Exercise 2.3.13: What is the recursive depth of quicksort, in the best, worst, and average cases? This is the size of the stack that the system needs to keep track of the recursive calls. See Exercise 2.3.20 for a way to guarantee that the recursive depth is logarithmic in the worst case.

Solution: The question is asking for recursion depth, it is not asking for the general QuickSort time complexity. The best depth is achieved when the recursion tree is balanced, i.e., if each partition halves the array; then, its depth is lgN. In the average case you expect the two parts in each partition to be about the same size, hence the recursion tree is again $\sim clgN$ for some constant $c \geq 1$. In the worst case, every partition has an empty side, thus forcing to recurse on subproblems reduced in size by just a single element; hence the depth of the recursion is linear.

For general QuickSort complexity, the worst case is $O(n^2)$. This occurs on cases where the array is already sorted(ascending or descending order) but also occurs when the array is full of items that are all the same, ie [5,5,5,5,5,5]. This is because the algorithm pivot gets stuck comparing all of them and ends up reading each element along with the recursively broken down elements. This results in $O(N^2)$ complexity due to the comparisons and swaps.

This worst case can be changed so that we get *expected* worst case to match that of avg case. We can do this by shuffling our array to make it random, so that the adversary can no longer pick an input for decreasing the performance of the algorithm. So the worst and avg case becomes $nlogn$ in this case due to randomness. If shuffling was not done, then worst case would remain as $O(n^2)$. Note, expected worst case is NOT the same as actual worst case because worst case still has possibility of appearing as $O(N^2)$.

Best case occurs when the recursive steps can divide the array exactly in half. SO the pivot goes in the middle.

Summary:
MergeSort is great but missing "in-place" sorting. It is however very stable. A stable algorithm always maintains the relative order of the values at their indices. The values stay in the same place with reference to each other and without much movements.

Quicksort is not stable but has its avg and best case happening more often. Therefore, it is often that Quicksort is best choice.