1. Derive the formula for worst-case complexity of quicksort.

Worst case complexity occurs when the pivot is always chosen to be the smallest or largest value in the array.
This can happen if the pivot is selected to be the first or last element of an array if said array is sorted. In this case, each set of quicksort calls will result in an empty array and an array just one element smaller than before, which results in a total of n recursive calls.
The partition function will have a linear time complexity, or $O(n)$, since it iterates linearly from the start and end of the array without overlapping or redundant swaps (hence the condition of the loop being while i < j).
Since there are n recursive calls, and each recursive call results in $n-1$ recursive calls plus a linear complexity partition call, the complexity can be represented as follows:
[1] $C(n)=C(n-1)+O(n)$, where $C(n)$ is the complexity of the quick sort and $O(n)$ is the complexity of the partitioning step. Then, $C(n)=C(n-1)+n, C(n-1)=C(n-2)+(n-1)...C(2)=C(1)+2\ldots$ This results in an arithmetic sequence: $C(n)=n+(n-1)+(n-2)+(n-2)+...+4+3+2$ which simplifies to $\frac{n(n-1)}{2}\Rightarrow n^2$, therefore the worst-case complexity of quick sort is $O(n^2)$.

2. Come up with a vector of 16 elements which incurs worst-case complexity. Manually show the workings of the algorithm until the vector is sorted.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

This vector will incur worst-case complexity if the first or last element of the subarray is chosen as the pivot. I will be assuming the first element is chosen as the pivot.

Partition is called on the entire array.
Pivot = 0
Since there is no element lower than 0, i >= j and the partition function returns without swapping.
Return value = 0

Quick sort is called with low = 0 and high = -1 which returns since low >= high.
Quick sort is called with low = 1 and high = 15

Partition is called on the [1, 15] subarray
Pivot = 1
Since there is no element lower than 1, i >= j and the partition function returns without swapping.
Return value = 1

Quick sort is called with low = 1 and high = 0 which returns since low >= high.
Quick sort is called with low = 2 and high = 15

[1]
S. Datta, “Quick Sort Worst Case Time Complexity | Baeldung on Computer Science,” www.baeldung.com, Aug. 11, 2020. https://www.baeldung.com/cs/quicksort-time-complexity-worst-case

Now to speed up:
partition called on [2, 15], no swaps
quicksort called on [2, 1], returns
quicksort called on [3, 15]

partition called on [3, 15], no swaps
quicksort called on [3, 2], returns
quicksort called on [4, 15]

partition called on [4, 15], no swaps
quicksort called on [4, 3], returns
quicksort called on [5, 15]

partition called on [5, 15], no swaps
quicksort called on [5, 4], returns
quicksort called on [6, 15]

partition called on [6, 15], no swaps
quicksort called on [6, 5], returns
quicksort called on [7, 15]

partition called on [7, 15], no swaps
quicksort called on [7, 6], returns
quicksort called on [8, 15]

partition called on [8, 15], no swaps
quicksort called on [8, 7], returns
quicksort called on [9, 15]

partition called on [9, 15], no swaps
quicksort called on [9, 8], returns
quicksort called on [10, 15]

partition called on [10, 15], no swaps
quicksort called on [10, 9], returns
quicksort called on [11, 15]

partition called on [11, 15], no swaps
quicksort called on [11, 10], returns
quicksort called on [12, 15]

partition called on [12, 15], no swaps
quicksort called on [12, 11], returns
quicksort called on [13, 15]

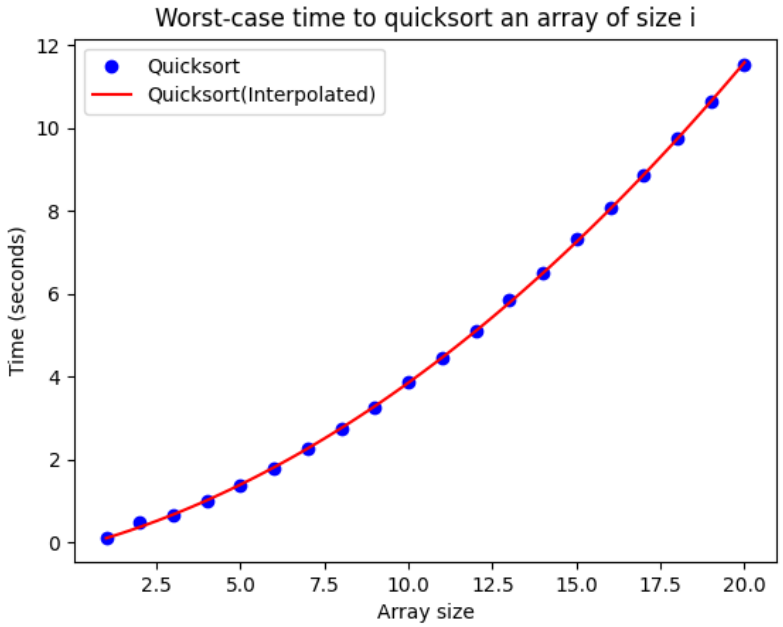
partition called on [13, 15], no swaps
quicksort called on [13, 12], returns

quicksort called on [14, 15]

partition called on [14, 15], no swaps
quicksort called on [14, 13], returns
quicksort called on [15, 15], returns since i == j

Algorithm ends without any swaps being done (Array is already sorted).

4.



The graph shows that the time complexity of quicksort in the worst case is quadratic, or $O(n^2)$, which matches my complexity analysis from question 1 which concluded the same thing.