MTech CSE – 1st Semester
Student ID: **A125023**
Student Name: **SRIJITA VERMA**

IIIT BHUBANESWAR
I I I T Bhubaneswar
Imagine, Innovate, Inspire
(A University established by Government of Odisha)

Question

Using the above result, prove that the time complexity of the BUILD-HEAP algorithm is $O(n)$.

We use the result proved in Question 3(a):

In a heap containing n elements, the number of nodes at height h is at most

$$N_h \leq \left\lceil \frac{n}{2^{h+1}} \right\rceil .$$

Answer

Overview of the Approach

The BUILD-HEAP algorithm constructs a heap from an arbitrary array by calling the HEAPIFY procedure on internal nodes, starting from the lowest internal nodes and moving upward to the root.

Although a single call to HEAPIFY can take $O(\log n)$ time in the worst case, we show that the total cost of all Heapify calls combined is linear, i.e., $O(n)$.

This is achieved by:

- Grouping nodes by their height,
- Bounding the number of nodes at each height using the result from Question 3(a),
- Summing the total work over all heights.

The BUILD-HEAP algorithm constructs the heap in a bottom-up manner by applying HEAPIFY starting from the lowest internal nodes up to the root.

Why Build-Heap Is Not $O(n \log n)$

A naive analysis might assume that since HEAPIFY takes $O(\log n)$ time and is called on $O(n)$ nodes, the total running time is $O(n \log n)$. This argument is incorrect because it ignores the fact that most nodes in a heap are located at very small heights. The height-based analysis shows that the majority of HEAPIFY calls take constant or near-constant time, while only a few nodes incur the maximum cost.

Preliminaries

Cost of Heapify

The Heapify procedure is called on a node and may move the node down the heap. The maximum number of levels it can move down equals the height of that node.

Therefore, the running time of Heapify on a node of height h is:

$$O(h).$$

Structure of Build-Heap

The standard Build-Heap algorithm works as follows:

- It calls Heapify on all non-leaf nodes.
- Leaves have height 0 and require no work.
- Nodes closer to the root have larger height and higher cost per Heapify call.

Leaf nodes are excluded since they trivially satisfy the heap property and require no heapification.

Step 1: Group Nodes by Height

Let:

- n be the number of elements in the heap,
- $h \geq 0$ denote a height,
- N_h be the number of nodes of height h.

From Question 3(a), we know that:

$$N_h \leq \left\lceil \frac{n}{2^{h+1}} \right\rceil .$$

Step 2: Cost Contributed by Nodes at Height h

Each node of height h:

- Requires $O(h)$ time for Heapify.

Thus, the total cost contributed by all nodes at height h is:

$$O(N_h \cdot h).$$

Substituting the bound on N_h:

$$O\left(\frac{n}{2^{h+1}} \cdot h\right).$$

Step 3: Total Cost of Build-Heap

To obtain the total running time, we sum over all possible heights:

$$T(n) = \sum_{h=0}^{\lfloor \log n \rfloor} O\Big(\frac{n}{2^{h+1}} \cdot h\Big).$$

Factoring out n:

$$T(n) = O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^{h+1}} \right).$$

Step 4: Evaluating the Summation

Consider the infinite series:

$$\sum_{h=0}^{\infty} \frac{h}{2^{h+1}}.$$

This series is convergent and evaluates to a constant. In fact:

$$\sum_{h=0}^{\infty} \frac{h}{2^{h+1}} = 1.$$

Since truncating a convergent series only decreases its value, we have:

$$\sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^{h+1}} \leq 1.$$

So, $\sum_{h=0}^{\infty} \frac{h}{2^{h+1}}$ is a convergent geometric series weighted by h, its value is bounded by a constant independent of n.

Step 5: Final Time Complexity

Substituting this bound back into the expression for $T(n)$:

$$T(n) = O(n \cdot 1) = O(n).$$

Final Result

$$\boxed{\textsc{Build-Heap runs in } O(n) \text{ time.}}$$

Moreover, this bound is tight, as every element in the input array must be examined at least once, implying a lower bound of $\Omega(n)$ for heap construction.

Remark

Although the Heapify operation has a worst-case time complexity of $O(\log n)$, the Build-Heap algorithm runs in linear time because only a small number of nodes are located at large heights. Most nodes are near the leaves and require constant or very small time to heapify. The bound on the number of nodes at each height ensures that the total work performed across all heap levels is linear in n.

Intuition Behind the Result

- Most nodes in a heap are near the leaves.
- Nodes near the leaves have very small height.
- Expensive Heapify operations are performed only on a few nodes near the root.
- The large number of cheap operations dominates the total cost.
- Nodes at level k from the root have height approximately $\log n - k$, and their number grows exponentially as k increases.

This imbalance ensures that the total running time is linear, not $O(n \log n)$.

Why This Result Is Important

This analysis explains a key and somewhat counterintuitive fact:

Even though Heapify takes $O(\log n)$ time in the worst case, Build-Heap runs in $O(n)$ time overall.

This result is fundamental in:

- Heap construction,
- Heapsort analysis,
- Priority queue implementations.