CS 5720 Design and Analysis of Algorithms
Homework #4

Raja Kantheti

October 8, 2024

Question 1: Solving Recurrence Relations Using Master Theorem

(a) $T(n) = 5T(n/3) + n$

- This recurrence fits the form $T(n) = aT(n/b) + f(n)$, where $a = 5$, $b = 3$, and $f(n) = n$.
- Calculate $\log_b a = \log_3 5 \approx 1.46497$.
- Compare $f(n) = n$ with $n^{\log_b a}$.
- Since $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$ (specifically, n^1 vs. $n^{1.46497}$), by the Master Theorem (Case 1):
- $T(n) = \Theta(n^{\log_3 5}) \approx \Theta(n^{1.46497})$

(b) $T(n) = 2.7T(n/5) + n^2$

- This recurrence fits the form $T(n) = aT(n/b) + f(n)$, where $a = 2.7$, $b = 5$, and $f(n) = n^2$.
- Calculate $\log_b a = \log_5 2.7 \approx 0.58975$.
- Compare $f(n) = n^2$ with $n^{\log_b a}$.
- Since $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ (specifically, n^2 vs. $n^{0.58975}$), by the Master Theorem (Case 3):
- $T(n) = \Theta(n^2)$

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

- This recurrence does not fit the Master Theorem directly because $T(n)$ is based on $T(n-1)$, not $T(n/b)$.
- However, this can be solved by iteration or substitution. This is a linear homogeneous recurrence with additional term:
- $T(n) = 2T(n-1) + n$

- Using iteration:

$$T(n) = 2(2T(n-2) + (n-1)) + n = 4T(n-2) + 2(n-1) + n$$
$$T(n) = 4(2T(n-3) + (n-2)) + 2(n-1) + n = 8T(n-3) + 4(n-2) + 2(n-1) + n$$

- Continuing this pattern and summing the series, we get:
- $T(n) = 2^n T(0) + \sum_{k=0}^{n-1} 2^k k$
- $T(n) = 2^n + n \cdot 2^n$
- The dominant term is 2^n, so:
- $T(n) = \Theta(2^n)$

(d) $T(n) = 1.1T(0.2n) + 1$

- This recurrence fits the form $T(n) = aT(n/b) + f(n)$, where $a = 1.1$, $b = 0.2$, and $f(n) = 1$.
- Calculate $\log_{0.2} 1.1$.
- Since $b < 1$, this doesn't fit the typical Master Theorem format directly, making it difficult to apply the theorem directly.

(e) $T(n) = 2T(n/2) + n \log_2 n$

- This recurrence fits the form $T(n) = aT(n/b) + f(n)$, where $a = 2$, $b = 2$, and $f(n) = n \log_2 n$.
- Calculate $\log_b a = \log_2 2 = 1$.
- Compare $f(n) = n \log_2 n$ with $n^{\log_b a} = n^1$.
- Since $f(n) = \Theta(n \log n)$, which is polynomially larger than n^1:
- $T(n) = \Theta(n \log^2 n)$

(f) $T(n) = 2T(n/2) + \sqrt{n}$

- This recurrence fits the form $T(n) = aT(n/b) + f(n)$, where $a = 2$, $b = 2$, and $f(n) = \sqrt{n}$.
- Calculate $\log_b a = \log_2 2 = 1$.
- Compare $f(n) = \sqrt{n}$ with $n^{\log_b a} = n^1$.
- Since $f(n) = O(n^{1-\epsilon})$:
- $T(n) = \Theta(n)$

(g) $T(n) = 4T(n/2) + \sqrt{n^4 - n + 10}$

- This recurrence fits the form $T(n) = aT(n/b) + f(n)$, where $a = 4$, $b = 2$, and $f(n) = \sqrt{n^4 - n + 10}$.
- Calculate $\log_b a = \log_2 4 = 2$.

- Compares $f(n) = \sqrt{n^4} = n^2$ with $n^{\log_b a} = n^2$.
- Since $f(n) = \Theta(n^2)$:
- $T(n) = \Theta(n^2 \log n)$

(h) $T(n) = 7T(n/3) + \sum_{i=1}^{n} i$

- This recurrence fits the form $T(n) = aT(n/b) + f(n)$, where $a = 7$, $b = 3$, and $f(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$.
- Calculate $\log_b a = \log_3 7 \approx 1.77124$.
- Compare $f(n) = n^2$ with $n^{\log_b a}$.
- Since $f(n) = \Theta(n^2)$, and $n^2 > n^{\log_b 7}$:
- $T(n) = \Theta(n^2)$

(i) $T(n) = 4T(n/2) + n^n$

- This recurrence cannot be solved by the Master Theorem because the function $f(n) = n^n$ is super-polynomial and does not fit the form required for the Master Theorem application.

(j) $T(n) = 8T(n/3) + n^3$

- This recurrence fits the form $T(n) = aT(n/b) + f(n)$, where $a = 8$, $b = 3$, and $f(n) = n^3$.
- Calculate $\log_b a = \log_3 8 \approx 1.89279$.
- Compare $f(n) = n^3$ with $n^{\log_b a}$.
- Since $f(n) = \Omega(n^{\log_b a+\epsilon})$:
- $T(n) = \Theta(n^3)$

Question 2: Divide-and-Conquer Algorithms

(a) Algorithm 1

Need to check this agian.

- **Worst-Case Order of Growth**:
  - $T(n) = T(n/3) + O(1)$
  - Using Master Theorem: $a = 1, b = 3, f(n) = O(1)$
  - Since $f(n) = O(n^c)$ for $c = 0$, and $c < \log_b a$ (i.e., 0 ¡ 0), case 1 applies.
  - Thus, $T(n) = \Theta(\log n)$.
- **Worst-Case Input**:

- An input where Rec1 always takes the second recursive call, causing maximum depth recursion.

- **Best-Case Complexity**:
  - The best-case occurs when the input array has only one element (n = 1), and the algorithm returns A[0] immediately.
  - then the complexity would be O(1)

- **Best-Case Input**:
  - input array has only one element (n = 1).

(b) Algorithm 2

- **Worst-Case Order of Growth**:
  - $T(n) = 2T(n/2) + O(1)$
  - Using Master Theorem: $a = 2, b = 2, f(n) = O(1)$
  - Since $f(n) = O(n^c)$ for $c = 0$, and $c < \log_b a$ (i.e., 0 ¡ 1), case 1 applies.
  - Thus, $T(n) = \Theta(n)$.

- **Worst-Case Input**:
  - Any input of length n, as the algorithm splits the array and recurses into both halves.

- **Best-Case Complexity**:
  - The best-case occurs when the input array has only one element (n = 1), and the algorithm returns A[0] immediately.
  - then the complexity would be O(1)

- **Best-Case Input**:
  - input array has only one element (n = 1).