Heap Data Structures: Implementation, Analysis, and Applications

Heapsort uses a binary heap data structure, which is a comparison-based algorithm. It will create a max heap from the input data and extract the maximum element to sort the array repeatedly in ascending order. It places the largest element at the end of the unsorted array. It does not assume input's order because it organizes data in a binary tree.

Analysis of Implementation:

- The time complexity of the heapsort will be logarithmic of n times. Each node in the heap will need to be operated using a bottom-up method. That means each call will be heapify with height h in heap times O(h). Heap construction and heap sort phase will remain the same on input sizes because removing the root will require O (1), and reconstructing the heap will take O(logn) because heap height will depend on log n. So, It will be O (n logn).
- It is like other algorithms, such as quick and merge sort, with an average O (n logn). However, because of cache efficiency, quicksort will perform better in an average case scenario.
- Quicksort and Merge sort are more stable than heapsort.

Priority Queue: Elements with higher priority are dequeued before elements with lower priority. Each element is associated with a priority value. If both elements have the same priority, they will be prioritized according to the queue.

Analysis of implementation (core operations)

- **Insert:** Task is added to the end of the heap, and the heap is maintained by shifting upward to the root. The best-case scenario will be when the priority is smaller. So, no

further adjustment is required, resulting in a constant time of O (1). The worst case will be if the heap height increases to log n.

- **Extract-Max:** To remove or extract the maximum element, it will be required to get the last element from the heap. Then, the heap is adjusted downward to restore the heap property. Because the maximum element will be the last element of the heap, the best case will be when the heap has only one element. That means it can extract with O(1) time. However, if the heap has more aspects of log n, it will slow the execution as it needs to move down to a leaf node O(logn).
- **Change Priority:** Once the priority is updated, the heap needs to adjust based on new priority values. That means it could be either downward or upward the heap. The best case will be that there will be no change in priority order. That time, it will just update the priority value, and there is no need to move up or down the node O (1). However, the worst case will be when the priority value changes, and their order needs to move upward or downward. O(logn) will be the worst-case scenario.

References:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

GeeksforGeeks. (n.d.). Heap Sort. Retrieved January 25, 2025, from https://www.geeksforgeeks.org/heap-sort/

GeeksforGeeks. (n.d.). Priority Queue | Set 1 (Introduction). Retrieved January 25, 2025, from https://www.geeksforgeeks.org/priority-queue-set-1-introduction/