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

   The worst-case time complexity of merge sort is O(n log n), and the best-case time complexity is also O(n log n). This is because merge sort always divides the input into halves and then recursively sorts them. The merging step takes O(n) time for each level of recursion, and there are log n levels of recursion. Therefore, the total time complexity is O(n log n) in both the worst and best cases.

2. Merge sort, as we have implemented it, has a recursive algorithm embedded in the code 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-O complexity and the best-case, Big-O complexity is for this iterative merge sort. Why do you think that?

The worst-case time complexity of the iterative merge sort is O(n log n), which is the same as the recursive version. The best-case time complexity is also O(n log n), which occurs when the array is already sorted. This is because the algorithm always divides the array into halves, and then merges them in linear time. The outer loop iterates log n times, and the inner loop iterates n times, resulting in a total time complexity of O(n log n).