CSE 21 HW 5

Brian Masse

Februrary 19, 2025

1. **Consider the algorithm IntersectCount that takes two sorted lists of distinct integers $a[1], \ldots, a[n]$ and $b[1], \ldots, b[n]$ and returns the number of elements they have in common (the cardinality of their intersection:)**

   a) What is the expected number of entries equal to zero using this sampling method?

   base case: t = 1

   - *case 1: count = 1 if* $a[1] \in (b[1]...b[n])$
   - *case 2: count = 0 if* $a[1] \notin (b[1]...b[n])$

   induction step: for t > 1: assume the loop invariant holds. Show that it also holds for t + 1

   - *case 1:* $a[t+1] \notin (b[1]...b[n])$
     count (by induction) = number of intersections $(a[1]...a[t{+}1])$ and $(b[1]...b[n])$

   - *case 2:* $a[t+1] \in (b[1]...b[n])$
     count = count + 1 = (by induction) number of intersections $(a[1]...a[t{+}1])$ and $(b[1]...b[n])$

   b) Do a runtime analysis and give a Big Theta bound for the runtime

   - Outside Loop = $\theta(n)$
   - Outside Loop = $\theta(log_2(n))$
   - Outside Loop = $\theta(n \cdot log_2(n))$

2. **For each situation below, first give the recurrence for the runtime of the algorithm. Then use the Master Theorem, if possible, and give the values for the parameters a b and d, and the O bound.**

a) Suppose an algorithm solves a problem of size n by recursively calling 3 sub-problems each of size $\frac{4n}{5}$. Then the non-recursive part of the algorithm takes $O(n)$ time.

$$T(n) = 3 \cdot T(\frac{4n}{5}) + O(n)$$

$a = 3$, $b = \frac{4n}{5}$, $d = 3$

$$\begin{aligned}
& n^{log_{5/4}(3)} > n^4 \implies \lim_{n\to\infty} \frac{n}{n^4} = 0 & (1)\\
\implies\ & f(n) \in O(n) \in O(n^4) = O(n^{log_b(a)-\epsilon}) & (2)\\
\implies\ & T(n) = \theta(n^{log_{5/4}(3)}) & (3)\\
& & (4)
\end{aligned}$$

b) Suppose an algorithm solves a problem of size n by recursively calling 9 sub-problems each of size $\frac{n}{4}$. Then the non-recursive part of the algorithm takes $O(n^2)$ time.

$$T(n) = 9 \cdot T(\frac{n}{4}) + O(n^2)$$

$a = 9$, $b = 4$, $d = 2$

$$\begin{aligned}
& n^{log_4(9)} < n^2 & (5)\\
\implies\ & f(n) \in \Omega(n^2) \in \Omega(n^{log_4(9)+\epsilon}) & (6)\\
\implies\ & T(n) = \theta(n^{log_4(9)}) & (7)\\
& & (8)
\end{aligned}$$

c) Suppose an algorithm solves a problem of size n by recursively calling 8 sub-problems each of size $\frac{n}{4}$. Then the non-recursive part of the algorithm takes $O(n\sqrt{n})$ time.

$$T(n) = 8 \cdot T(\frac{n}{4}) + O(n \cdot \sqrt{n})$$

$a = 9$, $b = 4$, $d = 2$

$$\begin{aligned} & n^{log_4(8)} < n^{1.5} && (9) \\ \implies \quad & f(n) = \Theta(n^{1.5}) = \Theta(n^{log_4(8)}) && (10) \\ \implies \quad & T(n) = \Theta(n^{1.5} \cdot log(n)) && (11) \\ & && (12) \end{aligned}$$

3. **Consider the following sorting algorithm that takes a list of integers as an input and outputs a sorted list of those elements.**

Consider the loop invariant: *After each iteration, every list in Q is sorted*

a) Prove this loop invarian using induction.

- *base case:* After 0 iterations, Q is a Queue of single-element lists, so each list is naturally sorted.

- *Induction Step:* Assume that the loop invariant is true after t iterations. Show that after the t + 1 iteration it is still true.

Take 2 lists from Q (sorted by induction), and merge + sort them in MergeSort, then requeue them. The new list in the queue is thus sorted. All other lists (sorted by induction) are untouched. Thus after (t + 1) iterations, all lists are sorted.

b) Use the loop invariant to show that the algorithm is correct.

