1. Using $\Theta$-notation, provide asymptotically tight bounds in terms of $n$ for the solution to each of the following recurrences.

(a) $T(n) = 8T(n/2) + n^3$
(b) $T(n) = 5T(n/3) + n$
(c) $T(n) = T(n-1) + n^2$
(d) $T(n) = T(n/2) + 2T(n/5) + n$
(e) $T(n) = T(\sqrt{n}) + T(\frac{\sqrt{n}}{2}) + \log n$

Solution:

(a) Apply the master theorem with $a = 8, b = 2, d = 3$. Then, $a(n/b)^d = 8 \cdot (n/2)^3 = n$. The third case applies so, $T(n) = \Theta(n^3 \log n)$.
(b) Apply the master theorem with $a = 5, b = 3, d = 1$. Then, $a(n/b)^d = 5 \cdot (n/3)^1 = (5/3)n$. The second case applies so $T(n) = \Theta(n \log_3 5)$.
(c) $T(n) = n^2 + (n-1)^2 + (n-3)^2 + \cdots = \sum_{i=1}^{n} i^2 - \sum_{i=1}^{n} \Theta(1) = \Theta(n^3)$.
(d) Use recursion trees. The nodes at depth $i$ sum to at most $(9/10)^i n$. Summing over all the rows gives a decreasing geometric series bounded by its largest term, $n$, so $T(n) = \Theta(n)$.
(e) We'll use the same transformation we saw in class. Let $m = \log n$. Let $S(m) = T(2^m) = T(2^{m/2}) + T(2^{m/3}) + m = S(m/2) + S(m/3) + m$. To solve this second recurrence, use recursion trees. The nodes at depth $i$ sum to at most $(5/6)^i m$. Summing over all the rows gives a decreasing geometric series bounded by its largest term, $m$, so $S(m) = \Theta(m)$ and $T(n) = \Theta(\log n)$.

Rubric: 10 points total: 2 points per part, 1 point for only a tight upper or lower bound, half credit if no explanation given.

2. Suppose you are given a matrix (a 2-dimensional array) $A[1..m][1..n]$ of numbers. An element $A[i][j]$ is called good if each of its neighbors $A[i-1][j], A[i+1][j], A[i][j-1], \text{ and } A[i][j+1]$ are at most $A[i][j]$.

(a) Suppose $m = 1$ so we only have the array $A[1][1..n]$. Design and analyze an algorithm to find a good element of $A$ in $O(\log n)$ time.

Solution: The $O(\log n)$ strongly suggests some kind of binary search. The procedure $\text{FindGood}(A[1][1..n])$ returns a good element of $A[1][1..n]$ assuming that elements beyond the boundary of $A$ are equal to $-\infty$. 
We will now prove correctness of \texttt{FindGood}. If \( n = 1 \), then the sole element is at least as great as its four neighbors beyond the boundaries of \( A \), and it is correct to return it. For larger \( n \), we will assume inductively that the procedure is correct for arrays \( A'[1 .. n'] \) where \( 1 \leq n' < n \). The algorithm checks if \( A[1][(n/2)] \) is at least as great as the two neighbors that may be within the confines of \( A[1 .. n] \). If it is at least as great as those members, then the algorithm correctly returns \( A[1][(n/2)] \). Otherwise, one of the neighbors, say \( A[1][(n/2) - 1] \), is greater. By the inductive hypothesis, the procedure finds a good element \( A[1][i] \) of \( A[1 .. (n/2) - 1] \). If \( i < (n/2) - 1 \), then this element is good in \( A[1][1 .. n] \) as well, because it has the same neighbors.

Is it good if \( i = (n/2) - 1 \), also, because we already know \( A[1][i] > A[1][(n/2)] \). Therefore, the algorithm is correct to return \( A[1][i] \). A similar argument holds if only \( A[1][(n/2) + 1] \) is greater than \( A[1][(n/2)] \).

Ignoring floors, the running time of the algorithm observes the recurrence \( T(n) = T(n/2) + \Theta(1) \). The solution is \( T(n) = \Theta(\log n) \) by the Master theorem (also, the algorithm follows the normal structure of a binary search).

\begin{itemize}
\item \textbf{Rubric:} 5 points total: 3 points for the algorithm, 1 point for the proof, 1 point for the running time analysis.
\end{itemize}

(b) Now suppose \( m = n \). Design and analyze an algorithm to find a good element of \( A \) in \( O(n \log n) \) time.

