**Avijit Saha**

**Student ID #005023281**

**Algorithms and Data Structures (MSCS-532-B01)**

**Assignment 4**

**November 10, 2024**

**Heapsort Implementation and Analysis**

**Heap Sort:** Heap sort is a comparison-based sorting technique based on Binary Heap Data Structure. It can be seen as an optimization over selection sort where we find the max (or min) element and swap it with the last (or first). We repeat the same process for the remaining elements. In Heap Sort, we use Binary Heap so that we can quickly find and move the max element in O(Log n) instead of O(n) and hence achieve the O(n Log n) time complexity.

First, I must build a max heap, then repeatedly extract the maximum element and maintain the heap property until the array is sorted.

**Complexity Analysis**

**Time Complexity:**

* **Building Heap:** The heap sort maintains the max-heap property by recursively adjusting nodes if any child node is more significant than its parents. We will call this function heapify(). Since the depth of each call is O(log n), building the heap is O(n). Since all nodes are not at the bottom of the tree, the average is out to O(n).
* **Sorting Phase:** Extracting each element from the heap takes O(log n), and we perform this n times, leading to O(n log n) completely.

So, the overall complexity for heapsort is O (n log n), and it is the same for all scenarios.

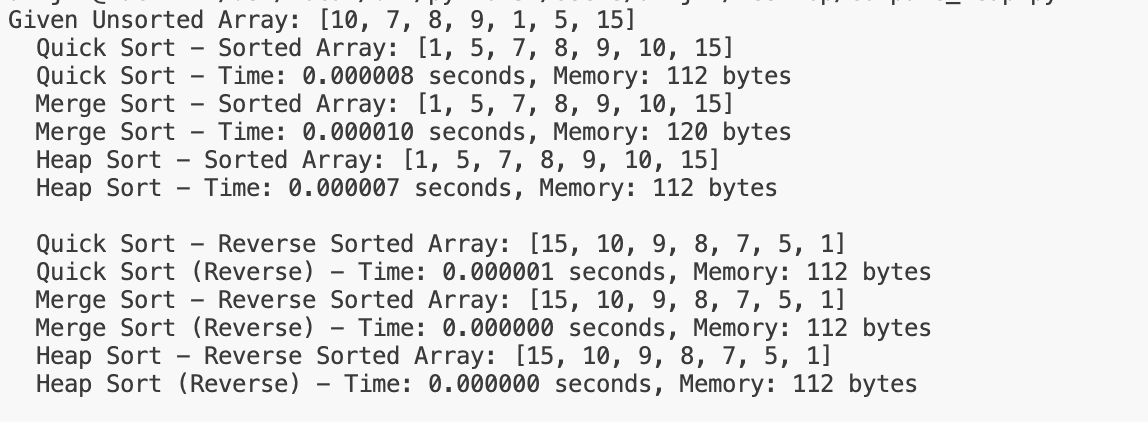
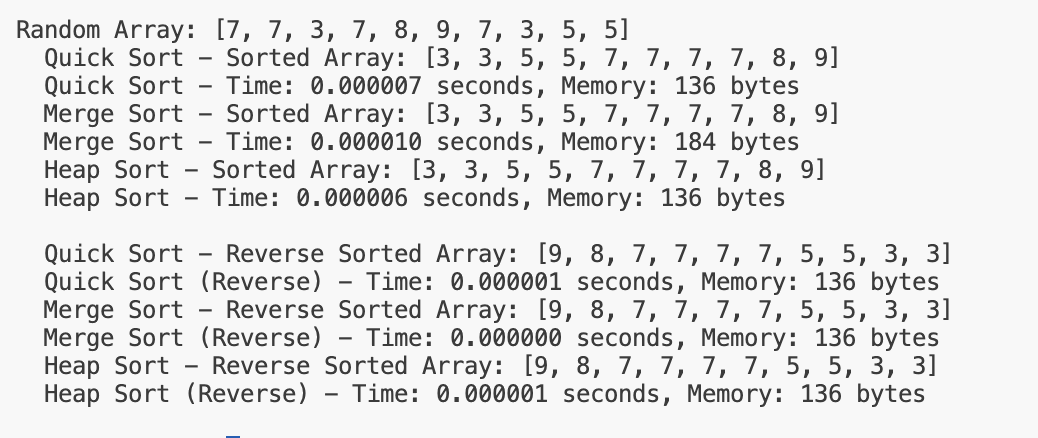
In the **worst case**, it will occur when the input array is in the worst condition, like in reverse order. But it will still work in O(n log n) time due to the repeated heapifying after each swap.

The time complexity for the **average case** is also the same because building the heap takes O(n), and the sorting phase takes O (n log n) regardless of the input distribution.

The **best-case** complexity is also O(n log n), as even if the array is already sorted, heapsort still needs to build the max heap and perform the sorting steps.

**Space Complexity:** Heap sort is an in-place sorting algorithm. So, the space complexity is O(1). This means it does not need any additional space proportional to the size of the input array for sorting purposes. The algorithm only uses a fixed amount of extra space for variables and function calls, and all heap operations are performed within the input array. This is a significant advantage over other O(n log n) algorithms like Merge Sort, which requires additional O(n) space.

**Comparison:** To compare it, I have compared it with both quick sort and merge sort by a random array and a given array. Then, compare it with the time and space taken.



