Alan García HW 6 answer sheets.

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?

   The worst case for bubble sort is definitely when the list goes from biggest to lowest, bubble sort will have to traverse the complete list -1(cumulative). n*n = O(n^2)

   Best case for bubble sort is when the numbers are all in order, that is when it only takes O(n) because bubble sort checks if one is less than the other, if not it stops.

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?

   Selection sort always performs more or less the same steps, because it always looks for the minimum number that it can find. That takes n - 1 cumulative for each element, therefore regardless of how the array is presorted, it will always take 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?

   No, it was an in place algorithm, I didn't have to copy anything. We used the same array to do our minimum value search

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

   I don't think so, because in all of the cases we looked for values linearly one after the other, random access to the array was very minimal (except for csort, because I don't know for sure what algorithm is performing).

   Swapping also does not resize the array, it actually performs random access, which is not done in linear time using a linked list. It would make it worse.

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

   I think it's quicksort. Thanks to the data we gathered, it appears to have a big O of O(logn) or O(nlogn) . The fastest algorithms that I know of are quicksort and mergesort.

   Mergesort is Big(NlogN) but requires a lot of memory because it has many recursion calls. That would not be very practical and it would often cause a stack overflow on embedded devices.

   On the other hand,quicksort is the most practical one. Although, standard quicksort has a remarkably large worst case of O(n^2). If we use a random pivot then we can have a somewhat constant big O (nlogn) without too many recursive calls.