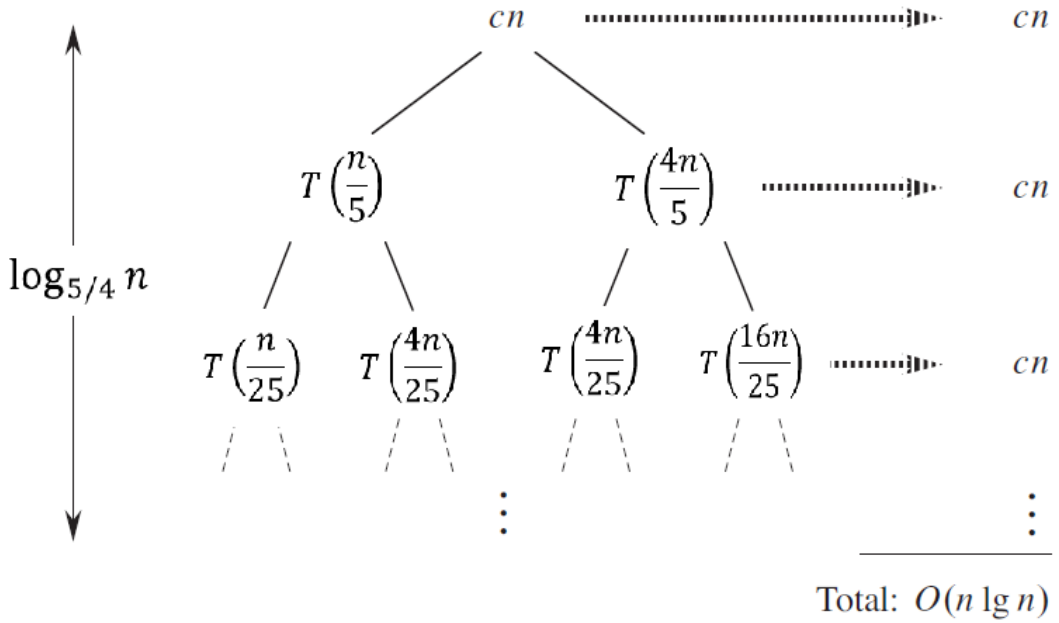
Mid-1 (Solution)

Question 1:

a) As $\mathbf{4^n} \leq (1000)2^n + 4^n \leq (1001)\mathbf{4^n}$ for all $n \geq 1$, therefore, $(1000)2^n + 4^n = \mathbf{\Theta(4^n)}$. Here, $c_1 = 1, c_2 = 1001,$ and $n_0 = 1$.

b) Recursion Tree for $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + n$ is given below:



Hence, the solution is $O(n \lg n)$.

c) We can solve the recurrence $T(n) = 25T\left(\frac{n}{5}\right) + n^2$, using the Master Method. Here, $a = 25, b = 5,$ and $f(n) = n^2$. As $n^{\log_5 25} = \Theta(n^2)$, the solution is $\mathbf{\Theta(n^2 \lg n)}$.

d) $2^{n+12} = O(2^n)$ as $2^{n+12} \leq c2^n$ for all $n \geq 1$, where c is a constant $\geq 2^{12} = 4096$.

e) $4^{12n} \neq O(2^n)$. To justify the answer, let's assume for the sake of contradiction that $4^{12n} = O(2^n)$. Then there exist positive constants c and n_0, such that $4^{12n} \leq c2^n$ for all $n \geq n_0$. Let's try to find the value of c: as $4^{12n} \leq c2^n$ it is same as $2^{24n} \leq c2^n$, or $2^{23n} \leq c$. Now, this is not possible because c is a constant and 2^{23n} is a variable having a positive growth rate. Hence, $4^{12n} \neq O(2^n)$.

f) The basic idea is that the last three elements must be included in the last call to sort the first seven elements (otherwise some element/s among the last three may not be included in any of the calls for MergeSort, and if any such element happens to have smaller value than the first seven elements, the final output would not be sorted).

i. Hence, the minimum value of $y - x + 1$ should be **6** to correctly sort the input.

ii. MergeSort(A, x, y) should be called as MergeSort(A, 5, 10).

Question 2:

```
CountAndSort(A, p, r)
1 if p < r
2     q=⌊p+r/2⌋
3     x=CountAndSort(A, p, q)
4     y=CountAndSort(A, q+1, r)
5     z=CountAndMerge(A, p, q, r)
6     return x+y+z
7 else return 0
```

```
CountAndMerge(A, p, q, r)
1 n1 = q - p + 1
2 n2 = r - q
3 let L[0…n1+1] and R[0…n2+1] be new arrays
4 for i = 1 to n1
5     L[i] = A[p + i – 1]
6 for j = 1 to n2
7     R[j] = A[q + j]
8 L[n1 + 1] = ∞
9 R[n2 + 1] = ∞
10 i = 1
11 j = 1
$$ count=0
12 for k = p to r
13    if L[i] <= R[j]
$$          count=count+(n2-j+1) //L[i] is compatible with all R[j…n2]
14          A[k] = L[i]
15          i = i + 1
16    else A[k] = R[j]
17          j = j + 1
$$ return count
```

The initial call would be `CountAndSort(A, 1, n).`