Homework 3 (50 points) Due: September 27, 2024 11:59 pm

COMPSCI 733: Advanced Algorithms and Designs

Benzon Carlitos Salazar (salazarbc24@uww.edu)

Documentation: (5 points) Type your solutions using Latex (www.overleaf.com or https://www.latex-project.org/). Submit your solutions (pdf is enough) to Canvas.

Problem 1: (15 points)

(a) Arrange the following functions into increasing order of time complexity; that is, $f(n)$ should come before $g(n)$ in your list if $f(n)$ is $O(g(n))$. (i.e, n comes before n^2 since n is $O(n^2)$.)

$$100000,\ \log(n),\ 2^n,\ n\log(n),\ n^{0.9999999}\log(n),\ n+\log(n),\ n^{50}-100n^2,$$

$$n^n,\ n!,\ 1.0000001^n,\ (\log(n))^n$$

Answer:

$$100000 < \log(n) < n+\log(n) < n^{0.9999999}\log(n) < n\log(n) < 1.0000001^n$$

$$< 2^n < (\log(n))^n < n! < n^n < n^{50}-100n^2$$

(b) Use the master theorem (the version given on the slides, not the CLRS textbook version)to give a tight bound (i.e., $(\Theta()))$ for the following recurrence relations. Write your steps using the format.

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$.

- Identify $a, b, f(n)$.
- Calculate $n^{\log_b a}$
- Compare $n^{\log_b a}$ with $f(n)$ and identify a case from the theorem.
- For Case 3 only: Check the regularity condition. i.e., Find $c < 1$ such that $af(n/b) \leq cf(n)$.
- Conclusion: The Θ bound or "The theorem cannot be applied."

(a) $T(n) = 2T(\frac{n}{4}) + 1$

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

(c) $T(n) = 2T(\frac{n}{2}) + n^2$

(d) $T(n) = 4T(\frac{n}{2}) + n^2 \log n$

Answer:

(a) $T(n) = 2T\left(\frac{n}{4}\right) + 1$

- Identify $a = 2$, $b = 4$, $f(n) = 1$
- Calculate $n^{\log_b a} = n^{\log_4 2} = n^{1/2}$
- Compare $n^{\log_b a}$ with $f(n)$. Since $f(n) = O(n^{\log_4 2-\epsilon})$ for some $\epsilon > 0$, this is Case 1.
- Conclusion: $T(n) = \Theta(n^{1/2})$

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

- Identify $a = 2$, $b = 4$, $f(n) = \sqrt{n} = n^{1/2}$
- Calculate $n^{\log_b a} = n^{\log_4 2} = n^{1/2}$
- Compare $n^{\log_b a}$ with $f(n)$. Since $f(n) = \Theta(n^{1/2})$, this is Case 2.
- Conclusion: $T(n) = \Theta(n^{1/2} \log n)$

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

- Identify $a = 2$, $b = 2$, $f(n) = n^2$
- Calculate $n^{\log_b a} = n^{\log_2 2} = n^1$
- Compare $n^{\log_b a}$ with $f(n)$. Since $f(n) = \Omega(n^{\log_2 2+\epsilon})$ for $\epsilon = 1$, we check the regularity condition: $af\left(\frac{n}{2}\right) = 2\left(\frac{n}{2}\right)^2 = \frac{n^2}{2}$, which satisfies the condition with $c = \frac{1}{2}$. This is Case 3.
- Conclusion: $T(n) = \Theta(n^2)$

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

- Identify $a = 4$, $b = 2$, $f(n) = n^2 \log n$
- Calculate $n^{\log_b a} = n^{\log_2 4} = n^2$
- Compare $n^{\log_b a}$ with $f(n)$. Since $f(n) = \Theta(n^2 \log n)$, this is Case 2.
- Conclusion: $T(n) = \Theta(n^2 \log^2 n)$

Problem 2: (15 points) An element x of an array $A[1 : n]$ is called a majority element if more than half of A are equal to x. Your task is to find whether a given array has a majority element and if so, what it is. We do not assume the elements of A are integers, or otherwise ordered; the only query you can access the elements of A is of the form whether $A[i] = A[j]$ in constant time, for any i and j.

(a) Suppose that A has a majority element, x. Let L and R partition A; i.e., L and R are disjoint subsets of A such that their union is A. Use proof by contradiction to prove that if x is a majority element in A, then it must also be a majority element in L or in R.

Proof. (By contradiction)

Suppose x is a majority element in A, meaning that more than half of the elements in A are equal to x. Therefore, the number of occurrences of x in A, denoted as $\text{count}_A(x)$, satisfies:

$$\text{count}_A(x) > \frac{n}{2}.$$

Assume, for contradiction, that x is not a majority element in either L or R.

