Explanation and Justification of Approach

The first task was to ensure that a specified number of tasks were run concurrently with the handling of the active ones.

Approach Explanation

The approach to solve this problem is setting the list of the tasks defining the maximum concurrent activity level with the help of the init function, and getting the current time with the help of the **getCurrentConcurrency** function. In this function, the concurrency level is returned 2 in peak time, 9 am – 8 pm, and 4 in the off-peak time.

The **manageConcurrency** function includes all the core logic for the solution of the problem and provides an organizational structure of the task execution by working with the **list** of ***active promises***. It applies a while loop to the end of the tasks so that all the tasks in **tasksList** will be iterated. Within this loop, it controls the current concurrency level by placing new task promises to the portion activePromises until the maximum level of concurrency is achieved. It then waits for the completion of any one task using ***Promise.race()*** before adding more tasks to maintain the concurrency level.

Considered and Rejected Methods

One method that required significant effort was using ***Promise.all()*** Many searches pointed towards this method as a solution, and it was initially implemented successfully. However, it did not align with the expected output due to its nature of working on an iterable list of promises and returning only a single promise. Additionally, ***Promise.all()*** rejects immediately if any of the input promises are rejected, which is not suitable for the desired functionality.