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

(a) $A(n) = 4A(n/2) + n^2$

**Solution:** Each level of the recursion tree sums to $n^2$. There are $O(\log n)$ levels, so $A(n) = \Theta(n^2 \log n)$. □

(b) $B(n) = 7B(n/2) + n^3$

**Solution:** The $i$th level of the recursion tree sums to $(7/8)^i n^3$, so the sum of level sums is a decreasing geometric series proportional to its largest term, the value of the root node. $B(n) = \Theta(n^3)$. □

(c) $C(n) = 5C(n/3) + n$

**Solution:** The $i$th level of the recursion tree sums to $(5/3)^i n$, so the sum of level sums is an increasing geometric series proportional to its largest term, the number of leaves. $C(n) = \Theta(n \log_3 5)$. □

(d) $D(n) = D(n/4) + D(n/2) + n$

**Solution:** The $i$th level of the recursion tree sums to at most $(3/4)^i n$, so the sum of level sums is at most a decreasing geometric series proportional to its largest term, the value of the root node. The sum of node values is also at least the value of the root node, so $D(n) = \Theta(n)$. □

(e) $E(n) = 3E(n/3) + n \lg n$

**Solution:** The $i$th level of the recursion tree sums to $n \lg(n/3^i) = n \lg n - i n \lg 3$, so the sum of level sums is a decreasing arithmetic series.

If we recall the formula for an arithmetic series, we see $E(n) = \Theta(n \log^2 n)$. Alternatively, we may observe that each of the first, say, $\lfloor (\log_3 n) / \lg 3 \rfloor$ levels sum to at least $n \lg n - n \log_3 n = \Omega(n \log n)$ (the specific number of levels is not important; we only need it to be a constant fraction of the tree’s depth). Also, all of the $O(\log n)$ levels sum to at most $n \lg n$, so again, $E(n) = \Theta(n \log^2 n)$. □

**Rubric:** 2 points per part. −1/2 points per part missing some kind of justification.
Describe and analyze an algorithm to count the number of inversions in an $n$-element array in an $O(n \log n)$ time.

**Solution:** As suggested, we'll slightly modify mergesort so it both sorts the input array and counts the inversions that were present before sorting. Procedure `CountAndMergeSort(A[1 .. n])` sorts $A$ and returns the number of inversions that were present in $A$ before it was sorted. It uses procedure `CountAndMerge(A[1 .. n], m)` that, assuming $A[1 .. m]$ and $A[m + 1 .. n]$ are both sorted, sorts $A[1 .. n]$ and returns the number of inversions in $A$ that were present before it was sorted by `CountAndMerge`.

```plaintext
CountAndMergeSort(A[1 .. n]):
  count ← 0
  if n > 1
    m ← ⌊n/2⌋
    count ← count + CountAndMergeSort(A[1 .. m])
    count ← count + CountAndMergeSort(A[m + 1 .. n])
    count ← count + CountAndMerge(A[1 .. n], m)
  return count

CountAndMerge(A[1 .. n], m):
  count ← 0
  i ← 1; j ← m + 1
  for k ← 1 to n
    if j > n
      B[k] ← A[i]; i ← i + 1
    else if i > m
      B[k] ← A[j]; j ← j + 1
    else if A[i] ≤ A[j]
      B[k] ← A[i]; i ← i + 1
    else
      count ← count + (m − i + 1)
      B[k] ← A[j]; j ← j + 1
  for k ← 1 to n
    A[k] ← B[k]
  return count
```

To prove correctness, we first observe that all changes to the input array are exactly the same as those done by the standard mergesort algorithm (although we now assign $A[i]$ to $B[i]$ when the merge procedure encounters a tie).

Now, consider the procedure `CountAndMerge`. We claim that for any iteration $k$ of the main for loop, along with all the other conditions proved about `Merge` during lecture, we will also increase the `count` variable by the number of inversions $(i', j')$ such that $i ≤ i' ≤ m$ and $j ≤ j' ≤ n$. The special case $k = 1$ implies `count` will be increased by exactly the number of inversions with one entry in $A[1 .. m]$ and one entry in $A[m+1 .. n]$. Therefore, `CountAndMerge` returns the number of inversions claimed earlier.
We now prove the claim. If $k = n + 1$, then there are no such $i'$ and $j'$, so we’re correct not to touch $\text{count}$. Suppose $k \leq n$. If either of the first two cases of the if else chain hold, then there are no such $i'$ or $j'$. It is trivially true than the all 0 inversions $(i', j')$ also appear in subarrays considered by the later iterations, so $\text{count}$ never increases again by induction. Suppose $A[i] \leq A[j]$. Then $A[i] \leq A[j']$ for all $j \leq j' \leq n$, because $A[j .. n]$ is sorted. We conclude there are no inversions of the form above with $i' = i$. We will correctly do all increases to $\text{count}$ after incrementing $i$ for later iterations. Finally, suppose $A[i] > A[j]$. We have $A[i'] \leq A[j]$ for all $i \leq i' \leq m$, because $A[i .. m]$ is sorted. Therefore, all $m - i + 1$ pairs $(i', j)$ are inversions, and we are fine to increase $\text{count}$ by that amount. Having considered every pair $(i', j)$, we know $j + 1 \leq j' \leq n$ in all remaining inversions. Those contribute to $\text{count}$ the right amount by induction.

Finally, we argue correctness of $\text{CountAndMergeSort}$ itself. There are clearly no inversions if $n \leq 1$. Otherwise, all inversions $(i, j)$ are one of three types: (a) $1 \leq i < j \leq m$, (b) $m + 1 \leq i < j \leq n$, and (c) $1 \leq i \leq m$ while $m + 1 \leq j \leq n$. The first recursive call returns the number of type (a) inversions while also sorting $A[1 .. m]$. This sorting does not affect the number type (b) or (c) inversions, because $j \geq m + 1$ in both cases. The second recursive call returns the number of type (b) inversions while also sorting $A[m + 1 .. n]$. Again, the number of type (c) inversions is unaffected. Finally, we just argued the $\text{CountAndMerge}$ returns the number of type (c) inversions. We correctly return the sum of all three counts.

Finally, for running time, we observe that our changes to mergesort introduce constant time operations next to those that existed before. The running time increases by at most a constant factor, so our algorithm runs in $O(n \log n)$ time.

\textbf{Rubric:} 10 points total: 5 points for the algorithm, 3 points for the justification, and 2 points for the running time.

- 2 points for increasing $\text{count}$ by only 1 in the last if else case. -1 point for increasing $\text{count}$ by $m - i$. 

2