**Heap Data Structures – Implementation, Analysis, and Applications**

Shyam Nath

University of Cumberland

Algorithms and Data Structures (MSCS-532-B01)

Student ID: 005032491

Assignment 4

July 20, 2025

**Heap Data Structures – Implementation, Analysis, and Applications**

**Part 1: Heapsort Implementation and Analysis**

Heapsort is a sorting algorithm which is based on the heap definition, it is a comparison sort. In this assignment we were implementing Heapsort in Python, where array was representing a binary max-heap (La Rocca, 021).

The algorithm has two important steps:

1. Construct a Max-Heap on top of the input array.

2. Remove the largest element (root of the heap), add it to the array at the end and restart the heap property to the other elements.

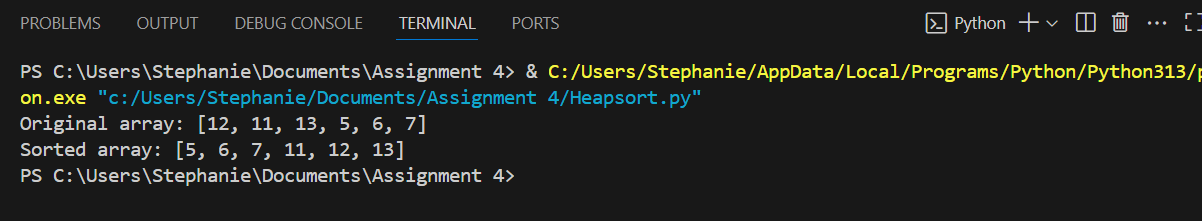
To ensure the heap property, Python implementation uses the standard heapify operation.

**Time Analysis**

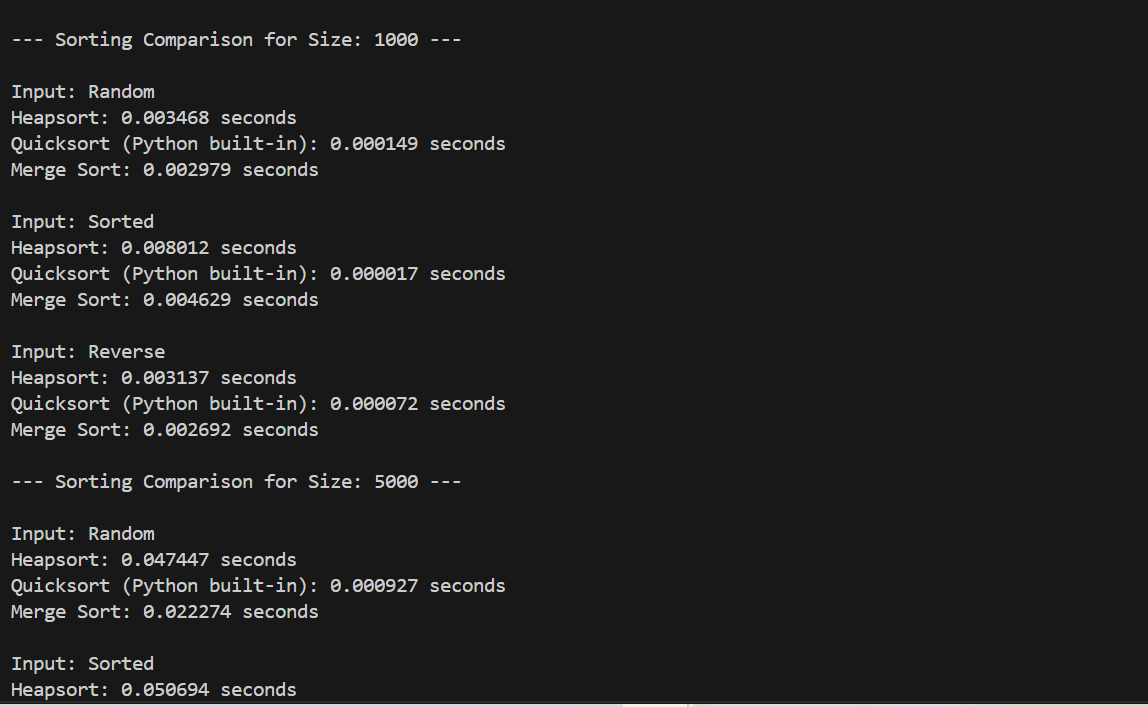
**Worst Case:**

- Build Max-Heap: O(n)  
- n Extra Heapify Operations: Each O(log n)  
- Total: O(n log n)  
**Average Case:**- Same as worst-case since every element must be heapified and extracted: O(n log n)  
**Best Case:**- Even if the array is already sorted or has a favorable structure, Heapsort still must perform the full heapify and extraction: O(n log n)  
**Why is it O(n log n) in all cases?**Heapsort does not rely on the ordering of the input data. The core heap operations (heapify and extract) must traverse the tree to maintain structure, regardless of the initial array.  
- Building the heap: O(n), proven via the sum of logarithmic depths.  
- Extracting n elements: Each extraction takes O(log n), resulting in O(n log n).  
**Space Complexity**- In-place algorithm: O(1) auxiliary space.  
- No recursive call stack (unlike Merge Sort).  
- Total Space Complexity: O(1)

