Please read the following guidelines carefully!

- Please write your name on the front and back of the exam. The latter is to help us return the exams.

- This examination has two questions. There are a total of 7 DOUBLE-SIDED pages, not including this title page.

- Answer questions clearly and completely. Give formal proofs unless explicitly asked not to. You may use any claim/result from class, unless you are being asked to prove that claim/result, or explicitly told not to.

- Any question you leave blank or clearly cross out your work and write “I don’t know” is worth 10% of the marks.

- Please hand in your aid sheet along with the exam.

Take a deep breath.

This is your chance to show us

How much you’ve learned.

We WANT to give you the credit

That you’ve earned.

A number does not define you.

Good luck!
1. In this question, you will devise a divide-and-conquer algorithm that checks whether an input string $s$ contains the substring '000'. Your algorithm must divide the input string into two halves. You may assume in input string has a length that is a power of $2$: $1, 2, 4, 8, \ldots$

You should treat strings like lists, and freely use notation like $s[0]$ and $s[1..\text{len}(s)-1]$.

(a) [5] Write pseudocode for your algorithm, and include some brief justification on why it is correct (no formal proof necessary).
(b) [3] Give a recurrence for the (worst-case) runtime of your algorithm (base case not necessary). Don’t forget to define $T(n)$, and be sure to justify your answer by referring to your code.

(c) [1] What is the asymptotic runtime of your algorithm? Justify your answer.

(d) [3] Consider another algorithm to solve this problem that checks if $s[0..2] == \text{'000'}$ (in constant time), and if it doesn’t, recursively checks the substring $s[1..\text{len}(s)-1]$. Write and justify a recurrence for the (worst-case) runtime of this algorithm, and then explain why the Master Theorem cannot be used to find an asymptotic bound for your recurrence.
   You do NOT need to find an asymptotic bound yourself.
2. Read the following program’s specifications and code carefully. Notice both the loop and the recursive call.

```python
1 def count_ordered(A):
2     '''
3     Pre: A is a list of numbers
4     Post: Outputs the number of pairs (i, j) such that i < j and A[i] <= A[j]
5     E.g., count_ordered([3, 2, 5, 1]) = 2 and count_ordered([10, 10]) = 1
6     '''
7     if len(A) == 0:
8         return 0
9     else:
10        count = 0
11        i = 1
12        while i < len(A):
13            if A[i] >= A[0]:
14                count += 1
15                i += 1
16        return count + count_ordered(A[1..len(A)-1])
```

(a) [5] State and prove a helpful loop invariant for the while loop (“steps 1 and 2” from lecture). You may assume \( i \in \mathbb{N} \) and \( i \leq \text{len}(A) \) have already been proven as loop invariant (so use them freely).
(b) [5] Using your loop invariant from part (a), prove the correctness of \texttt{count\_ordered} according to the specifications. Once again, note that \texttt{count\_ordered} is recursive, so you need to analyse a recursive call.
Use this page for rough work.
Use this page for rough work.
Name:

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grade</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Out Of</td>
<td>12</td>
<td>10</td>
<td>22</td>
</tr>
</tbody>
</table>