Section L6: Quiz 1 Solution

Question 1

Solve the recurrence relation $T(n) = 4T(n/4) + n$.

Solution by Substitution (Unfolding)

We expand the recurrence to find the pattern.

Step 1: Expand the recurrence

$$\begin{aligned}
T(n) &= 4T\left(\frac{n}{4}\right) + n \\
T\left(\frac{n}{4}\right) &= 4T\left(\frac{n}{16}\right) + \frac{n}{4} \\
\text{Substituting: } T(n) &= 4\left[4T\left(\frac{n}{16}\right) + \frac{n}{4}\right] + n \\
&= 16T\left(\frac{n}{16}\right) + n + n \\
&= 16T\left(\frac{n}{16}\right) + 2n \\
T\left(\frac{n}{16}\right) &= 4T\left(\frac{n}{64}\right) + \frac{n}{16} \\
\text{Substituting: } T(n) &= 16\left[4T\left(\frac{n}{64}\right) + \frac{n}{16}\right] + 2n \\
&= 64T\left(\frac{n}{64}\right) + n + 2n \\
&= 64T\left(\frac{n}{64}\right) + 3n
\end{aligned}$$

Step 2: Generalize the pattern

The general pattern is:

$$T(n) = 4^k T\left(\frac{n}{4^k}\right) + kn$$

Step 3: Base case and stop condition

The recurrence stops when $\frac{n}{4^k} = 1$, i.e., $k = \log_4 n$. At this point:

$$T(1) = c \quad (\text{ some constant})$$

Substituting $k = \log_4 n$:

$$\begin{aligned} T(n) &= 4^{\log_4 n} T(1) + (\log_4 n)n \\ &= n \cdot c + n \cdot \log_4 n \end{aligned}$$

Final Solution

The solution is:

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

Solution by Mathematical Induction

Solution for $T(n) = 4T(n/4) + n$ by Induction

We solve the recurrence relation $T(n) = 4T(n/4)+n$ using induction and explicit substitution for $T(n/4)$. Let us proceed step by step:

Step 1: Base Case

For $n = 1$, $T(1)$ is constant (denoted c):

$$T(1) = c.$$

The base case holds.

Step 2: Inductive Hypothesis

Guess the solution:

$$T(n) = c \cdot n \log(n),$$

where c is a constant. Assume:

$$T(n/4) = c \cdot \frac{n}{4} \log\left(\frac{n}{4}\right).$$

Step 3: Inductive Step

Substitute $T(n/4)$ into the recurrence relation $T(n) = 4T(n/4) + n$:

$$T(n) = 4T(n/4) + n.$$

Using the inductive hypothesis, substitute for $T(n/4)$:

$$T(n/4) = c \cdot \frac{n}{4} \log\left(\frac{n}{4}\right).$$

Substitute this into $T(n)$:

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

Simplify the first term:

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

Expand $\log\left(\frac{n}{4}\right)$ using logarithm rules ($\log(a/b) = \log(a) - \log(b)$):

$$\log\left(\frac{n}{4}\right) = \log(n) - \log(4).$$

Substitute this back into $T(n)$:

$$T(n) = c \cdot n\left(\log(n) - \log(4)\right) + n.$$

Simplify:

$$T(n) = c \cdot n \log(n) - c \cdot n \log(4) + n.$$

Factorize:

$$T(n) = c \cdot n \log(n) + n\left(1 - c \cdot \log(4)\right).$$

Step 4: Conclusion

By induction, the solution to the recurrence relation $T(n) = 4T(n/4) + n$ is:

$$T(n) = c \cdot n \log(n) + n\left(1 - c \cdot \log(4)\right).$$

As n grows large, the dominant term is $c \cdot n \log(n)$, confirming the growth rate of $T(n)$. Take c=1 and n0=1

Question 2

2. Prove or disprove $f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$.

Using the definitions of little-o and little-omega:

- $f(n) = o(g(n))$ implies:

  $$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

  This means that $f(n)$ grows asymptotically slower than $g(n)$.

- $g(n) = \omega(f(n))$ implies:
$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty.$$
This means that $g(n)$ grows asymptotically faster than $f(n)$.

The two definitions are equivalent because:

- If $f(n) = o(g(n))$, then by definition, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, which is equivalent to $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$, implying $g(n) = \omega(f(n))$.
- Similarly, if $g(n) = \omega(f(n))$, then $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$, which is equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, implying $f(n) = o(g(n))$.

Conclusion: The statement is true. $f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$.

Question 3

Solution for $n^2/2 = \omega(n)$ (Little Omega)

To determine whether $\frac{n^2}{2} = \omega(n)$, we use the formal definition of little omega (ω):
A function $f(n) = \omega(g(n))$ if and only if:

$$\forall c > 0, \exists n_0 > 0 \text{ such that } |f(n)| > c \cdot |g(n)| \text{ for all } n \geq n_0.$$

Here, $f(n) = \frac{n^2}{2}$ and $g(n) = n$.
Step-by-Step Verification:
1. Set up the inequality:
$$\frac{n^2}{2} > c \cdot n.$$
2. Simplify the inequality: Divide both sides by n (valid for $n > 0$):
$$\frac{n}{2} > c.$$

3. Analyze the inequality: For any constant $c > 0$, there exists an $n_0 > 0$ such that $\frac{n}{2} > c$ for all $n \geq n_0$. For example, choosing $n_0 = 2c$, the inequality holds because:
$$\frac{n_0}{2} = \frac{2c}{2} = c.$$
Thus, for $n \geq n_0$, $\frac{n}{2} > c$.
4. Conclusion: Since the inequality holds for any $c > 0$ and for sufficiently large n, we conclude that:
$$\frac{n^2}{2} = \omega(n).$$

Question 4

4. Sorting algorithm pseudocode and worst-case running time.

Pseudocode

```
function SelectionSort(A[1:n]):
    for i = 1 to n-1:
        minIndex = i
        for j = i+1 to n:
            if A[j] < A[minIndex]:
                minIndex = j
        Swap(A[i], A[minIndex])
```

Worst-case running time

The worst-case running time occurs when the algorithm performs the maximum number of comparisons and swaps. For an array of size n:

- In the first iteration, $n-1$ comparisons are made.
- In the second iteration, $n-2$ comparisons are made.
- This pattern continues until 1 comparison is made in the last iteration.

The total number of comparisons is given by:

$$T(n) = (n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2}.$$

Thus, the worst-case time complexity is:

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