1. Explain the allocation and use of each shared memory segment

The program allocates two shared memory segments, one is for storing the input array and one is a temporary buffer for merging. The input array is initialized in shared memory and accessed by child processes for sorting, and the temporary shared memory holds intermediate merge results before being copied back to the main array.

2. Are there potential race conditions (i.e., processes use and update a shared data item concurrently) in your program?

Yes, I think that potential race conditions exist in the program because multiple processes access and modify the shared memory concurrently during the merge phase. We learned that, if two processes attempt to update overlapping sections of shm simultaneously it could lead to data corruption or incorrect sorting. I used wait() in main.c to make sure I have it under control.

3. Why you should not save the assigned array entry back to the given array in the binary merge phase? State explicitly your reason.

I should not save the assigned array entry back to the given array in the binary merge phase because I think it might overwrite data that has not yet been processed and could lead to incorrect sorting results, since merge phase relies on comparing values from two subarrays and modifying the original array while merging.

4. Explain how you allocate a temporary array to hold the intermediate result in each execution of Mergesort().

Temporary shared memory segment is allocated using shmget with a unique key to store intermediate results during the merge phase where each merge process attaches to this shared memory and writes intermediate sorted values, then transfers them back to the main shared memory segment after merging is complete.

5. We assigned a process to each array entry every time when M e r g e s o r t ( ) is executed to perform binary merge. Then, M e r g e s o r t ( ) waits for all of these processes before terminates itself. As a result, we repeatedly create and terminate n processes log2 n times, which is a waste of time and system resource. Suppose you are allowed to use busy waiting. Can you only create n processes at the beginning of the MERGE phase so that they can be used in each binary merge? State your answer as clearly as possible.

Yes, I think that we can create n processes at the beginning of the merge phase and have them persist throughout the entire merging process instead of creating, every single time, and terminating processes for each merge step, these processes can synchronize using shared memory waiting for their turn to merge assigned sections.