Alan Garcia - Merge Sort

1. Explain what you think the worst-case, big-Oh complexity and the best-case, big-Oh complexity of merge sort is. Why do you think that?

The beautiful thing about merge sort is that no matter how the input array is arranged, Big-oh complexity will be exactly the same.

Why?

We are splitting the input array in half each time until we create a subarray that is of size 1. Note that this doesn't happen chronologically. It's always a constant calculation that depends on the size of N. How the array is sorted is irrelevant.

Each time a merge() is called it takes around O(n) to join two subarrays and sort them.
That's why this algorithm is O(n * Logn), note that this is a very high level explanation and is a **very rough estimation.**

2. Explain what you think the worst-case, big-Oh complexity and the best-case, big-Oh complexity is for this iterative merge sort. Why do you think that?:

On paper both recursive and iterative mergeSort are Big O(nlogn), because they are performing mostly the same number of calculations. It must be noted that the iterative approach has a bottoms up strategy, instead of starting with the whole array it divides it into bigger and bigger subarrays. Furthermore, if the array is not even , then there is some extra logic used to sort the array[m] value. But either way, they should perform similarly to one another.

In the real world:
My small research indicates that is something that is highly dependable on the implementation. On some languages and machines the recursive cost is higher, meanwhile on other cases iterative merge sort is way cheaper and faster. The general consensus is that the recursive approach is better.