Advanced Data Structures and Algorithms

Comprehensive Assignment Solutions

Arya Raag
Roll No: A125002

M.Tech (Computer Science and Engineering)

January 5, 2026

1 Heap Operations and Analysis

Question 1. Prove that the time complexity of the recursive Heapify operation is $O(\log n)$ using the recurrence relation:

$$T(n) = T\left(\frac{2n}{3}\right) + O(1)$$

Detailed Solution:

Problem Analysis

The `Max-Heapify` operation is a fundamental procedure used to maintain the max-heap property. When called on a node i, it assumes that the binary trees rooted at $\text{Left}(i)$ and $\text{Right}(i)$ are max-heaps, but $A[i]$ might be smaller than its children. The algorithm "floats" the value at $A[i]$ down the tree.

The worst-case running time occurs when the bottom level of the tree is exactly half full. In this specific configuration, the left subtree contains more nodes than the right subtree. Specifically, the left subtree can have size at most $\frac{2n}{3}$.

Formal Proof via Recurrence Solving

The recurrence relation governing the worst-case time complexity is given by:

$$T(n) \leq T\left(\frac{2n}{3}\right) + \Theta(1)$$

Here, $\Theta(1)$ represents the constant time required to compare the node with its children and perform a swap if necessary.

To solve this, we can employ the expansion method (also known as the iteration method).

Step 1: Expand the Recurrence

We iteratively substitute the recurrence into itself:

$$\begin{aligned}
T(n) &= T\left(\frac{2}{3}n\right) + c \\
&= \left[T\left(\frac{2}{3}\cdot\frac{2}{3}n\right) + c\right] + c = T\left(\left(\frac{2}{3}\right)^2 n\right) + 2c \\
&= T\left(\left(\frac{2}{3}\right)^3 n\right) + 3c \\
&\vdots \\
&= T\left(\left(\frac{2}{3}\right)^k n\right) + k\cdot c
\end{aligned}$$

Step 2: Determine the Stopping Condition
The recursion terminates when the problem size reduces to a base case, typically $n = 1$. Let k be the depth of recursion where this occurs:

$$\left(\frac{2}{3}\right)^k n = 1$$

Solving for k:

$$n = \left(\frac{3}{2}\right)^k$$

Taking the logarithm (base 3/2) on both sides:

$$k = \log_{3/2} n$$

Using the change of base formula, we know that $\log_{3/2} n = \frac{\ln n}{\ln 1.5}$. Since $\ln 1.5$ is a constant, $k \approx c' \ln n$. Thus, $k = O(\log n)$.

Step 3: Calculate Total Cost
Substituting k back into the expanded equation:

$$T(n) = T(1) + O(1)\cdot\log_{3/2} n$$

$$T(n) = O(1) + O(\log n)$$

Conclusion

The height of a binary heap is logarithmic with respect to the number of nodes. Since `Heapify` performs constant work at each level of the tree and descends at most the height of the tree, the time complexity is:

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