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

Jacob Jeffers

MSCS 532

Dr. Vanessa Cooper

Fall 2024

11/10/2024

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

Part 1: Heapsort Implementation and Analysis

Implement: To implement the Heapsort Algorithm, the first step is to understand what it does and how it functions. The heapsort algorithm converts an array into a max-heap, moves the largest element to the end, and then repeats the process (Cormen et al., 2022). To implement this in VS Code, I defined a function called "Heapify" and used "if" statements to program what should happen. It ensures that the largest value is chosen. I also defined a function called "Heapsort," which builds a max-heap from an array and then called the "Heapify" function to swap the largest element with the last element in the heap, reduce the heap size, and then start the process again. Below is a screenshot of the code and the output from the terminal.

A screenshot of a computer program

Description automatically generated

Analysis: Heapsort has a time complexity of O(nlogn) in all cases—worst, average, and best—because it involves two main phases: building a max-heap O(n) time and then performing n extractions, each costing O(logn) (Cormen et al., 2022). This makes it consistently efficient regardless of input order. It also has an O(1) space complexity since it sorts in place without needing additional memory proportional to the input size. However, frequent element swaps can make Heapsort slower in practice than other O(nlogn) algorithms, and its memory access pattern is not as cache-friendly, which can add some overhead (Cormen et al., 2022).

Comparison: In practice, Heapsort is slower than other O(nlogn) algorithms like Quicksort and Mergesort. After combining my code for these algorithms, I implemented code to test these algorithms on sorted, reverse-sorted, and random input sizes. The results can be seen in the screenshot below.

A screen shot of a computer program

Description automatically generated

From the results, Quicksort (using Python's Timsort) consistently outperformed Mergesort and Heapsort, especially on sorted and reverse-sorted arrays where Timsort's optimizations detect order and achieve close to O(n) performance. Mergesort maintained stable O(nlogn) times across all input types, performing slightly slower than Quicksort. While theoretically O(nlogn), Heapsort lagged both algorithms on all inputs.

Part 2: Priority Queue Implementation and Applications

1. Data Structure – I chose to implement an array in Python for this part. Working with arrays in my work and this course, I felt more comfortable using this than another data structure. Next, I designed a Task class that contains task\_id, priority, arrival\_time, and deadline in Python.

A screenshot of a computer program

Description automatically generated

I plan to use a min-heap in the last part since the task requires a deadline.

1. Core Operations

To build upon the previous work, I created the insert(self, task) function, which adds a new task (child) to the heap by appending it to the end of the array and then compares the task to its parent and swapping as necessary.

A screen shot of a computer code

Description automatically generated

For the next step, I created the extract\_min function to remove and return the task with the highest priority.

A screenshot of a computer program

Description automatically generated

Next, I created a function that updates the priority of a task and restores the heap property depending on the priority.

A computer screen shot of text

Description automatically generated

Lastly, I coded the is\_empty() function to check if the queue is empty.

A black background with white text

Description automatically generated

Lastly, I created a simple test to ensure that the inserted and extracted tasks were identical.

A computer screen shot of a program

Description automatically generated

The test passed!

A screenshot of a computer

Description automatically generated

GitHub Repository for Assignment 4: <https://github.com/jakejeffers/MSCS-532-Assignment-4>

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms, fourth edition*. MIT Press.