1. Comparing functions
   (a) (10 points) Order the following functions according to their order of growth from the lowest to the highest. If you think that two functions are of the same order (i.e., \( f(n) \in \Theta(g(n)) \)), put them in the same group.
   
   \[ 5n^2 + n \log^2 n, n^{2.5}, 3n + n \log n, 3^n, 5n^2 \log n, \log_{10}(n!), 2^{2n}, 2n^2 - 3n + 5. \]
   
   \[ \Theta(n^2) \quad \Theta(n \log n) \quad \Theta(n \log n) \quad \Theta(4^n) \quad \Theta(n^2) \]
   
   \[ 3^n \ll 4^n \]

   (b) (10 points) For each pair of functions in the table below, determine whether \( f(n) \in O(g(n)) \), \( f(n) \in \Omega(g(n)) \), \( f(n) \in \Theta(g(n)) \), or all of them. It is NOT necessary to justify your answer.

<table>
<thead>
<tr>
<th>( f(n) )</th>
<th>( O )</th>
<th>( \Omega )</th>
<th>( \Theta )</th>
<th>( g(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) ( (n^2 + 5n) \log n )</td>
<td>X</td>
<td></td>
<td></td>
<td>( n^2 + \log n )</td>
</tr>
<tr>
<td>2) ( n^2 \log n )</td>
<td>X</td>
<td></td>
<td></td>
<td>( 5n^3 + n \log n )</td>
</tr>
<tr>
<td>3) ( n \log n )</td>
<td>X</td>
<td></td>
<td></td>
<td>( n + \sqrt{n} )</td>
</tr>
<tr>
<td>4) ( n^2 + 10 \log(n!) )</td>
<td>X</td>
<td>X</td>
<td></td>
<td>( 5n^2 + \sqrt{n} )</td>
</tr>
<tr>
<td>5) ( \log_2(n^2) )</td>
<td>X</td>
<td></td>
<td></td>
<td>( n \log_2 n )</td>
</tr>
</tbody>
</table>

\[ 2 \log_2 n \in O(\log n) \]

2. (10 points) Using the definition of \( O \), prove that \( \sum_{i=1}^{n} i \in O(n^2) \).

   \[
   \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}
   \]

   \[
   \leq \frac{n^2}{2} + \frac{n}{2} \quad \text{for } n > 1
   \]

   \[
   \leq n^2
   \]

   \[
   \therefore \text{if we choose } c = 1 \text{ and } n_0 = 1, \text{ we have}
   \]

   \[
   \sum_{i=1}^{n} i \leq c n^2 \quad \text{for all } n > n_0.
   \]

   \[
   \text{So by def, } \sum_{i=1}^{n} i \in O(n^2)
   \]
3. (10 points) You are given the following iterative algorithm:

```
1   Mystery(A[1..n])
2       //Input: An array A[1..n] of n real numbers
3       for (i = 1; i <= n-1; i++) {
4           for (j = i+1; j <= n; j++) {
5               if (A[i] == A[j])
6                   return false;
7           }
8       }
9   return true;
```

a. What does this algorithm compute?

checks if array A contains only unique elements.

b. What is the worst-case time complexity of the algorithm, as a function of $n$? Give an input that can lead to the worst-case running time.

$$
\sum_{i=1}^{n-1} (n-i) = \sum_{i=1}^{n-1} i = \Theta(n^2) \\
$$

when all elements all different

c. What is the best-case time complexity of the algorithm, as a function of $n$? Give an input that can lead to the best-case running time.

$\Theta(1)$, when the first two elements are equal.
4. (25 points) Assume that $T(1) \in \Theta(1)$. Solve the following recurrence functions using the **master method**. If the master method cannot be applied, state the reason. Justify your answer.

a. $T(n) = 3T(n/3) + n$;

   **Case 2.** $O(n \log n)$

b. $T(n) = 3T(n/2) + n^2$;

   $n^2 \text{ vs } n^{\log_2 3}$

   **Case 3.** $O(n^2)$

   **Reg cond.** If we choose $C = \frac{3}{4}$.
c. \( T(n) = 4T(n/2) + (n + 3 \log n); \)

\[ n^2 \ vs \ n + 3 \log n \]

Case 1.