\textbf{Solution:} Again, we will do a binary search, but now we will search over entire columns of \( A \) by comparing their maximum elements. The procedure \texttt{FindGoodSquare}(\( A[1 .. m][(1 .. n)] \)) returns a good element of \( A[1 .. m][(1 .. n)] \) assuming that elements beyond the boundary of \( A \) are equal to \(-\infty\). \textit{This good element is guaranteed to be the maximum of its column} \( A[.][j] \).

\begin{verbatim}
FindGoodSquare(A[1 .. m][(1 .. n)]):
  // Find maximum element in column [(n/2)]
  p ← 0
  for i ← 1 to m
    if A[i][(n/2)] > A[p][(n/2)]
      p ← i
    if A[p][(n/2)] < A[p][(n/2) − 1]
      return FindGoodSquare(A[1 .. m][(1 .. (n/2) − 1)])
    else if A[p][(n/2)] < A[p][(n/2) + 1]
      return FindGoodSquare(A[1 .. m][(1 .. (n/2) + 1 .. n)])
  return A[p][(n/2)]
\end{verbatim}

We will now prove correctness for \texttt{FindGoodSquare} by doing induction on \( n \), the number of columns in the matrix. If \( n = 1 \) the procedure finds and returns the maximum element in the only column, and that element is good since it's neighbors to the left and right are \(-\infty\). For larger \( n \), we will assume inductively that the procedure returns a good element when there are fewer than \( n \) columns, and that
it is the largest element in its column. Now, if the largest element in column \(\lfloor n/2 \rfloor\) is good, the algorithm correctly returns it. Otherwise, one of its neighbors to the left or right is larger, and the procedure inductively returns a good element that is maximum for its column from the half of \(A\) containing the larger element. If that good element does not lie adjacent to column \(\lfloor n/2 \rfloor\) then it has the same neighbors in \(A\) as in the recursive call and must be good. Otherwise, it is the largest element in its column, meaning it is larger than the neighbor of column \(\lfloor n/2 \rfloor\)'s largest element. The recursively found element is larger than all the elements in column \(\lfloor n/2 \rfloor\) as well as its neighbors in the recursive call, so it is good.

Ignoring floors, the number of searches for a column maximum follows the recurrence \(T(n) = T(n/2) + 1\). Again, the solution is \(T(n) = \Theta(\log n)\). It takes \(\Theta(m)\) time to search for a column max, so the overall running time is \(\Theta(m \log n) = \Theta(n \log n)\). ■

Rubric: 5 points total: 3 points for the algorithm, 1 point for the proof, 1 point for the running time analysis.

(c) Extra credit: Design and analyze an algorithm to find a good element of \(A\) in \(O(n)\) time.

Solution: We will follow a similar strategy to the above two algorithms, except now we will cut up the matrix into four squares by computing the maximum element on the boundary or through the middle row or column. The procedure \(\text{FINDGOODFAST}(A[1 \ldots n][1 \ldots n])\) returns a good element of \(A[1 \ldots n][1 \ldots n]\) assuming that elements beyond the boundary of \(A\) are equal to \(-\infty\). If this good element lies on the boundary of \(A\), then it is the maximum such element. The following pseudocode is more detailed than necessary for full credit.
We will now prove correctness for `FindGoodFast` by doing induction on \( n \). If \( n = 1 \) the procedure correctly finds and returns the one element in the matrix. For larger \( n \), we will assume inductively that the procedure returns a good element in matrices of size \( n' \times n' \) for \( 1 \leq n' < n \). Further, if this good element lies on the boundary of the matrix, then it is the maximum such element. The first group of lines in `FindGoodFast` find the maximum element \( A[p][q] \) along the boundary of \( A \) and its median rows and columns by iteratively checking every such element. Next, the procedure checks if it has any larger neighbors \( A[r][s] \); if not, the procedure correctly
returns $A[p][q]$. And indeed, if $A[p][q]$ lies on the boundary of $A$, it is the largest such element as we guaranteed. If $A[p][q]$ is not good, then note the elements checked by the first group of lines break $A$ into four $(\lfloor n/2 \rfloor - 1) \times (\lfloor n/2 \rfloor - 1)$ square sub-matrices. By the inductive hypothesis, a good element for the sub-matrix holding $A[r][s]$ is returned. If the element does not lie on the boundary of the sub-matrix, then it has the same neighbors in $A$ and is good for $A$. Otherwise, it is the largest element on the boundary of the sub-matrix, meaning it is at least as large as $A[r][s] > A[p][q]$, which is greater than all elements surrounding the sub-matrix. The recursively computed element is good for $A$ as well and is correctly returned.

Ignoring floors, the running time of the algorithm observes the recurrence $T(n) = T(n/2) + \Theta(n)$. The solution is $T(n) = \Theta(n)$ by the Master theorem.

**Rubric:** 5 points extra credit total: 3 points for the algorithm, 1 point for the proof, 1 point for the running time analysis.