Hw5, Algorithm

1 Q1. Recerrence Relation

1. $t(n) = 10t(n/3) + 11n$

$$\begin{aligned}
&a = 10, \quad b = 3, \quad f(n) = 11n \\
&f(n) = \Theta(n^1) \\
&a = b^1 \\
&t(n) = \Theta(n \log n)
\end{aligned}$$

2. $t(n) = 10t(n/3) + 11n$

$$\text{Same as above, } t(n) = \Theta(n \log n)$$

3. $t(n) = 27t(n/3) + 11n$

$$\begin{aligned}
&a = 27, \quad b = 3, \quad f(n) = 11n \\
&f(n) = \Theta(n^1) \\
&a = b^3 \\
&t(n) = \Theta(n^3)
\end{aligned}$$

4. $t(n) = 64t(n/4) + 10n^3 \log^2 n$

$$\begin{aligned}
&a = 64, \quad b = 4, \quad f(n) = 10n^3 \log^2 n \\
&f(n) = \Theta(n^3 \log^2 n) \\
&a = b^3 \\
&t(n) = \Theta(n^3 \log^2 n)
\end{aligned}$$

5. $t(n) = 9t(n/2) + n^2$

$$\begin{aligned}
&a = 9, \quad b = 2, \quad f(n) = n^2 \\
&f(n) = \Theta(n^2) \\
&a = b^2 \\
&t(n) = \Theta(n^2 \log n)
\end{aligned}$$

6. $t(n) = 3t(n/8) + n^2 2^n \log n$

$$\begin{aligned}
&a = 3, \quad b = 8, \quad f(n) = n^2 2^n \log n \\
&f(n) = \Theta(n^2 2^n \log n) \\
&t(n) = \Theta(n^2 2^n \log n)
\end{aligned}$$

7. $t(n) = 128t(n/2) + 6n$

$$\begin{aligned}
&a = 128, \quad b = 2, \quad f(n) = 6n \\
&f(n) = \Theta(n) \\
&a > b^1 \\
&t(n) = \Theta(n^7)
\end{aligned}$$

8. $t(n) = 128t(n/2) + 6n/n$

$$\text{Same as above, } t(n) = \Theta(n^7)$$

9. $t(n) = 128t(n/2) + 2/n$

$$\begin{aligned}
&a = 128, \quad b = 2, \quad f(n) = 2/n \\
&f(n) = \Theta(1/n) \\
&a > b^0 \\
&t(n) = \Theta(n^7)
\end{aligned}$$

10. $t(n) = 128t(n/2) + \log n$

$$\begin{aligned}
&a = 128, \quad b = 2, \quad f(n) = \log n \\
&f(n) = \Theta(\log n) \\
&a > b^0 \\
&t(n) = \Theta(n^7)
\end{aligned}$$

2 Q2. Selection Problem

When $r = 3$, can the selection problem be solved in $O(n)$ time complexity?

For the selection problem to be solved in $O(n)$ time, the algorithm should be linear. When $r = 3$, based on the table provided, the corresponding $f(n)$ term is $\Theta(n)$, which indicates that the problem can be solved in linear time $O(n)$.

3 Q3. Matrix Multiplication

What is the time complexity of the selected algorithm when $r = 7$?

For $r = 7$, the corresponding $h(n)$ is $\Theta(n^7)$. According to the table provided, the time complexity of the selected algorithm will be $\Theta(n^7)$.

To prove that the time complexity of the divide-and-conquer algorithm derived from the usual 2×2 block matrix multiplication is $\Theta(n^3)$, we refer to Example 14-3 on page 435.

Consider the matrices A and B divided into 2×2 blocks:

$$A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix}, \quad B = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix}$$

The product $C = A \times B$ is given by:

$$C = \begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix}$$

where

$$\begin{aligned} C_1 &= A_1B_1 + A_2B_3 \\ C_2 &= A_1B_2 + A_2B_4 \\ C_3 &= A_3B_1 + A_4B_3 \\ C_4 &= A_3B_2 + A_4B_4 \end{aligned}$$

Using these equations, the divide-and-conquer algorithm recursively multiplies the $n/2 \times n/2$ matrices and combines the results. The recurrence relation for the time complexity $T(n)$ is:

$$T(n) = 8T(n/2) + \Theta(n^2)$$

Solving this recurrence using the Master Theorem, we find:

$$a = 8, \quad b = 2, \quad f(n) = \Theta(n^2)$$

Since $a = b^3$, by Case 1 of the Master Theorem:

$$T(n) = \Theta(n^3)$$

Thus, the time complexity of the divide-and-conquer algorithm is $\Theta(n^3)$.

4 Q4. Recurrence Relation for Defective Chessboard

The defective chessboard problem involves covering a $2^k \times 2^k$ board with one defective square using triominoes. For $t(k)$, the time needed to cover the board, the recurrence relation is:

$$t(k) = \begin{cases} d & \text{if } k = 0 \\ 4t(k-1) + c & \text{if } k > 0 \end{cases}$$

Base Case

For $k = 0$:

$$t(0) = d$$

Recursive Case

For $k > 0$:

$$t(k) = 4t(k-1) + c$$

Solving the Recurrence

To solve the recurrence, we can expand it step-by-step:

$$\begin{aligned}
t(k) &= 4t(k-1) + c \\
&= 4(4t(k-2) + c) + c \\
&= 4^2 t(k-2) + 4c + c \\
&= 4^2(4t(k-3) + c) + 4c + c \\
&= 4^3 t(k-3) + 4^2 c + 4c + c \\
&= \dots \\
&= 4^k t(0) + c \sum_{i=0}^{k-1} 4^i \\
&= 4^k d + c \frac{4^k - 1}{3} \\
&= \Theta(4^k)
\end{aligned}$$

Thus, the time complexity of the algorithm is $\Theta(4^k)$.