**Figure 1**: Time and Space Comparision with Quick Sort and Merge Sort

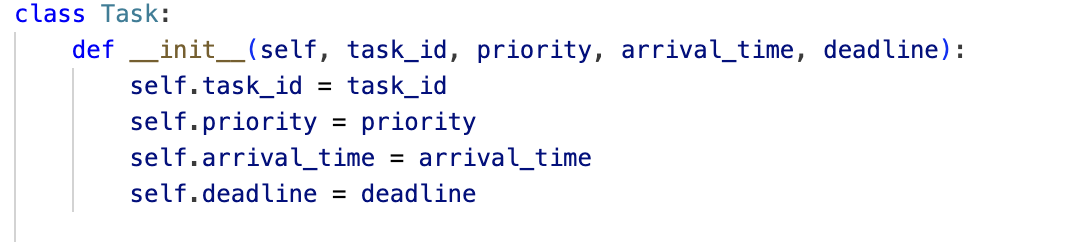
The picture above shows that it takes less time and space than the merge sort and is almost the same amount of time as the quick sort. The time is almost the same in all cases, showing that the time complexity is always the same: O (n log n). This may be due to the repeated typifying after each swap.

**Priority Queue Implementation and Applications**

**Priority Queue:** A priority queue is a type of queue that arranges elements based on their priority values. Elements with higher priority values are typically retrieved or removed before elements with lower priority values. Each component has a priority value associated with it. When we add an item, it is inserted in a position based on its priority value. A priority Queue is an extension of the queue with the following properties.

* Every item has a priority associated with it.
* An element with high priority is dequeued before an element with low priority.
* If two elements have the same priority, they are served according to their order in the queue.

**Data Structure:** A priority queue can be implemented efficiently using a binary heap, where each node has a priority value. I will use a max-heap so that the task with the highest priority is at the top and can be accessed quickly.

**Figure 2:** Task Class Definition

**Task Class:** This task class will hold information about each task, including its ID, priority, arrival time, and deadline.

**A screenshot of a computer program

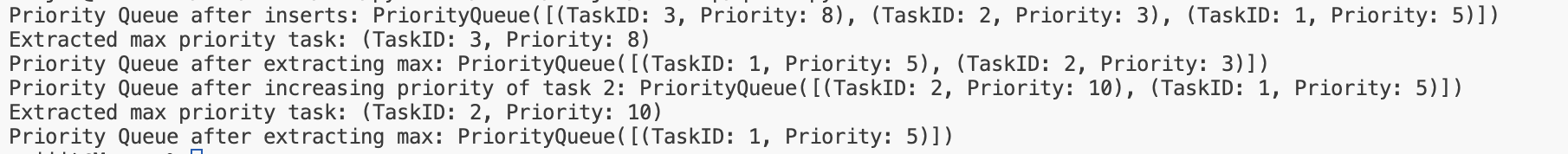
Description automatically generatedFigure 3:** Definition of the Required Methods

**Priority Queue Operation:**

* Insert (task): It will add a task to the heap and maintain the max-heap property.
* extract\_max(): This function will remove and return the task with the highest priority.
* increase\_key(task, new\_priority): Increases the priority of a task and adjusts its position in the heap.
* is\_empty(): Checks if the priority queue is empty.

**Complexity Analysis**

* **Insert Operation:** O(logn) because inserting an element into a heap involves adding it to the end and then "bubbling up" to maintain the heap property.
* **Extract Max Operation:** O(logn) because it involves removing the root of the heap and then "bubbling down" the last element to maintain the heap property.
* **Increase Key Operation:** O(n) because we rebuild the heap after updating the priority of a task. This could be optimized with a more complex data structure, but it is O(n) for simplicity.

**Result Analysis:**

**Figure 4:** Scheduling Output

**A computer code with text

Description automatically generated with medium confidence**From the figure above, we can see that the priority queue printed as it is expected. The maximum priority is at the first, and the minimum priority is at the last of the list since the heap used a max heap approach.

**Figure 5:** Various Task Implementation

Then, **Figure 4** shows that it extended the max priority task, increased the priority with ID #2, and removed the max priority by extracting it. F**igure 5** shows how I have implemented the technique to extract and increase the priority of a task.

**Reference:**

Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1974). *The design and analysis of computer algorithms*. Addison-Wesley.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). The MIT Press.

GeeksforGeeks. (n.d.). *Heap sort*. GeeksforGeeks. Retrieved November 10, 2024, from [https://www.geeksforgeeks.org/heap-sort/#](https://www.geeksforgeeks.org/heap-sort/)

GeeksforGeeks. (n.d.). *Priority queue (Set 1) | Introduction*. GeeksforGeeks. Retrieved November 10, 2024, from [https://www.geeksforgeeks.org/priority-queue-set-1-introduction/#](https://www.geeksforgeeks.org/priority-queue-set-1-introduction/)

Goodrich, M. T., & Tamassia, R. (2010). *Data structures and algorithms in Java* (6th ed.).

Knuth, D. E. (1998). *The art of computer programming: Sorting and searching* (Vol. 3). Addison-Wesley.

Weiss, M. A. (2013). *Data structures and algorithm analysis in C++* (4th ed.). Pearson.