Let L and R be disjoint subsets of A such that their union is A, and let the sizes of L and R be n_L and n_R, respectively, where:

$$n_L + n_R = n.$$

Since x is not a majority element in L, the number of occurrences of x in L, denoted as $\text{count}_L(x)$, satisfies:

$$\text{count}_L(x) < \frac{n_L}{2}.$$

Similarly, since x is not a majority element in R, the number of occurrences of x in R, denoted as $\text{count}_R(x)$, satisfies:

$$\text{count}_R(x) < \frac{n_R}{2}.$$

Now, the total number of occurrences of x in A is the sum of its occurrences in L and R:

$$\text{count}_A(x) = \text{count}_L(x) + \text{count}_R(x).$$

Using the bounds on $\text{count}_L(x)$ and $\text{count}_R(x)$, we have:

$$\text{count}_A(x) < \frac{n_L}{2} + \frac{n_R}{2} = \frac{n_L + n_R}{2} = \frac{n}{2}.$$

But this contradicts the fact that $\text{count}_A(x) > \frac{n}{2}$ because x is a majority element in A.

Therefore, our assumption that x is not a majority element in either L or R must be false. Hence, if x is a majority element in A, then x must be a majority element in at least one of L or R. This completes the proof by contradiction.

□

(b) Solve this problem with a Divide-and-Conquer algorithm that runs in time $O(n \log n)$. Write a pseudo code of your algorithm. (Hint: By dividing A into two halves A_1 and A_2, and using Part (a). You can use English statements to explain the steps of your pseudo code. Direct copying of online versions is not accepted.

Algorithm 1 Majority(A)

```
Input: Array $A[1:n]$
Output: Majority element of $A$, or None if no majority exists

if $n == 1$ then
    return $A[1]$                                  ▷ Base case: Only one element
end if

$m = \lfloor n/2 \rfloor$                          ▷ Find the middle index to divide the array
$majority_1 = Majority(A[1:m])$                    ▷ Recursive call on left half
$majority_2 = Majority(A[m+1:n])$                  ▷ Recursive call on right half

if $majority_1 == majority_2$ then
    return $majority_1$                            ▷ Both halves agree on the majority element
end if

$count_1 = \text{Count}(A, majority_1)$            ▷ Count occurrences of $majority_1$ in the full array
$count_2 = \text{Count}(A, majority_2)$            ▷ Count occurrences of $majority_2$ in the full array

if $count_1 > n/2$ then
    return $majority_1$                            ▷ Return if $majority_1$ is a valid majority in the full array
else if $count_2 > n/2$ then
    return $majority_2$                            ▷ Return if $majority_2$ is a valid majority in the full array
else
    return None                                    ▷ No majority element exists in the array
end if
```

(c) Write a recurrence relation for the execution time, $T(n)$, and apply Master's theorem to prove Θ time complexity of $T(n)$.

Answer:

The recurrence relation for the execution time $T(n)$ is:

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

This is a recurrence of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, where $a = 2$, $b = 2$, and $f(n) = O(n)$.

Using the Master Theorem:

$$n^{\log_b a} = n^{\log_2 2} = n.$$

Since $f(n) = O(n)$ matches $n^{\log_b a}$, this falls under Case 2 of the Master Theorem. Therefore, the solution to the recurrence is:

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

Problem 3: (15 points)

The Towers of Hanoi puzzle has 3 towers labeled 0, 1, and 2. There are n disks of sizes 1 to n, initially placed on tower 0 from top to bottom in the order of their sizes. The goal is to move all n disks from tower 0 to tower 2 under certain rules. One can only move one disk at a time, and move the top disk on a tower to be placed on the top of another tower, and cannot place a larger disk on top of a smaller disk. The question is, what is the smallest number of moves needed. Our problem is a variant of the Towers of Hanoi puzzle: The three towers are still labeled 0, 1, and 2. There is one additional constraint: You are only allowed to move disks from a tower labeled i to the next one labeled $(i + 1) \bmod 3$. You can imagine the towers as being arranged in a circular fashion. So, for example, in order to move a disk from tower 0 to tower 2, you would have to first move it to tower 1 (if both $0 \to 1$ and $1 \to 2$ are legal moves at this point), and this sequence would cost two moves instead of just one.

(a) Explain that starting at any legal configuration, there is at least one legal move. (Any legal configuration means several legal moves may have already been made. Any tower may be empty or non-empty.)

Answer: In the Towers of Hanoi problem, the towers are arranged in a circular manner (0, 1, 2), and disks can only be moved from tower i to tower $(i + 1) \mod 3$. Additionally, disks can only be moved if the disk being moved is smaller than the disk it is placed on top of (or if the destination tower is empty).