\( \Theta(n^2) \)

d. \( T(n) = 4T(n/2) + n^2 \log n; \)

\[ n^2 \ vs \ n^2 \log n \]

Standard MT does not apply.

Ext. case 2. gives \( \Theta(n^2 \log n) \)

e. \( T(n) = 2T(n/2) + \log n; \)

\[ n \ vs \ \log n \]

Case 1. \( \Theta(n) \)
5. (10 points) Recursion tree method.
Assume that $T(1) \in \Theta(1)$. Solve the following recurrence function using the recursion tree method to get an asymptotically tight bound.

$$T(n) = 3T(n-2) + 1$$

\[
\sum_{i=0}^{h} 3^i = \Theta(3^h) = \Theta(3^{\frac{n}{2}}) = \Theta(n^{\log_3 3})
\]

$h = \frac{n}{2}$

$\Theta(3^n) \times$
6. (10 points) Substitution method.

Assume that \( T(1) \in \Theta(1) \) and \( T(n) = 2T(n/2) + 3T(n/3) + n^2 \). Prove \( T(n) \in O(n^2) \) using the substitution method.

\[
T(n) \leq C \cdot n^2 \quad \text{for some } c > 0 \text{ and all } n > n_0.
\]

Assume \( T(n/2) \leq C \cdot \left( \frac{n}{2} \right)^2 \)
and \( T(n/3) \leq C \cdot \left( \frac{n}{3} \right)^2 \),

\[
\begin{align*}
T(n) &= 2T(n/2) + 3T(n/3) + n^2 \\
&\leq 2 \cdot C \cdot \left( \frac{n}{2} \right)^2 + 3 \cdot C \cdot \left( \frac{n}{3} \right)^2 + n^2 \\
&\leq \frac{c}{2} n^2 + \frac{c}{3} n^2 + n^2 \\
&\leq (\frac{5c}{6} + 1) n^2 \\
&\leq C n^2 \quad \text{if } \frac{5c}{6} + 1 \leq C
\end{align*}
\]

\[
\frac{5c}{6} + 1 \leq C \\
\implies 1 \leq C - \frac{5c}{6} \\
\implies 1 \leq \frac{c}{6} \\
\implies 6 \leq c
\]

\[
\therefore T(n) \leq C n^2 \text{ for } c > 6 \text{ and all } n.
\]

by def., \( T(n) \in O(n^2) \).
7. (15 points) Analysis of recursive algorithms.

Consider the pseudocode of the following two algorithms. In both algorithms, the input \( A \) is an array of size \( n \), which is then split into 3 subarrays in subsequent recursive calls.

<table>
<thead>
<tr>
<th>AlgX ( (A[1..n]) )</th>
<th>AlgY ( (A[1..n]) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>if ( (n &lt; 3) ) return sum( (A[1..n]) );</td>
<td>if ( (n &lt; 3) ) return sum( (A[1..n]) );</td>
</tr>
<tr>
<td>( p = \text{floor}(n/3); )</td>
<td>( p = \text{floor}(n/3); )</td>
</tr>
<tr>
<td>// ( r ) is a random integer between</td>
<td>// ( r ) is a random integer between</td>
</tr>
<tr>
<td>// 1 and 3</td>
<td>1 and 3</td>
</tr>
<tr>
<td>( r = \text{ceil}(3 * \text{rand}()); )</td>
<td>( r = \text{ceil}(3 * \text{rand}()); )</td>
</tr>
<tr>
<td>if ( (r == 1) )</td>
<td>if ( (r == 1) )</td>
</tr>
<tr>
<td>return AlgX ( (A[1..p]) );</td>
<td>return AlgY ( (A[1..p]) );</td>
</tr>
<tr>
<td>else if ( (r == 2) )</td>
<td>else if ( (r == 2) )</td>
</tr>
<tr>
<td>return AlgX ( (A[p+1..2p]) );</td>
<td>return AlgY ( (A[p+1..2p]) );</td>
</tr>
<tr>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td>return AlgX ( (A[2p+1..n]) );</td>
<td>return AlgY ( (A[2p+1..n]) );</td>
</tr>
<tr>
<td>end</td>
<td>end</td>
</tr>
<tr>
<td>end</td>
<td>end</td>
</tr>
</tbody>
</table>

