My initial thought into how to sort out the problem (before sitting down and coding) was to go with an array based approach to the problem wherein some memory at that beginning of the array was allocated to Positions of the Queues and then to do some in place organization of the data in memory as we went. This would have involved allocating ~3 bytes or memory (2^11 \* 2 for pointers + 2 bits for logic purposes) and then implementing some form of hash or binary placement algorithm to minimize collision on Queue Creation and Enqueuing.

On my second read through, again before coding, I noticed that the specification was on Worst Case Speed and not Average Case speed, which threw out my original thought process as the worst-case scenario is having to move Arrays around at O(N). So, I went back and envisioned it as a group of storage blocks connected to each other via a basic Linked List (in this case a circular list), splitting each block into a group of 15 bytes of Data and a pointer (7 bit in this case) to the next block. This is better for the specifications since insertion and removal from a list is mostly trivial, even in worst case, though in this case there is a small amount of copying that must be done occasionally. Due to this, the maintenance of free portions of the data is simple, as the memory is split into storage units ahead of time, and allows the list of queues to exist in a state where the worst-case scenario of adding or removing from the queues causes only a minor amount of pointer switching and 14 bytes either being allocated.