Design and Analysis of Computer Algorithms

11059005 蕭耕宏

Assignment 1

Due Mar 24, 2025 (before class)

Please use A4-sized papers to write your answers for the homework sets.

1. Show that the solution to

$$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor + 17\right) + n$$

   is $\mathrm{O}(n \log n)$.

   Solution:

   1. Base Case, Assume T(n) is constant for $n < n_0$
   2. Inductive hypothesis, Assume T(n) is $\mathrm{O}(n \log n)$ for $n < n_0$. That is $T(k) \le ck \log k$ for some constant c > 0 and k < n.
   3. Inductive step, for T(n),

$$T(n) \le cn \log n$$

      for some constant c > 0.

      $T(n) = 2T(\lfloor \frac{n}{2} \rfloor + 17) + n$

      By assumption,

      $T(n) \le 2c(\lfloor \frac{n}{2} \rfloor + 17) \log(\lfloor \frac{n}{2} \rfloor + 17) + n$

      For large n, $\lfloor \frac{n}{2} \rfloor \approx \frac{n}{2}$ and $\log(\frac{n}{2} + 17) \approx \log \frac{n}{2} = \log n - \log 2$.

      $T(n) \le 2c(\frac{n}{2} + 17) \log(\frac{n}{2} + 17) + n$

      $\approx cn(\log n - \log 2) + 34c(\log n - \log 2) + n$

      we need this to be $\le cn \log n$ for some constant c > 0.

      $cn(\log n - \log 2) + 34c(\log n - \log 2) + n \le cn \log n$

      $cn \log n - cn \log 2 + 34c \log n - 34c \log 2 + n \le cn \log n$

      $34c \log n - cn \log 2 - 34c \log 2 + n \le 0$

      $34c(\log n - \log 2) - n(1 - c \log 2) \le 0$

      $34c\left(\frac{\log n}{n} - \log \frac{2}{n}\right) - (1 - c \log 2) \le 0$

      As $n \to \infty$, $\left(\frac{\log n}{n}\right) \to 0$ and $\log \frac{2}{n} \to 0$.

      $1 - c \log 2 \le 0$

      $c \log 2 \ge 1$

      $c \ge \frac{1}{\log 2} \approx 1.44$ suppose the base of the logarithm is e.

      Therefore, $T(n) = \mathrm{O}(n \log n)$ for $n \ge 1.44 > 0$.

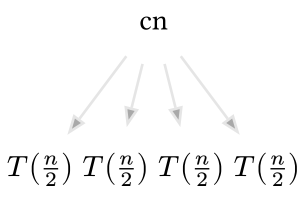
2. Draw the recursion tree for

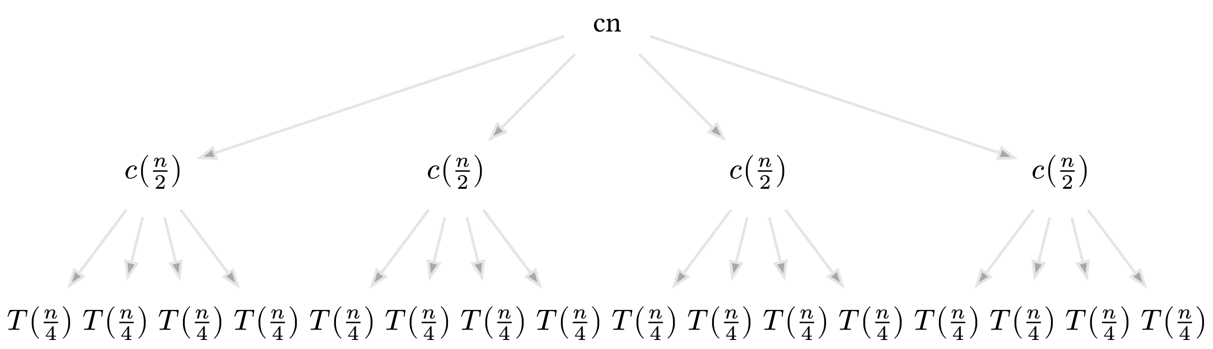
$$T(n) = 4T\left(\frac{n}{2}\right) + cn$$

, where c is a constant,

and provide a tight asymptotic bound on its solution. Verify your bound by the substitution method.

Solution:


cn
$T(\frac{n}{2})$ $T(\frac{n}{2})$ $T(\frac{n}{2})$ $T(\frac{n}{2})$


cn
$c(\frac{n}{2})$ $c(\frac{n}{2})$ $c(\frac{n}{2})$ $c(\frac{n}{2})$
$T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$

The tree height is $\log_2 n$ and there are $4^{\log_2 n} = n^2$ leaves.

The cost of each leaf is dn^2 where d is a constant.

For non-leaf nodes, the cost is

$cn\left(1 + 2 + 2^2 + \ldots + 2^{\log_2 n - 1}\right)$

$= cn \sum_{i=0}^{\log_2 n - 1} 2^i$

$= cn(n - 1)$

In total, the cost is $dn^2 + cn(n-1) = dn^2 + cn^2 - cn = (c+d)n^2 - cn$.

For its asymptotic upper bound, we have

$T(n) = (c+d)n^2 - cn \leq (c+d)n^2$

Therefore, $T(n) = \mathrm{O}(n^2)$.

Using the substitution method, assume $T(n) \leq bn^2 - cn$ for some constant b > 0 to show that $T(n) = \mathrm{O}(n^2)$.

