1.Explain what you think the worst-case big O complexity and the best-case big O complexity of bubble sort is as implemented in our code. Give reasons for why you think that is the big O complexity for each case.

Worst case is O(n**2), and the best case is O(n).
In worst case, the array is in reverse order, the algorithm need to go through two nested for loop, each of the for loop is O(n), two nested for loop is O(n**2)
In best case, the array is already sorted, the algorithm only need to compare each element with its next element to sort the array, there is n comparsions needed, so time complexity is o(n).

2.Is there a more efficient way to write bubble sort that changes the performance in the best case? If so, describe (in general terms, we don't need the exact C code) how that implementation of bubble sort would be different.

Yes there is a more efficient way to change best cases performance. We can add an if statement in two nested loops; if we find that any two adjacent elements are not sorted, we continue the loop. If we find that there are no unsorted adjacent elements, we can end the nested loop beforehand to change performance in best cases.

3.Explain what you think the worst-case big O complexity and the best-case big O complexity of selection sort is as you've implemented it. Why do you think that?

The worst case of selection sort is O(n**2) and best case is also O(n**2), this is because not matter what input array is we need to iterate through array which is o(n), and inside the for loop there is find minimal which is also O(n), so the result is always O(n**2)

4.Does selection sort require any additional storage (i.e. did you have to allocate any extra memory to perform the sort?) beyond the original array?
No selection sort is in place algorithm, so there is no additional storage.

5.Would the big O complexity of any of these algorithms change if we used a linked list instead of an array? Why or why not?

They are still both O(n**2), because the idea of bubble sort and selection sort is still the same, they still have same numbers of operations.

6.Explain what you think big O complexity of sorting algorithm that is built into the c libraries is. Why do you think that?

I think c lib's algorithm is O(n log n), which is the same time complexity as quick sort and merge sort [1], and its significantly faster than O(n**2) algorithm.

Reference

[1] Merge sort vs. quicksort: Algorithm performance analysis. [Online]. Available: https://www.interviewkickstart.com/learn/merge-sort-vs-quicksort-performance-analysis#:~:text=The%20time%20complexity%20of%20merge,n2)%20in%20the%20worst%20case. [Accessed: 09-Feb-2023].