Assignment 4:

# Heap Data Structures: Implementation, Analysis, and Applications

---

Anna Levinskaia

ID 005038541

Algorithms and Data Structures (MSCS-532-B01)

University of the Cumberlands

November 7, 2025

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

## 1. Introduction

In this assignment, I worked with heap data structures in Python. I implemented Heapsort and a Priority Queue to understand how heaps organize and manage data efficiently. Both programs use arrays (Python lists) to store data.

## 2. Heapsort Implementation

Below is my Heapsort code. It builds a max-heap, then sorts the array in ascending order.

```
def heapify(arr, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    if left < n and arr[left] > arr[largest]:
        largest = left
    if right < n and arr[right] > arr[largest]:
        largest = right

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heapsort(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)


numbers = [12, 11, 13, 5, 6, 7]
print("Original array:", numbers)
heapsort(numbers)
print("Sorted array:", numbers)
```

Output:

```
Original array: [12, 11, 13, 5, 6, 7]
Sorted array: [5, 6, 7, 11, 12, 13]
```

## 3. Analysis of Heapsort

Building the heap: O(n)

Heapify per element: O(log n)

Total sorting time: O(n log n)

Best, average, and worst case: O(n log n)

Space complexity: O(1), because sorting happens in place.

Heapsort is efficient, reliable, and doesn’t need extra memory like Merge Sort does.

## 4. Comparison (Short Discussion)

I compared Heapsort with Quicksort and Merge Sort. All three algorithms have O(n log n) complexity, but their speed can vary in real tests. Heapsort is stable and works the same for sorted or unsorted data. Quicksort can be faster on average but can slow down on sorted data without random pivoting. Merge Sort is very stable but needs extra space for merging. Overall, Heapsort is a good choice when we need in-place sorting and predictable performance.

## 5. Priority Queue Implementation

Next, I implemented a Priority Queue using Python’s built-in heapq module, which is based on a min-heap. Here, tasks with smaller priority numbers are treated as more important.

```
import heapq
class Task:
    def __init__(self, name, priority):
        self.name = name
        self.priority = priority

    def __lt__(self, other):
        return self.priority < other.priority

    def __repr__(self):
        return f"{self.name} (priority: {self.priority})"

class PriorityQueue:
    def __init__(self):
        self.heap = []

    def insert(self, task):
        heapq.heappush(self.heap, task)
        print(f"Inserted: {task}")

    def extract_min(self):
        if self.is_empty():
            print("Queue is empty!")
            return None
        return heapq.heappop(self.heap)


    def is_empty(self):
        return len(self.heap) == 0
```

Example Output:

```
Inserted: Do homework (priority: 2)
Inserted: Wash dishes (priority: 3)
Inserted: Pay bills (priority: 1)


Tasks in order of priority:
Pay bills (priority: 1)
Do homework (priority: 2)
Wash dishes (priority: 3)
```

## 6. Time Complexity of Priority Queue Operations

insert(): O(log n)

extract_min(): O(log n)

is_empty(): O(1)

All operations are efficient and suitable for real-world scheduling systems like CPU task queues or print job managers.

## 7. Conclusion

This assignment helped me understand how heaps organize data and support both sorting and priority-based scheduling. Heapsort is a consistent O(n log n) sorting algorithm that works in place. The priority queue is a practical real-world application, efficiently managing tasks by priority using heap operations.

## 8. GitHub

https://github.com/AnnaLevin/Algorithms-and-Data-Structures-MSCS-532-B01-Assignment 4.git

Files:

- heapsort.py – Heapsort algorithm
- priority_queue.py – Priority Queue with heapq

How to run:

```
python heapsort.py
python priority_queue.py
```