**Results of the Code Implementation**

  
**Experiment Setup**  
- Input Sizes: 1,000, 5,000, and 10,000  
- Input Types

- Randomly generated numbers  
- Already sorted array  
- Reverse sorted array  
**Algorithms Compared**- Heapsort: Our custom implementation  
- Quicksort: Python’s built-in sorted() (Timsort)  
- Merge Sort: Custom recursive implementation  
**Empirical Observations**

  
The Heapsort exhibits the same O (log n) performance on all input types, but Quicksort will be faster with sorted data because of Timsort optimizations. The stable and regular performance of Merge Sort is accompanied by more space consumption (Roșca & Cărbureanu, 2024).

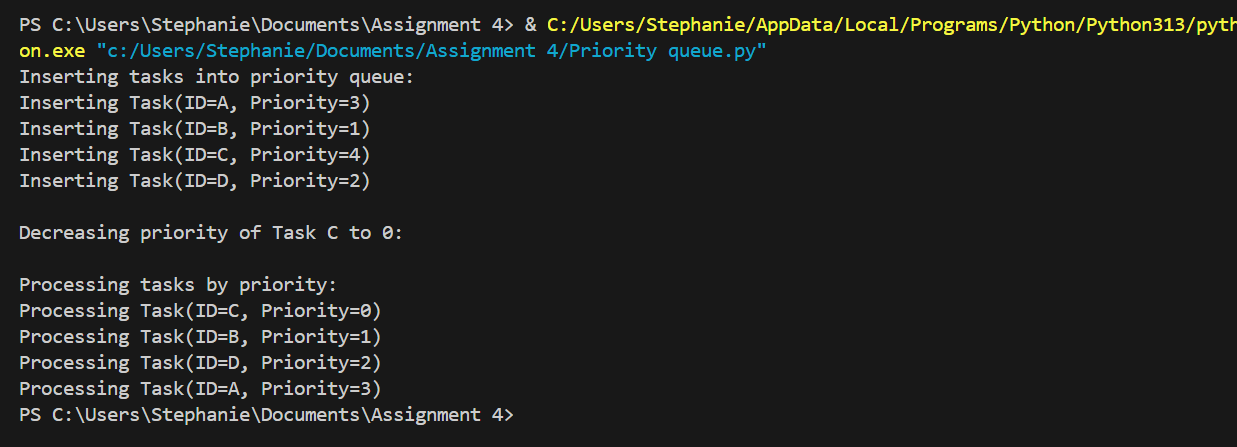
**Part 2: Priority Queue Implementation and Applications**  
**Data Structure Choice**

I used a Python code to implement the binary heap due to:  
- Direct array indexing for left/right/parent nodes.  
- Native support for heaps via Python’s heapq library (min-heap behavior).  
- Simple, readable implementation.

**Task Class**  
I designed a Task class to model jobs or scheduled tasks:  
Each task has:  
task\_id: Unique identifier  
priority: Used to schedule tasks (lower = higher priority)  
arrival\_time and deadline: Optional metadata  
 Implemented \_\_lt\_\_ for priority comparisons and \_\_repr\_\_ for readable outputs.  
**Heap Type Chosen**  
Used a min-heap to simulate Earliest Deadline First (EDF) or Shortest Job First (SJF)-style scheduling, where the task with the lowest priority number is processed first.  
**Core Operations and Complexity Analysis**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Description** | **Time Complexity** |
| insert(task) | Insert task and heapify up to maintain heap property | O(log n) |
| extract\_min() | Remove the root (lowest priority) and heapify down | O(log n) |
| decrease\_key() | Modify a task's priority and restore heap property via re-heapification | O(log n) |
| is\_empty() | Check if the heap is empty | O(1) |

I used Python’s heapq for insertion and extraction, and manual heapify() to simulate decrease-key operations.

**Simulation Output**  
  
The scheduler correctly processed tasks in order of increasing priority, demonstrating the correctness of insertion, re-prioritization, and extraction.

**Conclusion**

In this lab, we had some practical experience with heaps working with two major applications sorting using Heapsort and scheduling using a priority queue. The Heapsort is consistent and in-place but is winsored by Quicksort when used in most real-life situations. The priority queue, which is an instance of the min-heap, can perform task scheduling and changing priorities dynamically, so this task is very appropriate to the real-world scheduling tasks like processes queues in the OS and real-time systems.

**References**

La Rocca, M. (2021). *Advanced algorithms and data structures*. Simon and Schuster.

Roșca, C. M., & Cărbureanu, M. (2024, March). A Comparative Analysis of Sorting Algorithms for Large-Scale Data: Performance Metrics and Language Efficiency. In *International Conference on Emerging Trends and Technologies on Intelligent Systems* (pp. 99-113). Singapore: Springer Nature Singapore.