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

Best case: array is already sorted in ascending order, then bubble sort only need one pass to find that no element needs to swap. O(N), n is the number of elements within array

Worst case: array is in descend order then during each iteration it needs to swap for every element, n, n-1, n-2…1, in total n(n-1)/2 operations, therefore O(N^2) for worst case.

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

Best case: array is already sorted in ascending order, selection still need to traverse to find the minimum for each iteration, n, n-1…1 therefore O(N^2)

Worst case: the same logic as best case, O(N^2)

3. 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?

since it's in-place sorting therefore it doesn't require additional storage.

4. Would the big-Oh complexity of any of these algorithms change if we used a linked list instead of an array?

For bubble sort, swap operation is the same for both linked list and array therefore complexity is the same.

For selection, it's the same.

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

Even though it uses quick sort but according to the documentation of C library, there is no performance guarantee about qsort. Normal quick sort complexity is O(nlogn), compared with sort() func which use introsort(hybrid sorting, introsort, quick sort and insertion sort) complexity is O(nlogn) with guarantee in C11, qsort() is slower than sort().