a. Write down the recurrence function for the running time of AlgX and solve it. If your analysis relies on any non-trivial assumption, state it.

\[
T(n) = T(n/3) + O(1) \quad \text{Assume passing array \& function calls take constant time.}
\]

\[
T(n) \in \Theta(\log n)
\]

b. Write down the recurrence function for the running time of AlgY and solve it. If your analysis relies on any non-trivial assumption, state it.

\[
T(n) = 3T(n/3) + O(1)
\]

\[
T(n) \in \Theta(n)
\]

c. Which algorithm is more efficient?

AlgX
8. (Extra credit - 20 points) K-way merge sort

In the classic merge sort algorithm, an input array of size \( n \) is split into two subarrays, which are recursively sorted and then combined with a linear time \textit{merge} function. Now consider a generalized idea that splits the input array into \( k \) equal-sized subarrays, where \( k \geq 2 \). As in the classic merge sort, each subarray is sorted recursively, and the sorted subarrays are merged to produce the final sorted array. See pseudocode below.

```plaintext
KwayMergeSort(A[1..n]);
    if (n <= 1) return A;
    Split A into k approximately equal-sized subarrays A1, A2, ... Ak.
    B = KwayMergeSort(A1);
    for (i = 2; i <= k; i++)
        merge(B, KwayMergeSort(Ai));
    end
    return B;
end
```

(a) Define the running time of the algorithm as a function of both \( n \) and \( k \). (Hint: pay closer attention to the total time needed to merge the sorted subarrays.)

(b) Solve the recurrence using the recursion tree method. For the tree height, keep the base of the logarithm function explicit.

(c) How does the running time change as \( k \) increases? In particular, in the most extreme case, when \( k = n \), what is the running time of the algorithm? Does it make intuitive sense? Explain.

\[
T(n) = \frac{k^2}{2} T\left(\frac{n}{k}\right) + \Theta(nk)
\]

\[
\begin{align*}
\sum_{i=2}^{k} \left( \frac{n}{k} \right) + \frac{n}{k} & = \sum_{i=2}^{k} \left( \frac{1}{k} \right) = \frac{1}{k} \sum_{i=2}^{k} i \\
& = \frac{\frac{1}{2} k^2 + \frac{1}{2} k}{2} \cdot n = \Theta(k) \cdot n
\end{align*}
\]

\[
\therefore T(n) = k \cdot T\left(\frac{n}{k}\right) + \Theta(nk)
\]
b. \[ \frac{n^k}{k \cdot \log^k n} \quad \frac{n^k}{k \cdot \log^k n} \quad \frac{n^k}{k \cdot \log^k n} \quad \frac{n^k}{k \cdot \log^k n} \quad \frac{n^k}{k \cdot \log^k n} \]

\[ h = \log \frac{n}{k} \]

\[ T(n) = O(n \cdot k \cdot \log k n) = O(n \cdot \log n \cdot \frac{k}{\log k}) \]

C. \[ \frac{k}{\log k} \] increase as \( k \) increases. Therefore \( T(n) \to \infty \) as \( k \to \infty \).

When \( k = n \), \( n \cdot \log n \cdot \frac{k}{\log k} = n^2 \).

The alg. essentially becomes an insertion sort.