Quicksort

In quicksort, we first decide on the pivot. This could be the element at any location in the input array. The function Partition is then invoked. Partition accomplishes the following: it places the pivot in the location that it should be in the output and places all elements that are at most the pivot to the left of the pivot and all elements greater than the pivot to its right. Then we recurse on both parts. The pseudocode for Quicksort is as follows.

\[
\text{QSort}(A[lo..hi])
\]

if hi <= lo then
    return
else
    pivotIndex = floor((lo+hi)/2) (this could have been any location)
    loc = Partition(A, lo, hi, pIndex)
    QSort(A[lo..loc-1])
    QSort(A[loc+1..hi])

One possible implementation of the function Partition is as follows.

\[
\text{Partition}(A, lo, hi, pIndex)
\]

pivot = A[pIndex]
swap(A, pivotIndex, hi)
left = lo
right = hi - 1
while left <= right do
    if (A[left] <= pivot) then
        left = left + 1
    else
        swap(A, left, right)
        right = right - 1
swap(A, left, hi)
return left

The worst case running time of the algorithm is given by

\[
T(n) = \begin{cases} 
1, & n = 1 \\
T(n - 1) + cn, & n \geq 2 
\end{cases}
\]

Hence the worst case running time of QSort is \( \Theta(n^2) \).

An instance where the quicksort algorithm in which the pivot is always the first element in the input array performs poorly is an array in descending order of its elements.