**Assignment 4: Heap Data Structures: Implementation, Analysis, and Applications**

**Heapsort Implementation and Analysis**

1. **Implementation**

Heapsort is a well-known sorting approach. In this implementation phase, two major methods are implemented i.e. ‘heapify’ and ‘sort\_heap’. Within the ‘heapify’ method, the max heap property is satisfied with the subtree rooted in some index (i). In this scenario, it helps in checking the right and left children of the node while maintaining that if it is found that the left or the right children are larger than the current node the values of the nodes are being swapped and the heap. Another method i.e. ‘sort\_heap’ where a max heap is created and it is starting from the last non-leaf node. In this method, the ‘heapify’ method is called to ensure the property of the heap for each node. Additionally, within the ‘sort\_heap’ method, elements are extracted from the heap and finally again ‘heapify’ is called to restore the property of the heap.

1. **Analysis of Implementation**

In this approach, the implementation is divided into two parts. The first one is the construction of the heap i.e. the building of the max heap and another one is sorting the heap i.e. the extraction. Assuming that the height of a binary tree is so for the ‘heapify’ method in the worst case, it is traversed down from the tree to a leaf which takes . Also, the ‘heapify’ method is called for every leaf node where the number of leaf nodes is .

So, if the height of the heap is then the construction of the max heap will take time and at the height the number of nodes is . Here the time total required will be .

Again, for the method ‘sort\_heap’, the extraction of the elements will be done and the ‘heapify’ method is called which takes time. The extraction of the elements will take time considering there is numbers of extractions are made.

Thus, the overall time complexity in all cases is .

In all of the cases i.e. worst case, average case and best case the time complexity for the heapsort is . In all of the cases, the construction of the heap is taking time and every extraction needed time. Thus, the overall time required is .

Considering the space complexity, first of all, heapsort does not require any additional space proportional to the size of the input array but when the recursive calls are made in the ‘heapify’ method, the space complexity will be as recursively it goes to the depth of .

There are additional overheads like recursive overhead, comparison overhead, overhead created while swapping etc. can be observed though there is no such changes in time complexity can be observed.

1. **Comparison**

The comparison between the running time of Heapsort with other sorting algorithms like Quicksort and Merge Sort on different input sizes and distributions are discussed below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sorting Algorithms** | **Random Input** | **Sorted Input** | **Reverse Sorted Input** |
| **Heapsort** | Taken time is similar to the Merge Sort i.e. . | Taken the similar time i.e. . | Taken the similar time i.e. . |
| **Quicksort** | The average time required for it is . | Considering the worst-case scenario the total time taken is . | Here the taken time is . |
| **Merge Sort** | The constant factors are higher than Quicksort but take time . | The required time is the same as the input order. | Taken time is . |

Considering the theoretical concepts, in the best case all of these three approaches are working well and apart from heapsort, the other approaches will take time . While in average cases, the quicksort performs better. Finally, for the worst-case segment, Merge sort and Quicksort are degraded by their efficiency and take time for poor pivots. Otherwise, Heapsort and Merge Sort completed their execution within time.

**Priority Queue Implementation and Applications**

**Part A: Priority Queue Implementation**

1. **Data Structure:**

In this approach, an array-based binary heap has been taken so that the utilization of space can be handled efficiently and heap property can be implemented in a simplified manner. Here the utilized binary tree has a parent node where at most two children will be there and the highest priority will always be at the root using max-heap functionality.

On the other hand, a Task Class has been created to store the information related to each task i.e. task ID, priority, arrival time, deadline etc.

Apart from that, all the required functionalities are implemented in the Priority Queue.