1. **Rates of growth.** Sort the following functions in order of their rate of growth. That is, sort them into a list $f_1, f_2, \ldots, f_n$, so that for $i = 1, \ldots, n - 1$, we have $f_i = O(f_{i+1})$. Also, for each adjacent pair of functions $f_i, f_{i+1}$ in your sorted list, indicate whether or not $f_i = o(f_{i+1})$.

- $n^2$, $n^n$, $\ln(n)$, $n \ln(n)$, $n \ln(\ln(n))$, $20n^2 + 117n - 6$,
- $n^2 \ln(n)$, $\log_2 n$, $(\log_2(n))^{100}$, $\log_2(n^{100})$, $n$, $2^n$, $n(\ln(n) + (\ln(\ln(n)))^2)$.

2. **Estimating sums by integrals.** Using the method of estimating a sum by an integral, show that

(a) $\sum_{i=1}^n 1/i = \ln(n) + O(1)$
(b) $\sum_{i=1}^n \ln(i) = n \ln(n) + O(n)$
(c) $\sum_{i=1}^n i \ln(i) = \frac{1}{2} n^2 \ln(n) + O(n^2)$

Recall that this method applies when $f$ is continuous and monotone on an interval $[a, b]$, where $a$ and $b$ are integers, and says that

$$\min(f(a), f(b)) \leq \sum_{i=a}^b f(i) - \int_a^b f(x) \, dx \leq \max(f(a), f(b)).$$

Note: just use known facts about integrals, from a textbook or an online calculator. Also note: when we say $f = g + O(h)$, we mean that $|f - g| = O(h)$.

3. **Mystery algorithm.** Consider the following algorithm, which operates on an array $A[1 \ldots n]$ of integers.

for $i \leftarrow 1$ to $n$ do
  $A[i] \leftarrow 0$
for $i \leftarrow 1$ to $n$ do
  $j \leftarrow i$
  while $j \leq n$ do
    $A[j] \leftarrow A[j] + 1$
    $j \leftarrow j + i$

(a) Show that the running time of this algorithm is $O(n \log n)$.
(b) Describe in words the value of $A[i]$ at the end of execution.

4. **Sieve of Eratosthenes.** The Sieve of Eratosthenes is an algorithm that works on an array $A[2 \ldots n]$ of bits, as follows:

for $i \leftarrow 2$ to $n$ do
  $A[i] \leftarrow 1$
for $i \leftarrow 2$ to $\lfloor \sqrt{n} \rfloor$ do
  if $A[i] = 1$ then
    $j \leftarrow 2i$
    while $j \leq n$ do
      $A[j] \leftarrow 0$
      $j \leftarrow j + i$

(a) Show that at the end of execution, for $i = 1, \ldots, n$, we have $A[i] = 1$ if and only if $i$ is prime.
(b) It is a fact that

$$\sum_{p \leq n} \frac{1}{p} = \ln(\ln(n)) + O(1),$$

where the sum is over all primes $p$ up to $n$. Use this fact to show that the running time of this algorithm is $O(n \log \log n)$. 

1
5. **Weird recursion tree analysis.** Suppose we have an algorithm that on problems of size \( n \), recursively solves two problems of size \( n/2 \), with a “local running time” of \( O(f(n)) \) for some function \( f(n) \). That is, the algorithm’s total running time satisfies the recurrence \( T(n) \leq 2T(n/2) + O(f(n)) \). For simplicity, assume that \( n \) is a power of 2.

Prove the following using a recursion tree analysis:

(a) If \( f(n) = n \log n \), then \( T(n) = O(n(\log n)^2) \).
(b) If \( f(n) = n/\log n \), then \( T(n) = O(n \log \log n) \).
(c) If \( f(n) = n/(\log n)^2 \), then \( T(n) = O(n) \).

6. **Uneven divide and conquer.** Suppose we have an algorithm that on problems of size \( n \), recursively solves \( k \) subproblems of unequal size: for \( i = 1, \ldots, k \), the \( i \)th subproblem is of size \( \lfloor n/b_i \rfloor \). Here, each \( b_i \) is a constant greater than 1, and \( k \) is also a constant. Furthermore, the “local running time” is \( O(n) \). That is, the algorithm’s total running time satisfies the recurrence

\[
T(n) \leq \sum_{i=1}^{k} T(\lfloor n/b_i \rfloor) + cn
\]

for some constant \( c \). You may assume the above inequality holds for all \( n \geq 1 \), and that \( T(0) = 0 \).

Let \( \delta := \sum_{i=1}^{k} 1/b_i \). Assuming that \( \delta < 1 \), use the method of recursion trees to prove that \( T(n) = O(n) \).

Hint: prove by induction on \( j \) the following: at each level \( j = 0, 1, 2, \ldots \) of the recursion tree, the sum of the subproblem sizes at level \( j \) is at most \( \delta^j n \).