Proof of Time Complexity of Recursive Heapify

Manav Mantry

Problem Statement

Prove that the time complexity of the recursive **Heapify** operation is

$$O(\log n)$$

using the recurrence relation:

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

Understanding the Recurrence

The Heapify operation compares a node with its children and possibly swaps it with the larger (or smaller) child. At each recursive step:

- The problem size reduces from n to at most $\frac{2n}{3}$.
- A constant amount of work $O(1)$ is done for comparisons and swaps.

Thus, the recurrence relation is:

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

where c is a positive constant.

Solving the Recurrence

We expand the recurrence step by step:

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

Base Case

The recursion stops when:

$$\frac{2^k n}{3^k} = 1$$

Solving for k:

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

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

Final Complexity

Substituting $k = \log_{3/2} n$ into the expanded recurrence:

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

Since logarithms with different bases differ only by a constant factor:

$$\log_{3/2} n = O(\log n)$$

Conclusion

$$\boxed{T(n) = O(\log n)}$$

Hence, the time complexity of the recursive Heapify operation is:

$$\boxed{O(\log n)}$$

This result matches the height of a binary heap, which is $\log n$, confirming that Heapify runs in logarithmic time.