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?

Both best/worst case time complexity of merge sort is O(nlogn) since it needs to divide the array into half and merge with linear time.

For space complexity O(n), it requires extra space for auxiliary array for left and right intermediate array; if recursion implementation, the recurrence call stack will also cost extra space O(logn).

2. Merge sort as we have implemented in there is a recursive algorithm as this is the easiest way to think about it. It is also possible to implement merge sort iteratively:

```
// Iteratively sort subarray `A[low…high]` using a temporary array
void mergesort(int A[], int temp[], int low, int high)
{
    // divide the array into blocks of size `m`
    // m = [1, 2, 4, 8, 16…]
    for (int m = 1; m <= high - low; m = 2 * m)
    {
        // for m = 1, i = 0, 2, 4, 6, 8…
        // for m = 2, i = 0, 4, 8…
        // for m = 4, i = 0, 8…
        // …
        for (int i = low; i < high; i += 2*m)
        {
            int from = i;
            int mid = i + m - 1;
            int to = min(i + 2*m - 1, high);

            merge(A, temp, from, mid, to);
        }
    }
}
```

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?

for worst case & best case time complexity, it’s the same as the recursive version since it still needs to sort sub-arrays of size 1 then merge into size 2, then merge size 2 into size4 …merge n/2 into size n which is O(nlogn).

but if interested, can add extra test check if array is already sorted to reduce the time complexity to O(n)

But for space complexity, it still needs extra auxiliary temp array O(n) to store intermediate values. just save the recursive call stack O(logn) therefore space complexity is still O(n)