1. Solve the following recurrences using the Master Theorem:
   (a) \( T(n) = 3T(n/2) + O(n^2) \)
   (b) \( T(n) = 4T(n/2) + O(n^2) \)
   (c) \( T(n) = 4T(n/2) + O(n^{1.6}) \)
   (d) \( T(n) = 9T(n/3) + O(n^2) \)
   (e) \( T(n) = 10T(n/3) + O(n^2) \)

2. Solve the following recurrences using recursion trees. Assume \( n \) is a power of 2.
   (a) \( T(n) = 2T(n/4) + T(n/8) + O(n) \)
   (b) \( T(n) = 2T(n/4) + T(n/2) + O(n) \)
   (c) \( T(n) = 2T(n/2) + O(n \log n) \)
   (d) \( T(d) = 2T(n/2) + O((\log n)^2) \)

3. Consider the following recursive algorithm \( A(n) \):
   
   \[
   A(n):
   \]
   
   if \( n > 0 \) then
   
   \[
   A(n - 1)
   \]
   
   print “Hello”
   
   \[
   A(n - 1)
   \]

   How many times does the word “Hello” get printed when \( A(n) \) is called? Express your answer as an exact function of \( n \).

4. Consider the MergeSort algorithm on inputs of size \( n = 2^k \). Normally, this algorithm would have a recursion depth of \( k \). Suppose that we modify the algorithm so that after \( k/2 \) levels of recursion, it switches over to insertion sort. What is the running time of this modified algorithm?

5. You have \( n \) coins — they all look identical, and all have the same weight except one, which is heavier than all the rest. You also have a balance scale, on which you can place one set of coins on one side, and another set of coins on the other, and the scale will tell you whether the two sets have the same weight, and if not, which is the heavier set.

   (a) Assuming that \( n = 3^k \), devise a strategy that will identify the heavy coin using at most \( k \) weighings in the worst case.

   (b) Without any assumption on \( n \), devise a strategy that will identify the heavy coin using \( \log_3 n + O(1) \) weighings.

6. Suppose we can multiply two \( n \)-bit numbers in time \( O(n^\alpha) \), where \( \alpha \) is a constant with \( 1 < \alpha < 2 \). Using this multiplication algorithm as a subroutine, give an algorithm for the following problem. The input is a list of \( r \) numbers \( a_1, \ldots, a_r \), where each \( a_i \) is an \( \ell \)-bit number. The output is the product \( A = \prod_{i=1}^r a_i \). Note that \( A \) itself is an \((r\ell)\)-bit number. Your algorithm should run in time \( O((r\ell)^\alpha) \). Use a divide-and-conquer strategy, and analyze your algorithm using recursion trees.

The following is for honors students only.

H1. Using the multiplication algorithm in the previous exercise as a subroutine, give an algorithm for the following problem. The input consists of two lists of \( r \) numbers \( a_1, \ldots, a_r \), and \( b_1, \ldots, b_r \), where each \( a_i \) and each \( b_i \) is an \( \ell \)-bit number. The output is the sum \( B = \sum_{i=1}^r b_i A/a_i \), where \( A = \prod_{i=1}^r a_i \). Your algorithm should run in time \( O((r\ell)^\alpha) \). Use a divide-and-conquer strategy, and analyze your algorithm using recursion trees.