$T(n) = 4T(\frac{n}{2}) + cn$

$\leq 4\left(b(\frac{n}{2})^2 - c(\frac{n}{2})\right) + cn$

$= bn^2 - 2cn + cn$

$= bn^2 - cn$

$\leq bn^2$

For its asymptotic lower bound, we have

$T(n) = (c + d)n^2 - cn \geq (c + d)n^2$

Therefore, $T(n) = \Omega(n^2)$.

Using the substitution method, assume $T(n) \geq bn^2$ for some constant b > 0 to show that $T(n) = \Omega(n^2)$.

$T(n) = 4T(\frac{n}{2}) + cn$

$\geq 4b(\frac{n}{2})^2 + cn$

$= bn^2 + cn$

$= bn^2 + cn$

$\geq bn^2$

Therefore, $T(n) = \Theta(n^2)$.

3. Find the asymptotic order of the following recurrence relations. Assume that T(n) is constant for n ≤ 2. Make your bounds as tight as possible, and justify your answers.

a.

$$T(n) = 7T\left(\frac{n}{3}\right) + n^2$$

Solution:

$a = 7, b = 3, f(n) = n^2$

$h(m) = \frac{m^2}{m^{\log_3 7}} = \Omega(m^r)$ where $r = 2 - \log_3 7 > 0$ (Master method case 3)

$T(n) = n^{\log_3 7}\left[T(1) + n^{2 - \log_3 7}\right] = n^{\log_3 7}\left[1 + n^{2 - \log_3 7}\right] = n^{\log_3 7} + n^2$

Therefore, $T(n) = \Theta(n^2)$.

b.

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

Solution:

$a = 4, b = 2, f(n) = n$

$h(m) = \frac{m}{m^2} = \mathrm{O}(m^r)$ where $r = -1 < 0$ (Master method case 1)

$T(n) = n^2[T(1) + O(1)] = n^2[1 + \mathrm{O}(1)] = n^2$

Therefore, $T(n) = \Theta(n^2)$.

c.

$$T(n) = 2T\left(\frac{n}{2}\right) + n^3$$

Solution:

$a = 2, b = 2, f(n) = n^3$

$h(m) = \frac{m^3}{m} = m^2 = \Omega(m^r)$ where $r = 2 > 0$ (Master method case 3)

$T(n) = n\left[T(1) + n^2\right] = n\left[1 + n^2\right] = n + n^3$

Therefore, $T(n) = \Theta(n^3)$.

d.

$$T(n) = 4T\left(\frac{n}{3}\right) + n$$

Solution::

$a = 4, b = 3, f(n) = n$

$h(m) = \frac{m}{m^{\log_3 4}} = \mathrm{O}(m^r)$ where $r = 1 - \log_3 4 < 0$ (Master method case 1)

$T(n) = n^{\log_3 4}[T(1) + \mathrm{O}(1)] = n^{\log_3 4}[1 + \mathrm{O}(1)] = n^{\log_3 4}$

Therefore, $T(n) = \Theta\left(n^{\log_3 4}\right)$.

4. Please use the Master method to find the tight asymptotic upper bound for the following recursion:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

.

Solution:

$a = 2, b = 2, f(n) = n \log n$

$h(m) = \frac{m \log m}{m} = \log m = \Theta(\log m)^i$ where $i = 1$ (Master method case 2)

$T(n) = n\left[T(1) + \Theta\left(\frac{\log^2 n}{2}\right)\right] = n\left[1 + \frac{\Theta((\log^2 n))}{2}\right] = n + n(\log n)^2$

Therefore, $T(n) = \Theta(n \log^2 n)$.

5. Given a sequence of n distinct numbers $a_1, a_2, ..., a_n$, we define a significant inversion to be a pair $i < j$ such that $a_i > 2a_j$. Give an $\mathrm{O}(n \log n)$ algorithm to count the number of significant inversions in the given sequence of n distinct numbers $a_1, a_2, ..., a_n$.

Solution:

A standard merge sort algorithm can be used count the number of inversions in the sequence.

To count the number of significant inversions in this case, we can dynamically store elements in a balanced BST which records the size of the left subtree of each node.

1. Initialize a counter to 0.
2. Initialize an empty BST.
3. For each element in the sequence, count the number of elements less than $\frac{a_i}{2}$ in the BST.
4. Insert the element into the BST.
5. Return the counter.

```
FindSignificantInversions($A$):
n = A.length
if n <= 1                                       // O(1)
    return 0                                    // O(1)
count = 0                                       // O(1)
BST = new BST()                                 // O(1)
for i = n- 1 to 0                               // O(n)
    count += BST.countLessThan(A[i] / 2)        // O(n log n)
    BST.insert(A[i])                            // O(n log n)
return count                                    // O(1)
```

A balanced BST without augmenting the subtree size will not be able to count the number of elements less than $\frac{a_i}{2}$ in $O(\log n)$ time. In the worst case, we need to traverse the entire tree to count the number of elements less than $\frac{a_i}{2}$, which results in $O(n)$. By augmenting the subtree size, we can count the number of elements less than $\frac{a_i}{2}$ in $O(\log n)$ time. Since the balanced BST records the size of the left subtree, BST.countLessThan queries the BST in $O(\log n)$ time. Also, the insertion of an element into the BST takes $O(\log n)$ time for rebalancing. Therefore, the total time complexity of the algorithm is $O(n \log n)$.