Assignment 4: Heap Data Structures

Shrisan Kapali

Student Id: 005032249

MSCS 532 – Algorithms and Data Structures

Satish Penmatsa

January 26, 2025

GitHub link

https://github.com/ShrisanKapali-Cumberlands/MSCS532_Assignment4

Heapsort Implementation and Analysis

As for GeeksforGeeks (2025), a heap sort is a technique that converts the given array into a max or min heap using heapify. A max heap is a complete binary tree where all the parent node elements are greater than or equal to a child node, and a min-heap is where the parent node has elements that are smaller than the child nodes.

The time complexity of a heap sort for all the cases, i.e., best, worst, and average cases, is O(nlogn). The steps that the heap sort uses are

1. First, the array is transformed into a complete binary tree by inserting the elements as a node in a breadth-first manner (Alake, n.d.).
2. The binary tree is then converted into a max or min heap.
3. Swap the root element with the last element.
4. Call the heapify function to restore the heap property.
5. Repeat steps 3 and 4 until all the elements are extracted and sorted.

The time complexity of building the heap is O(n), and sorting and performing each heapify is O(logn). As there are n number of elements in the array, and we perform heapify on each element, the overall time complexity is O(nlogn). Thus, regardless of the array size, as the heapify function is performed n times, the time complexity of the heap sort in the worst, average, and best case is always O(nlogn).

As heap sort is an in-place algorithm, it does not need additional memory to sort the array elements (Alake, n.d.). Thus, the space complexity of a heap sort is O (1).

While heap sort provides the same time complexity in all the cases as merge sort, which is O (nlogn), it has certain additional overheads. As in heap sort, we swap the arrays, and the relative order of the elements is not maintained. This may be a problem where the stability of the array is of concern. Heap sort has a higher constant factor, which may take additional execution time compared to merge and quicksort.

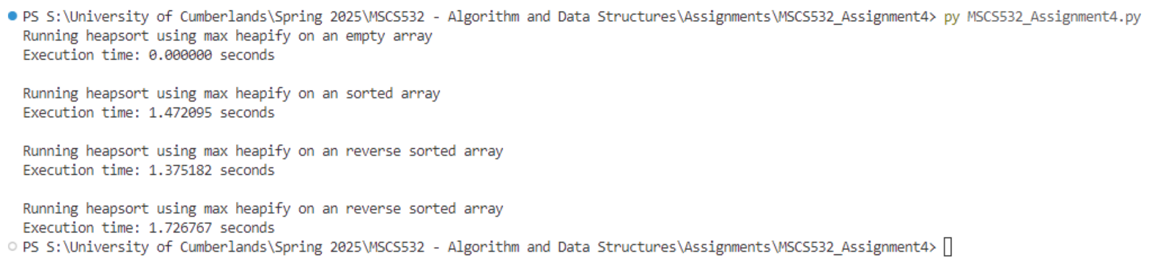
The time complexity of heap sort when compared to quick and merge sort is:

Heap Sort : O (nlogn) [for all cases]

Quick Sort: O (nlogn) [best and average case] $O(n^2)$ [for worst case]

Merge Sort: O (nlogn) [for all cases]

In assignment 3, while running randomized quicksort in an array of size 500000 on an array which is sorted order, it took 0.722362 seconds; on a reverse order, it took 0.703093 seconds and randomized took 0.0000 seconds. When running heap sort on the same set of test arrays, it took 1.472095 seconds to sort the array that was already in sorted order, 1.375182 seconds to sort the array that was initially in reverse sorted order, and 1.7626767 seconds to sort an array size 500000 which was in randomly sorted order. Likewise, on assignment 2, when we ran the merge sort on the same data set, the merge sort took nearly 0 seconds to run the sort.



```
PS S:\University of Cumberlands\Spring 2025\MSCS532 - Algorithm and Data Structures\Assignments\MSCS532_Assignment4> py MSCS532_Assignment4.py
Running heapsort using max heapify on an empty array
Execution time: 0.000000 seconds

Running heapsort using max heapify on an sorted array
Execution time: 1.472095 seconds

Running heapsort using max heapify on an reverse sorted array
Execution time: 1.375182 seconds

Running heapsort using max heapify on an reverse sorted array
Execution time: 1.726767 seconds
PS S:\University of Cumberlands\Spring 2025\MSCS532 - Algorithm and Data Structures\Assignments\MSCS532_Assignment4>
```

Although the theoretical time complexity of quicksort and merge sort in best and average case is same as in heap sort, the quicksort and merge sort execution time in practical applications are far less than the heap sort.

Priority Queue Implementation and Applications

A priority queue is a queue where every item has a priority assigned to it, and the element with the highest priority is always extracted first. A priority queue can be implemented using an array, linked list, heap data, or a binary search tree ("What is priority queue," 2024). The time complexity of the priority queue when using different data structures is given as follows:

Type	Insert	Delete highest priority item	Get the highest priority item
Array	O (1)	O (n)	O (n)
Linked List	O (n)	O (1)	O (1)
Binary Tree	O (1)	O (logn)	O (logn)
Heap	O (logn)	O (logn)	O (1)

In the Python program, the priority queue has been implemented using a min-heap. Heap provides better time complexity when compared to an array or linked list. A min-heap is where the lowest item is at the root. A priority queue with a min-heap means the item with the lowest priority is popped up or removed the first. Python has built-in support for heap implementation using a list, so we used "heapq" to implement our min heap priority queue.

As in a binary heap, we need to maintain the heap property every time a new record is added, and the main root element is extracted, which takes a time complexity of O(logn). Therefore, the time complexity of insertion and deletion is O(logn).

In the program, we declare the tasks class that takes parameters id, priority, arrival time, and deadline. It has a comparator operator "__lt__" where we let the python's heap know to treat lower priority as the higher priority. We then implemented methods to insert and extract items with minimum priority, decrease the priority of the inserted task, and a method to check if the heap is empty.

The insert function adds a new record in the priority queue while maintaining the min-heap property. As a heap has a time complexity of O (logn) for insertion, and the priority queue uses min heap, the insert function has a time complexity of O(logn).

To remove a record, i.e., parent node element, a min heap has a time complexity of O (logn), and as the extract_min function is removing the element with the highest priority, i.e., in min-heap, it is the element with the lowest priority, it has a time complexity of O (logn).

The is_empty method is simply checking the size of the heap, which takes a constant time of O (1).

The increase_key and decrease_key functions update the priority of the given tasks and use the heapify method to maintain the min-heap property. Heapify has a time complexity of O(n).

References

Alake, R. (n.d.). *Heap sort, explained*. Built In. https://builtin.com/data-science/heap-sort

GeeksforGeeks. (2024, December 6). *What is priority queue: Introduction to priority queue*. GeeksforGeeks. https://www.geeksforgeeks.org/priority-queue-set-1-introduction/

Heap sort - data structures and algorithms tutorials. GeeksforGeeks. (2025, January 2). https://www.geeksforgeeks.org/heap-sort/