Problem 1A (Recurrence Relations — 3 Styles, 9 pts)

Use induction to show the asymptotic run-time corresponding to the following recurrence relation is $O(\log^2(n))$. **Do not use the Master Theorem here.**

$$T(0) = T(1) = 1. \text{ For all } n \geq 2, T(n) = T(\lceil \tfrac{n}{2} \rceil - 1) + \log n.$$

Solution

We will use induction to show that $T(n) = O((\log n)^2)$.

Base Case: When $n = 2$, for any $c \geq 2$,we have

$$1 + \log 2 \leq c \cdot \log^2(2)$$
$$2 \leq c \cdot 1$$

Inductive Hypothesis: Assume that for all $k < n$ there exists a constant $C > \frac{1}{2}$ such that:

$$T(k) \leq C(\log k)^2.$$

Inductive Step: For $n \geq 2$, we have:

$$T(n) = T\Big(\Big\lceil \frac{n}{2} \Big\rceil - 1\Big) + \log n.$$

Since:

$$\Big\lceil \frac{n}{2} \Big\rceil - 1 \leq \frac{n}{2},$$

the inductive hypothesis implies:

$$T(n) \leq C \left(\log \frac{n}{2}\right)^2 + \log n.$$

since $\log \frac{n}{2} = \log n - 1$, we have:

$$T(n) \leq C(\log n - 1)^2 + \log n.$$

Expanding the inequality, we have:

$$T(n) \leq C\left[(\log n)^2 - 2\log n + 1\right] + \log n = C(\log n)^2 - 2C \log n + C + \log n.$$

and this simplifies to:

$$T(n) \leq C(\log n)^2 - (2C - 1)\log n + C.$$

For the inequality $T(n) \leq C(\log n)^2$ to hold, it is sufficient that:

$$-(2C - 1)\log n + C \leq 0.$$

This inequality is satisfied for all sufficiently large n provided:

$$2C - 1 > 0 \quad \Longleftrightarrow \quad C > \frac{1}{2}.$$

Thus, by choosing any constant $C > \frac{1}{2}$, the inductive step is verified.

Conclusion: By mathematical induction, for all sufficiently large n,

$$T(n) \leq C(\log n)^2.$$

Hence, we conclude that $T(n) = O((\log n)^2)$.

□

Problem 1B (9 pts)

Use a recursion tree to find the asymptotic run-time corresponding to the following recurrence relation. You may assume any fractional input to T is the greatest integer less than it (e.g., $T(\frac{2n}{3}) = T(\lfloor \frac{2n}{3} \rfloor)$).

$$T(0) = T(1) = 1. \text{ For all } n \geq 2,\ T(n) = T(\tfrac{2n}{3}) + T(\tfrac{n}{5}) + n$$

Solution

Given the recurrence, we will first draw the recursion tree, then use it to sum all the costs to reach the conclusion.

Step 1: Constructing the Recursion Tree

- **Level 0 (Root):** The cost is n.
- **Level 1:** The recurrence splits into two subproblems of sizes $\frac{2n}{3}$ and $\frac{n}{5}$. Therefore, the cost at this level is:
$$\frac{2n}{3} + \frac{n}{5} = \frac{13n}{15}.$$
- **Level i:** At any node with subproblem size x, the work done is proportional to x. Since each node produces subproblems of sizes $\frac{2}{3}x$ and $\frac{1}{5}x$, the sum of the subproblem sizes is:
$$\frac{2}{3}x + \frac{1}{5}x = \frac{13}{15}x.$$
Hence, if the total cost at level i is $n\left(\frac{13}{15}\right)^i$, then the total cost at level $i+1$ is:
$$n\left(\frac{13}{15}\right)^{i+1}.$$

Step 2: Summing the Costs

The total work is the sum over all levels until the subproblems become constant (which happens after $\Theta(\log n)$ levels):

$$T(n) = n + \frac{13n}{15} + \left(\frac{13}{15}\right)^2 n + \cdots.$$

This is a geometric series with common ratio $\frac{13}{15} < 1$. Thus, the sum is:

$$T(n) \leq n \sum_{i=0}^{\infty} \left(\frac{13}{15}\right)^i = n \cdot \frac{1}{1 - \frac{13}{15}} = n \cdot \frac{1}{\frac{2}{15}} = \frac{15}{2}n.$$

Therefore,

$$T(n) = \Theta(n).$$

Problem 1C (12 pts)

Find the asymptotic run-time corresponding to each of the following recurrence relations using the Master Theorem. For each, explain which case of the Master Theorem applies and why.

(a) $T(n) = 2T(\frac{n}{5}) + \sqrt{2n}$.

