1. **Java (3):**
   Write a java method that sorts an array using any method. List input, output, preconditions, and post conditions.

   Many solutions are possible

   ```java
   import java.util.Arrays;
   public static void javaSort(array A)
   {
       Arrays.sort(A);
   }
   
   PreCondition: Array A exists and is an array
   PostCondition: Array A is sorted
   Input: An array A
   Output: A sorted array A
   ```

2. **Divide and Conquer (10):**
   Write an algorithm which sorts a set of n numbers using at most $\log(n!) + n$ number of comparisons.

   (You may use any operation that does not compare a pair of elements as many times as you wish)

   Build a new list from the old one, and use binary search to place elements in the new list. This would require a large amount operations pertaining to maintaining the array, but those are not comparisons, so we need not worry about them.

3. **Induction (6):**
   Given a set of $n>2$ distinct points on a 2 dimensional plane, show that it is always possible to draw a polygon with $n$ sides containing all points as vertices, such that no two sides intersect.

   *There is a small mistake in this question – an additional condition stating that all points cannot lie on a single line is required. Thus, give yourself 6 out of 6 for this one.*

   The following is a sketch of the proof for those that are interested:

   Base case: Triangles are possible

   Induction hypothesis: Polygons are possible with $n-1$ points

   Induction step: Draw a polygon with $n-1$ points. For the last point, break an edge off of the polygon and tack the point there.

4. **Landau Symbols (Big O notation) (12):**
   The notation $o$ (read: small o) can be interpreted to mean the following:

   If $f(n) = o(g(n))$, 
then \( f(n) = O(g(n)) \) but \( g(n) \neq O(f(n)) \)

a) Find a function \( f(n) \) such that:
\[ f(n) = o(n) \]
\[ \log(n) = o(f(n)) \]

\[ n^c, \text{for any} \ c < 1 \text{ or } \log(n)^c, \text{for any} \ c > 1 \]

b) Find a function \( f(n) \) such that:
\[ f(n) = o(n^c), \text{for any} \ c > 0 \]
\[ (\log(n))^k = o(f(n)), \text{for any} \ k > 0 \]
\[ e^{\sqrt{\ln(n)}} \]

c) Find a function \( f(n) \) that uses only the binary operations +,-,\(^\cdot\),\(^*\),\(^/\),\(\log\), such that \( f(n) = O(n^n!) \) and \( n!/n = O(f(n)) \).

\( (Recall \ that \int \ln(x)dx = x\ln(x) + x) \)

\[ n^n/e^n \]

5. **Quicksort (8):**

There exists an algorithm which can find the \( k_{th} \) element in a list in \( O(n) \) time, and suppose that it is in place. Using this algorithm, write an in place sorting algorithm that runs in worst case \( O(n\log(n)) \), and prove that it does. Given that this algorithm exists, why is mergesort still used?

Use findKthElement\((n/2)\) on each iteration of Quicksort to find the pivot, and you will discover that the running time is \( O(n\log(n)) \). Though the big \( O \) is the same as mergesort, the constant will be much larger.

6. **ADTs (1):**

Complete the following table with optimal \( \text{big O} \) running times, given the data structure:

<table>
<thead>
<tr>
<th>Get ( i_{th} ) entry in list, where ( i ) is any number between 1 and ( n )</th>
<th>Array of size ( n )</th>
<th>Linked list of size ( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O(1) )</td>
<td>( O(n) )</td>
<td></td>
</tr>
<tr>
<td>Concatenate two lists</td>
<td>( O(n) )</td>
<td>( O(1) )</td>
</tr>
</tbody>
</table>