Office Hours Practice Problems — Solutions

SCIS 313: Data Structures and Algorithm Analysis

Prof. Antonio Khalil Moretti Spelman College

Problem 1 — Unrolling (division recurrence)

Step 1 (substitute $n/3$ for n, then plug back in):

$$\begin{aligned} T(n) &= 3T(n/3) + 2n \\ &= 3\big[3T(n/9) + 2(n/3)\big] + 2n = 9T(n/9) + 2n + 2n \end{aligned}$$

Step 2 (substitute $n/9$ for n, then plug back in):

$$= 9\big[3T(n/27) + 2(n/9)\big] + 2n + 2n = 27T(n/27) + 2n + 2n + 2n$$

Step 3:

$$= 81T(n/81) + 2n + 2n + 2n + 2n$$

Pattern after k steps:

$$T(n) = 3^k T(n/3^k) + 2n \cdot k$$

Note: at each level the outside term is always $2n$ (not $2 \cdot n/3^i$), because $3^i \cdot 2(n/3^i) = 2n$ — the 3^i and the $1/3^i$ cancel exactly. This is what happens in Case 1: equal work at every level.

Stopping condition: $n/3^k = 1 \Rightarrow k = \log_3 n$.

$$T(n) = 3^{\log_3 n} \cdot T(1) + 2n \log_3 n = n \cdot 1 + 2n \log_3 n = \Theta(n \log n)$$

Verify with theorem: $a = 3,\ b = 3,\ d = 1.$ $b^d = 3 = a$ **(Case 1).** $T(n) = O(n^d \log n) = O(n \log n)$. ✓

Problem 2 — Unrolling (subtraction recurrence)

Step 1 (substitute $n - 1$ for n, then plug back in):

$$\begin{aligned} T(n) &= 4T(n-1) + 1 \\ &= 4\big[4T(n-2) + 1\big] + 1 = 16T(n-2) + 4 + 1 \end{aligned}$$

Step 2:

$$= 16\big[4T(n-3) + 1\big] + 4 + 1 = 64T(n-3) + 16 + 4 + 1$$

Step 3:

$$= 256T(n-4) + 64 + 16 + 4 + 1$$

Pattern after k steps:

$$T(n) = 4^k T(n-k) + \sum_{j=0}^{k-1} 4^j = 4^k T(n-k) + \frac{4^k - 1}{4 - 1}$$

Common mistake to watch for: the coefficient on T is 4^k (doubling… quadrupling each time), while the sum of the outside terms is a geometric series with ratio 4, *not* ratio 1/4. This is growing, not shrinking — a signal that the recursion will dominate.

Stopping condition: $n - k = 1 \Rightarrow k = n - 1$.

$$T(n) = 4^{n-1} T(1) + \frac{4^{n-1} - 1}{3} = \Theta(4^n)$$

The Three-Case Recurrence Theorem does **not** apply here because the subproblem decreases by subtraction $(n-1)$, not division. The result is exponential, as always happens when $a > 1$ in a subtraction recurrence.

Problem 3 — Three-Case Recurrence Theorem

All three have $a = 9$, $b = 3$, so $\log_b a = \log_3 9 = 2$.

(a) $T(n) = 9T(n/3) + O(n)$: $d = 1,\ b^d = 3,\ a = 9 > 3$ **(Case 3).**

$$T(n) = O(n^{\log_3 9}) = O(n^2)$$

(b) $T(n) = 9T(n/3) + O(n^2)$: $d = 2,\ b^d = 9,\ a = 9 = 9$ **(Case 1).**

$$T(n) = O(n^2 \log n)$$

(c) $T(n) = 9T(n/3) + O(n^3)$: $d = 3,\ b^d = 27,\ a = 9 < 27$ **(Case 2).**

$$T(n) = O(n^3)$$

Key observation: the recursion structure ($a = 9$, $b = 3$) is identical in all three — only the combine cost changes. In (a) the recursion dominates; in (b) they are balanced and we pay an extra $\log n$; in (c) the combine step is so expensive it drowns out the recursion entirely. This illustrates that you cannot determine the complexity from the recursion alone — the combine cost is equally important.

Problem 4 — Writing a recurrence from code

(a) Three recursive calls on $n/2$; the double loop runs n^2 times.

$$T(1) = c \qquad T(n) = 3T(n/2) + O(n^2)$$

$a = 3,\ b = 2,\ d = 2$.

(b) $b^d = 2^2 = 4$. $a = 3 < 4 = b^d$ **Case 2.**

$$T(n) = O(n^d) = O(n^2)$$

The quadratic double loop at the root dominates; the three-way recursion into halves is not branchy enough to keep up with the shrinking subproblem sizes.

(c) With a single loop the outside work is $O(n)$, so $d = 1$:

$$T(n) = 3T(n/2) + O(n)$$

$b^d = 2^1 = 2$. $a = 3 > 2 = b^d$ **Case 3.**

$$T(n) = O(n^{\log_2 3}) \approx O(n^{1.585})$$

Yes, the complexity changes — and in an interesting direction. Reducing the combine cost from $O(n^2)$ to $O(n)$ actually *worsens* the overall bound (from $O(n^2)$ to $O(n^{1.585})$... wait,

$n^{1.585} < n^2$, so it actually *improves*!). This is a good reminder: a cheaper combine step does not automatically mean a better overall runtime, because it can shift which term dominates. Here it tips us from Case 2 into Case 3, where the recursion takes over.