**Question 1** (Multiplication Runtime, 30 points). *Consider the following divide and conquer multiplication algorithm. What is the asymptotic runtime for computing the product of two n-bit numbers?*

```plaintext
Multiply(N, M)
  if max(N, M) < 1000
    return N*M
  let X be a power of 2, approximately max(N, M)^(1/3)
  let N = AX^2 + BX + C, M = DX^2 + EX + F
  P1 = Multiply(A, D)
  P2 = Multiply(C, F)
  P3 = Multiply(A, E)
  P4 = Multiply(B, D)
  P5 = Multiply(A+B+C, D+E+F)
  P6 = Multiply(A-B+C, D-E+F)
  return P1*X^4 + [P3+P4]*X^3 + [P5/2+P6/2-P1-P2]*X^2 + [P5/2-P6/2-P3-P4]*X + P2
```

The algorithm spends $\Theta(n)$ time writing $N$ and $M$ in the appropriate form, and $\Theta(n)$ time performing necessary additions. It also performs 6 recursive multiplications on numbers with $n/3 + O(1)$ bits. Therefore the runtime satisfies the recurrence

$$T(n) = 6T(n/3 + O(1)) + \Theta(n).$$

Noting that $6 > 3^1$, by the Master Theorem, our final runtime is

$$T(n) = \Theta(n^{\log_3(6)}) = \Theta(n^{1.63\ldots}).$$
**Question 2** (Comparison Counting, 35 points). *Given two sets of numbers, A and B each of size n, give an algorithm to compute the number of pairs of \( a \in A, b \in B \) so that \( a > b \). The algorithm should run in time \( O(n \log(n)) \) or better for full credit.*

One such algorithm is as follows:

1. Sort \( B \)
2. Let \( \text{tot} = 0 \)
3. For each \( a \) in \( A \)
   - Binary Search \( B \) to find the predecessor of \( a \) in \( B \)
   - Let \( k \) be the index of the last element of \( B \) less than \( a \)
   - \( \text{tot} = \text{tot} + k \)
4. Return \( \text{tot} \)

Note that in each iteration of the for loop, \( k \) will be the number of elements of \( B \) that are less than \( a \). Summing this over all \( a \in A \), yields the answer that we want.

For the runtime, note that sorting \( B \) take \( O(n \log(n)) \) time. For each of the \( n \) elements of \( A \), we then need to binary search \( B \), which takes \( O(\log(n)) \) time. Therefore, the final runtime is

\[
O(n \log(n)) + nO(\log(n)) = O(n \log(n)).
\]
**Question 3** (Task Scheduling, 35 points). Dave has a number of tasks to accomplish each with a deadline $d_i$ and a time $t_i$ that it takes to complete. Dave can only work on one task at a time, but wishes to complete each task before it’s appropriate deadline. Dave has an idea for a greedy procedure for accomplishing this if it is possible. In particular, he plans to work on tasks starting with the earliest deadline and working in order of increasing deadline. So, for example if he has three tasks that take times 1, 2, and 3 and have deadlines of 4, 3, and 7, respectively, he will work on the second task from times 0 to 2, the first task from times 2 to 3, and the third task from times 3 to 6, completing each before its deadline.

Prove that this procedure is guaranteed to finish all tasks by their deadline if it is possible to do so.

First sort the deadlines in increasing order, so that $d_1 \leq d_2 \leq \ldots \leq d_n$. Dave’s procedure suggests that he work on tasks in order $1, 2, 3, \ldots, n$. Suppose that this procedure does not complete all tasks by their appropriate deadline. That means that there is some $k$ so that it completes the $k^{th}$ task after $d_k$. In other words, this means that $t_1 + t_2 + \ldots + t_k > d_k$. On the other hand, this means that there is not enough time to complete all of the first $k$ tasks by time $d_k$. Therefore, no matter what Dave does, at least one of these tasks, say the $j^{th}$ one for some $j \leq k$, will remain incomplete at time $d_k$. However, since $d_j \leq d_k$, this means that no matter what Dave does at least one task will not be finished by its deadline.