- **At least one non-empty tower:** Since the problem starts with all disks on tower 0 and disks are moved one at a time, at least one tower will always contain disks (until the puzzle is fully solved, when all disks are on tower 2).
- **The smallest disk is always movable:** A key property of the Towers of Hanoi problem (both the standard version and this variant) is that the smallest disk can always be moved. There is no restriction on placing smaller disks on top of larger ones. Hence, if a tower contains the smallest disk, it can always be moved to an adjacent tower if the rules allow.
- **Valid moves are always possible:** In any legal configuration, no larger disk can be placed on top of a smaller disk. This ensures that at least one top disk from a non-empty tower can always be legally moved to the next tower in the circular sequence, either because the destination tower is empty or because the top disk on the destination tower is larger than the disk being moved.
- **Circular nature of the problem:** The towers are arranged in a circular fashion, so even if a direct move from one tower to another is not possible (e.g., from tower 0 to tower 2), the move can still be made indirectly by moving the disk first to tower 1. This ensures that there is always a legal sequence of moves that can be performed.

Therefore, At any legal configuration, at least one tower will have a disk that can be legally moved to an adjacent tower (under the mod 3 rule), meaning there is always at least one legal move available.

(b) Inductively prove that for any $n \geq 1$, this variant of the Towers of Hanoi puzzle has a solution consisting of finitely many legal moves.

$P(n)$: This variant of the Towers of Hanoi puzzle with n disks has a solution consisting of finitely many legal moves.

Prove: $P(n)$ is true for all $n \geq 1$.

Proof. (By induction on n)

Let $P(n)$ be the statement that this variant of the Towers of Hanoi puzzle with n disks has a solution consisting of finitely many legal moves.

- **Base Case:**
  For $n = 1$, we have a single disk. We can move it from tower 0 to tower 1, and then from tower 1 to tower 2, in two legal moves. Thus, $P(1)$ is true.
- **Inductive Hypothesis:**
  Assume that for some $k \geq 1$, the variant of the Towers of Hanoi puzzle with k disks can be solved with finitely many legal moves. This means that starting with k disks on tower 0, we can move all k disks to tower 2 using a finite sequence of moves that adheres to the circular move constraint.
- **Inductive Step:**
  We need to show that $P(k+1)$ is true, i.e., that the puzzle with $k+1$ disks can also be solved in finitely many legal moves.
  - Move the top k disks from tower 0 to tower 1. By the inductive hypothesis, this can be done in a finite number of moves.
  - Move the $k+1$-th (largest) disk from tower 0 to tower 2 in one legal move.
  - Move the k disks from tower 1 to tower 2. Again, by the inductive hypothesis, this can be done in a finite number of moves.

  Therefore, the puzzle with $k+1$ disks can be solved in a finite number of moves.
- **Conclusion:**
  By the principle of mathematical induction, $P(n)$ is true for all $n \geq 1$. Thus, this variant of the Towers of Hanoi puzzle can be solved with finitely many legal moves for any number of disks.

□

(c) Follow the given steps to show a recurrence of $T(n)$, the number of moves needed for the variant of the Towers of Hanoi puzzle with n disks, satisfies:

$T(0) = 0, T(1) = 2, T(n) = 2T(n-1) + 2T(n-2) + 3.$

(a) Introduce a quantity $S(n)$ for the number of moves needed to move n disks from tower 0 to tower 1 (or i to $(i+1) \mod 3$). First, we show that:

$$S(n) = 2T(n-1) + 1.$$

To move n disks from tower 0 to tower 1:

* First, move the top $n-1$ disks from tower 0 to tower 2, which takes $T(n-1)$ moves.
* Then, move the n-th disk from tower 0 to tower 1, which takes 1 move.
* Finally, move the $n-1$ disks from tower 2 to tower 1, which takes $T(n-1)$ moves.

Therefore, the total number of moves is:

$$S(n) = 2T(n-1) + 1.$$

(b) Derive $T(n)$ using $T(n-1)$ and $S(n-1)$. To move n disks from tower 0 to tower 2:

* First, move the top $n-1$ disks from tower 0 to tower 1. This takes $S(n-1) = 2T(n-2) + 1$ moves.
* Then, move the largest disk from tower 0 to tower 2. This takes 1 move.
* Finally, move the $n-1$ disks from tower 1 to tower 2, which takes $T(n-1)$ moves.

Therefore, the total number of moves is:

$$T(n) = S(n-1) + 1 + T(n-1).$$

Substituting $S(n-1) = 2T(n-2) + 1$ into the equation, we get:

$$T(n) = (2T(n-2) + 1) + 1 + T(n-1) = 2T(n-2) + T(n-1) + 2.$$

(c) Use Part (a) to derive $T(n)$ using $T(n-1)$ and $T(n-2)$. From Part (b), we derived the recurrence relation:

$$T(n) = 2T(n-2) + T(n-1) + 2.$$