(b) $T(n) = 3T(\frac{n}{3}) + n/2$.

(c) $T(n) = 4T(\frac{n}{2}) + n \log n$.

Solution

(a) Consider the recurrence

$$T(n) = 2T\left(\frac{n}{5}\right) + \sqrt{2n}.$$

We have:

$$a = 2, \quad b = 5, \quad f(n) = \sqrt{2n} = \Theta(\sqrt{n}).$$

$$n^{\log_b a} = n^{\log_5 2} = \Theta\left(n^{\log_5 2}\right).$$

Since $\log_5 2 \approx 0.4307$,

$$n^{\log_5 2} = \Theta\left(n^{0.4307}\right).$$

Comparing $f(n) = \Theta\left(n^{0.5}\right)$ with $n^{0.4307}$, we see that

$$f(n) = \Omega\left(n^{\log_5 2+\epsilon}\right) \quad \text{for } \epsilon \approx 0.0693 > 0.$$

Moreover, the regularity condition holds since

$$a\, f\left(\frac{n}{b}\right) = 2\,\Theta\left(\sqrt{\frac{n}{5}}\right) = \Theta\left(\sqrt{n}\right)$$

with a constant factor $2/\sqrt{5} < 1$. Hence, by Case 3 of the Master Theorem,

$$T(n) = \Theta\left(f(n)\right) = \Theta\left(\sqrt{n}\right).$$

(b) Consider the recurrence

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

We have:

$$a = 3, \quad b = 3, \quad f(n) = \frac{n}{2} = \Theta(n).$$

$$n^{\log_b a} = n^{\log_3 3} = n.$$

Since $f(n) = \Theta(n) = \Theta\left(n^{\log_3 3}\right)$, this corresponds to Case 2 of the Master Theorem. Thus,

$$T(n) = \Theta(n \log n).$$

(c) Consider the recurrence

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

We have:

$$a = 4, \quad b = 2, \quad f(n) = n\log n.$$
$$n^{\log_b a} = n^{\log_2 4} = n^2.$$

Since $n\log n = o(n^2)$, we can write $f(n) = O\big(n^{2-\epsilon}\big)$ for some $0 < \epsilon < 1$ (for instance, $\epsilon = \frac{1}{2}$). Therefore, by Case 1 of the Master Theorem,

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

Problem 2 (Array Search, 30 pts)

You are given an array of n integers $a_1 < a_2 < \cdots < a_n$. Give an $O(\log n)$ algorithm that outputs an index i where $a_i = i$, or outputs $\perp$ if such i does not exist. Justify the correctness and time complexity of your proposed algorithm.

Solution

Define the function

$$f(i) = a_i - i.$$

Since the array is strictly increasing, $f(i)$ is non-decreasing. We then perform a binary search for an index i with $f(i) = 0$ (i.e., $a_i = i$).

FindFixedPoint(A, n)

```
1: L ← 1
2: R ← n
3: while L ≤ R do
4:    mid ← ⌊(L + R)/2⌋
5:    if A[mid] = mid then
6:       return mid
7:    else if A[mid] > mid then
8:       R ← mid − 1
9:    else
10:      L ← mid + 1
11:   end if
12: end while
13: return ⊥
```

Now we justify the correctness and time complexity of my proposed FindFixedPoint algorithm.

Correctness: Since $f(i) = a_i - i$ is non-decreasing, if $f(mid) < 0$ then for all $i < mid$ we have $f(i) \leq f(mid) < 0$. Similarly, if $f(mid) > 0$ then for all $i > mid$, $f(i) \geq f(mid) > 0$. Therefore, if there is an index i such that $f(i) = 0$, the binary search will locate it.

Time Complexity: Each iteration of the loop reduces the size of the search interval by at least half. Since the initial interval is of size n, the number of iterations is $O(\log n)$. Each iteration does a constant amount of work, so the overall time complexity is $O(\log n)$.

Problem 3 (Malfunctioning Phones, 40 pts)

A manufacturer has a recall on a set of n cell phones, some of which have a malfunction which makes them unreliable. The manufacturer has built a machine that allows a pair of phones to test each other's correctness. Let C_1, C_2 be a pair of phones. The machine M runs in the following way:

1. $M(C_1, C_2) = 11$ if both phones say the other is working.
2. $M(C_1, C_2) = 10$ if one phone says the other is working and one phone says the other is malfunctioning.
3. $M(C_1, C_2) = 00$ if both phones say the other is malfunctioning.

Remember that malfunctioning phones cannot be trusted, so they may lie, tell the truth, or throw out a random response. Working phones, on the other hand, can be assumed to know if the other phone is working or malfunctioning always.

