CS 1203: Data Structures **Semester:** Monsoon 2023

Assignment #3

Instructor: Subhamoy Mitra **Student Name:** Varsha Baisane
Due: September 15, 2023 varsha.baisane_ug25#@ashoka.edu.in

Broad topics discussed

1. Binary tree: Terminologies, child, parent, node, root, leaf nodes. Degrees, height and also edges. We discussed what a complete binary tree is, the way a binary tree is filled is by filling as left as possible, where if parent is i, left child is $2i+1$ and right child $2i+2$. If the child's given by j, then the parent is $(j-1) \div 2$
Implementations: Searching and sorting algorithms.

2. Heaps: Data structure represented using binary trees, can be used as a sorting algorithm and proves to be an efficient one generally implemented using arrays. In a heap, all elements are heaped as min or max depending on the type of heapify algorithm, min.heap or max.heap.

3. Max-heap and min-heap: For any given node, the value of that node is greater than or equal to the values of its children. This means that the maximum element is found at the root. In a min-heap, for any given node, the value of that node is less than or equal to the values of its children. This means that the minimum element is found at the root.

4. Operations:
Insertion: Adding an element to the heap is done in O(log n) time complexity. Deletion (maximum or minimum): Removing the maximum (or minimum) element from the heap is done in O(log n) time.

5. Heapify: The process of converting an array into a valid heap is called "heapify." It ensures that the heap property is maintained. Note: Heap is not a sorted data structure and thus needs to be sorted according to the stated need.

6. Applications:
Priority Queues: Heaps are used to implement priority queues. Elements are assigned priorities, and the heap ensures efficient retrieval of the element with the highest priority.
Sorting: Heapsort is a sorting algorithm that uses a max-heap to sort elements in O(n log n) time complexity.

Asssignment Question 1
Explain the heapify algorithm and talk about the gap in the program:

Heapify:

1. Start at the last non-leaf node in the heap. This node is at index (n/2) - 1, where "n" is the total number of elements in the heap .
2. For each node, perform a down operation, which involves comparing the node's value with the values of its children. If the node's value violates the heap property (e.g., in a max-heap, if the node's value is smaller than one of its children), you swap the node with the larger (in a max-heap) or smaller (in a min-heap) child.
3. After swapping, move to the next non-leaf node and repeat the "sift-down" operation.
4. Continue this process until you've checked and, if necessary, swapped each non-leaf node in the heap. This effectively restores the heap property throughout the entire heap.

Time Complexity: O(n), where ”n” is the number of elements in the heap. This is because we perform a constant amount of work for each non-leaf node in the heap, and there are roughly n/2 non-leaf nodes in a binary heap.

Heapify is used to turn an array into a valid heap, either a max-heap or a min-heap, where the heap property is satisfied. The heap property states that in a max-heap, the value of each node is greater than or equal to the values of its children, and in a min-heap, the value of each node is less than or equal to the values of its children.

A. Heap insert:

Heap insert inserts an element into the heap and doing so requires the program to insert while maintaining the heap property as mentioned within the heapify algorithm. (Comparing parents and nodes after inserting and swapping if property violated).

The gap in the program arises in the execution that leads to the difference in the time complexities of these algorithms.
Heapify works on the entire heap and swaps the elements where the property might be violated giving us a sorted heap or array (min or max). As it begins at n/2 - 1, the leaf nodes and compares, the time complexity associated with this is O(n)
whereas for the inserting program it is O(log n) as the insert algorithm might need to traverse the tree which is of height log n.
Given that both follow the heapify algorithm, the exceution and process leads to different time complexities that arise in the approach taken to execute.
Therefore, to conclude, heapify when used in insertion of an element takes O(log n) time whereas for building a heap, i.e. when taken the entire array takes O(n) time as it need to attend to every node in implementing and maintaining the property of say min heap or max heap.

B. Heap Delete:

Heap delete is used to remove and return the root element of the heap while maintaining the heap property. The specifics of this operation depend on whether the heap is a max-heap or a min-heap.

Max-Heap:
In a max-heap, the maximum element is stored at the root. To perform a heap delete in a max-heap, the root element (the maximum element) is removed. To maintain the heap property, the last leaf element in the heap is moved to the root position (replacing the removed maximum element). After the replacement, a down operation is performed, where the new root element is compared with its children. If the heap property is violated, swaps are made with the larger child node to restore the property. The down operation continues recursively until the heap property is satisfied throughout the heap.

Min-Heap:
In a min-heap, the minimum element is stored at the root. To perform a heap delete in a min-heap, the root element (the minimum element) is removed. To maintain the heap property, the last leaf element in the heap is moved to the root position (replacing the removed minimum element). After the replacement, a down operation is performed,

where the new root element is compared with its children. If the heap property is violated, swaps are made with the smaller child node to restore the property. The down operation continues recursively until the heap property is satisfied throughout the heap.

Time Complexity Analysis:

We know that an n-element heap has height $\lfloor \log n \rfloor$. Plus, an n-element heap has at most $\lfloor \frac{n}{2^{h+1}} \rfloor$ nodes of any height h So, expanding, we get the time complexity for the heapify program.//

$$\sum_{h=0}^{\lfloor \log n \rfloor} O(\log n) = O\left(\sum_{h=0}^{\lfloor \log n \rfloor} \log n \right)$$

The sum of $\log n$ from $h = 0$ to $\lfloor \log n \rfloor$ is $O(2^n) = O(n)$.

Proof

From geometric series:

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \text{ when } |x| < 1.$$

Differentiating this formula, we get:

$$\sum_{k=0}^{\infty} kx^{k-1} = \frac{1}{(1-x)^2},$$

and multiplying by x, we get:

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}.$$

Letting $x = \frac{1}{2}$ makes:

$$\sum_{h=0}^{\infty} h \left(\frac{1}{2} \right)^h = 2.$$

Source: CS 161 Lecture 4 document