After each iteration, the Queue shrinks by one (2 lists are pulled out, and sorted and put back in as 1)

thus, after n-1 iterations, there is only 1 list left in the queue, and by the loop invariant, it must be sorted.

c) Use the runtime method we learned in class to show that this algorithm runs in $O(n^2)$ time.

Outer loop runs in O(n), merge sort runs, at worst, in O(n)

so by the product rule, the algorithm is upper bounded by $O(n^2) = O(n * n)$

d) Show that $O(n^2)$ is not a tight bound by doing a more careful analysis

$$
\begin{aligned}
T(n) &\leq \sum_{k=1}^{\lceil log(n) \rceil} \frac{n}{2^k} \cdot 2^k && (13) \\
&= \sum_{k=1}^{\lceil log(n) \rceil} n && (14) \\
&= n \cdot \lceil log(n) \rceil && (15) \\
&\implies \lim_{n \to \infty} \frac{n \cdot \lceil log(n) \rceil}{n^2} = 0 && (16)
\end{aligned}
$$

Thus $O(n^2)$ is not a tight bound ($T(n) \notin \Theta(n^2)$), since it can be shown $\lim_{n \to \infty} \frac{T(n)}{n^2} = 0$

Justification:

- $\frac{n}{2^k}$: Max number of occurrences of merges with 2^k elements during Queue Sort operations.

  Consider the example with $n = 7$. There are $3 < 3.5 = 7/2^1$ merges with 2 elements. There are 2 merges with 3 or 4 elements. There is 1 merge with 7 elements.

  Likewise, when $n = 8$, there are 4 merges with 2 elements, 2 merges with 4 elements, and 1 merge with 8 elements.

- 2^k : Run time for merge sort with $i + j = 2^k$ elements

4. **Given an integer $x \geq 0$ this algorithm returns the value x^2**

a) Prove Squared correctly returns x^2 for any input $x \geq 0$

Base Case: show that 0 is correctly squared
$0^2 = 0$

Induction Step: Assume Squared(t - 1) is correct for some $t \geq 1$. Show that Squared(t) is also correct.

Case 1: t is odd:

$$\begin{aligned} (t)^2 &= (t^2 - 2t + 1) + 2t - 1 && (17) \\ &= (t-1)^2 + 2t - 1 && (18) \\ &= Squared(t-1) + 2t - 1 && (19) \end{aligned}$$

Thus when t is odd, the algorithm correctly returns t^2

Case 2: t is even:
note t = 2k for some $k \in \mathbb{Z}$

$$\begin{aligned} (t)^2 &= (2k)^2 && (20) \\ &= 4k^2 = 4\left(\frac{t}{2}\right)^2 && (21) \\ &= 4 \cdot Squared\left(\frac{t}{2}\right) && (22) \end{aligned}$$

Thus when t is even, the algorithm correctly returns t^2

Thus for all $t \geq 0$, the algorithm correctly returns t^2

b) Assuming that $x = 2^k$ for some integer $k \geq 0$. In terms of k, how many recursive calls are neccessary to reduce x down to 0 (base case)

When the input is even, the next recursive value of x, x_i, $= x_{i+1}$. Because the first input, 2^k is even, after each recursive call x_i will continue to be even:

$$2^k \textit{ is even}$$

$$2^{k-1} \textit{ is even}$$

$$\vdots$$

After k + 1 iterations, $x_k = 2^{k-k} = 2^0 = 1$. Once $x_i = 1$, which is odd, the algorithm computes $x_{i+1} = x_i - 1 = 0$

Thus the algorithm takes k + 2 executions to get to 0. (k + 1 recursive calls)

c) Assuming that $x = 2^k - 1$ for some integer $k \geq 0$. In terms of k, how many recursive calls are neccessary to reduce x down to 0 (base case)

$x_0 = 2^k - 1$ is odd, thus the first recursive call gets value $x_1 = 2^k - 1 - 1$, which is even. Because x_1 is even, $x_2 = \frac{2^k - 2}{2} = 2^{k-1} - 1$. This restarts the problem with $x = 2^{k-1} - 1$

This process repeats k times until $x_{2k} = 2^{k-k} - 1 = 0$

Thus the algorithm takes 2k + 1 executions to get to 0. (2k recursive calls)