptotically more efficient. You do not need to give any formal proofs, but your runtimes should be in the simplest terms possible.]**

More “problems” on next page...

\(^1\)It could be that there are multiple tallest buildings that all have the same height; in this case, your algorithm should return any one of them.
Feedback

There’s no “correct” answer here, and it is completely anonymous.

8. (1 pt.) Please fill out the following poll, which asks about your expectations for the course:

https://forms.gle/Nai4Kd8RnyRtutXz9

Did you fill out the poll?

[We are expecting: The answer “yes.”]

\LaTeX tips for formatting tables (for problem 5a and 7c)

Note: if after filling in the table provided in the Homework 1 Template, it is too wide for the page, you can either use a command like `\footnotesize` before you start the table and `\normalsize` afterwards to put it in a smaller font; or you can edit the code provided to break the table up across multiple lines.