1. **Are there any issues concerning the fair treatment of the tasks?**

In the current implementation, tasks are treated somewhat based on their priority. High-priority tasks are processed before normal-priority tasks. This ensures that essential tasks are addressed promptly, which is generally a fair approach using the specification of how the management system should work. However, to implement the task efficiently, I had to do extra work to ensure I kept the task with high priority on the top of the stack. We could use other data structures that can provide the task with the highest priority is always at the top so that it is processed first.

1. **Do we need stack and queue both to solve the given problem?’**

It is not necessary that we must stack and queue. There might be another approach that may work better than this because I had to create two objects, stack and queue, which takes overheads that are not necessary. However, concerning the specification given, stack and queue made the job easier to accomplish because I think that if we were to use one data structure to handle the high priority and the typical priority tasks, It would be more mindboggling when it comes to making sure that the tasks with high priority are treated first while keeping the normal tasks in their order. Thus, combining stack and queue made the system easier to implement.

1. **How are such systems usually built in practice? What data structures do they use?**

Task management systems in practice often use more complex data structures and databases to manage tasks efficiently. These systems may involve databases to store and retrieve tasks, advanced data structures for efficient task scheduling, and user interfaces for interaction. The use of stacks and queues, as employed in the program, are simplified examples and may need to be revised for complex real-world scenarios. The current system we built could not retrieve tasks after the program had been closed. It would be challenging in any practical sense.

1. **Is the best solution still using stacks and queues for fair treatment on task processing?**

The current solution using stacks and queues is a reasonable approach for maintaining fairness based on priority, although it requires extra work to ensure that the task with high priority is always at the top. However, in practice, you might encounter more complex scenarios where other data structures and algorithms, like a priority queue (heap), are better suited for ensuring fairness and efficient task processing.

1. **Can the application be built using an alternative data structure, such as Deque alone, instead of a stack and queue? If so, describe the changes you would make to the line numbers in your code.**

Yes, you can use a Deque to implement this system without separate stack and queue structures.

I created two classes, Queue and Stack( Line 85 till 242). I would have to create one class called Dequeue. I should make sure that all normal priority tasks are added using the tail and all high-priority tasks are inserted using the tail. I would then create an instance variable that tracks the length of high-priority priority starting from the tail and the size of the normal-priority task starting from the head.

For processing, I would make sure to keep the high priority sorted in descending order so that every time we process any task, we use the tail always, and by the time that the high priority is finished, we will process the normal task form the oldest added to the new recently added.