(a) Show that if you know at least one working phone, all other working phones can be found by using $O(n)$ queries to M.

(b) Assume the majority of the phones are working, i.e., there are greater than $n/2$ working phones. Give an algorithm that can find a working phone in $O(n)$ queries to M. Justify the correctness and time complexity of your proposed algorithm. [Hint: Start by explaining how to use $O(n)$ queries to reduce the problem size by a constant factor.]

(c) Assume the majority of the phones are malfunctioning, i.e., there are fewer than $n/2$ working phones. Is there still a procedure (using M) that is guaranteed to find a working phone? Give a brief justification.

Solution

(a)

Suppose we know phone W is working. Since working phones always report correctly, we can determine whether any other phone X is working by querying the machine $M(W, X)$. In particular, when W tests X:

- If W reports that X is working (i.e. the output is either 11 or, in a mixed response, the trusted answer from W is that X is working), then X must be working.
- If W reports that X is malfunctioning, then X is malfunctioning.

Thus, by making one query for each of the other $n - 1$ phones, we can identify all working phones using $O(n)$ queries.

(b)

Assume that more than $\frac{n}{2}$ phones are working. We now describe an algorithm that finds a working phone using $O(n)$ queries to the machine M:

Algorithm

1. **Pairing:** Partition the n phones arbitrarily into pairs. If n is odd, leave one phone unpaired.
2. **Testing:** For each pair (A, B), perform a query $M(A, B)$.
   - If $M(A, B) = 11$, then *at least one* of A or B is working. Retain *one* of these (say, A) as a candidate.
   - If $M(A, B) = 10$ or 00, discard both A and B.
3. **Iteration:** Replace the current set of phones with the set of retained candidates and repeat the pairing and testing process until a single candidate remains. Declare this candidate as working.

Correctness and Detailed Justification

Majority Preservation Let W and L denote the number of working and malfunctioning phones, respectively. Initially, we have

$$W > L.$$

In each round of pairing, consider two types of pairs:

- **Pairs with outcome** 10 **or** 00**:** In a pair with one working and one malfunctioning phone, the working phone correctly identifies the malfunctioning one, resulting in an outcome 10 (or 00 if the malfunctioning phone lies). In this case, both phones are discarded. Thus, one working phone and one malfunctioning phone are eliminated.

  In a pair of two malfunctioning phones, both are eliminated.

  Let C be the total number of working phones eliminated and D the total number of malfunctioning phones eliminated in this round. In all such pairs, it holds that

  $$C \leq D.$$

  Applying the hint (if $A > B$ and $C \leq B$, then $A - C > B - D$) with $A = W$ and $B = L$, we deduce that

  $$W - C > L - D.$$

  That is, the remaining candidate set still has a strict majority of working phones.

- **Pairs with outcome** 11**:** In a pair that produces $M(A, B) = 11$, at least one phone is working. We retain one candidate from such a pair. Note that it is possible that the pair consists of two malfunctioning phones that collude to output 11. However, even if this happens, the effect is simply that one malfunctioning phone is carried forward into the candidate set.

Why Pairs of Two Malfunctioning Phones (Yielding 11**) Do Not Disrupt the Process.** Even when two malfunctioning phones are paired and collude to produce $M(A, B) = 11$, the algorithm retains one candidate from the pair. Although this candidate is malfunctioning, the overall elimination process in the rounds where outcomes 10 or 00 occur guarantees that the number of working phones eliminated is never more than the number of malfunctioning ones eliminated. By the professor's hint, subtracting a smaller (or equal) number from the initial majority ($W > L$) leaves us with a positive difference:

$$W - C > L - D.$$

Thus, even if some pairs of malfunctioning phones sneak into the candidate set by yielding 11, the net majority of working phones is preserved across rounds. When the process finally reduces the candidate set to a single phone, the preserved majority guarantees that this candidate is working.

Time Complexity

In the first round, we make at most $\lfloor n/2 \rfloor$ queries. In each subsequent round, the number of candidates is reduced by a constant factor. The total number of queries is bounded by:

$$\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots \leq n,$$

which is $O(n)$.

(c)

When fewer than $n/2$ phones are working, the working phones are in the minority. In this case, no procedure using the machine M is guaranteed to find a working phone. The reason is that malfunctioning phones can provide arbitrary, even adversarial, responses. They may collude to mimic the responses of working phones (e.g., always outputting 11) and thus hide the existence of any working phone. Without a reliable majority, any elimination strategy can be subverted by the faulty phones' unpredictable behavior.