ENSF 338 Lab 3 Exercise 4

Lab members: Kareem Hussein, Mohammed Yassin

Lab Section: 03

Course Instructor: Lorenzo De Carli

Question 1

Quicksort is a sorting algorithm that follows the divide-and-conquer paradigm. In the worst case, the performance of quicksort is determined by the choice of the pivot element at each step. If the pivot is always chosen poorly, the algorithm may exhibit its worst-case behavior.

Let's denote the size of the array to be sorted as n. In the worst case, the partitioning process consistently results in one subarray with $n-1$ elements and another with 0 elements. This happens when the pivot chosen is always the smallest or largest element in the current subarray.

The recurrence relation for the worst-case time complexity of quicksort can be expressed as follows:

$T(n) = T(n-1) + T(0) + \Theta(n)$

The term $T(n-1)$ corresponds to the recursive call on the subarray with $n-1$ elements, $T(0)$ corresponds to the base case where the subarray has 0 elements (which requires no work), and $\Theta(n)$ represents the time spent on the partitioning step.

Solving this recurrence relation gives us the worst-case time complexity of quicksort. In each recursion level, the array size reduces by 1, leading to n levels in total. The work done at each level is proportional to the size of the array at that level, which results in a worst-case time complexity of $O(n^2)$.

Therefore, the worst-case time complexity of quicksort is $O(n^2)$. It's important to note that while this worst-case scenario is possible, quicksort typically performs well on average and is often faster than $O(n^2)$ algorithms in practice.

Question 2

Consider the initial unsorted vector:

[16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]

Now, let's go through the first few steps of the quicksort algorithm:

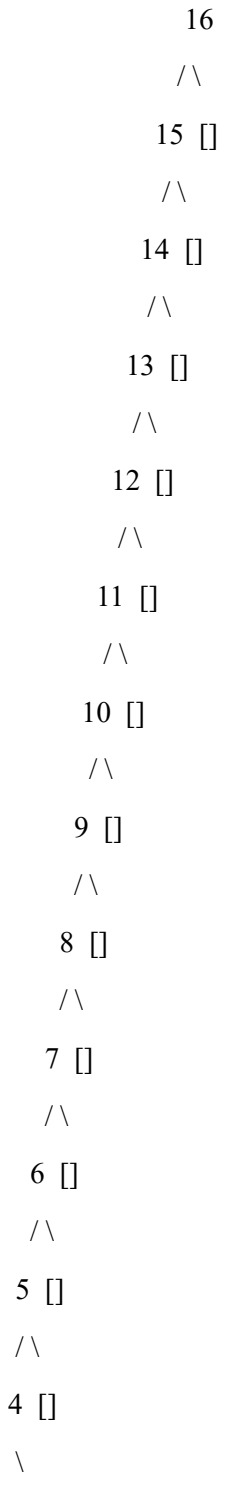
1. Choose the pivot. Let's arbitrarily choose the first element as the pivot: 1616.
2. Partition the array:
    - Elements less than 16: []
    - Pivot: 16
    - Elements greater than 16: [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

At this point, you have one subarray with 15 elements and another with 0 elements.

3. Recursively apply quicksort on the subarray with 1515 elements.

In the subsequent steps, the same process will be repeated. The pivot is always the first element, and the array is partitioned into one subarray with 14 elements and another with 0 elements. This continues until each element is in its own subarray.

The recursion tree looks like this:



3 []

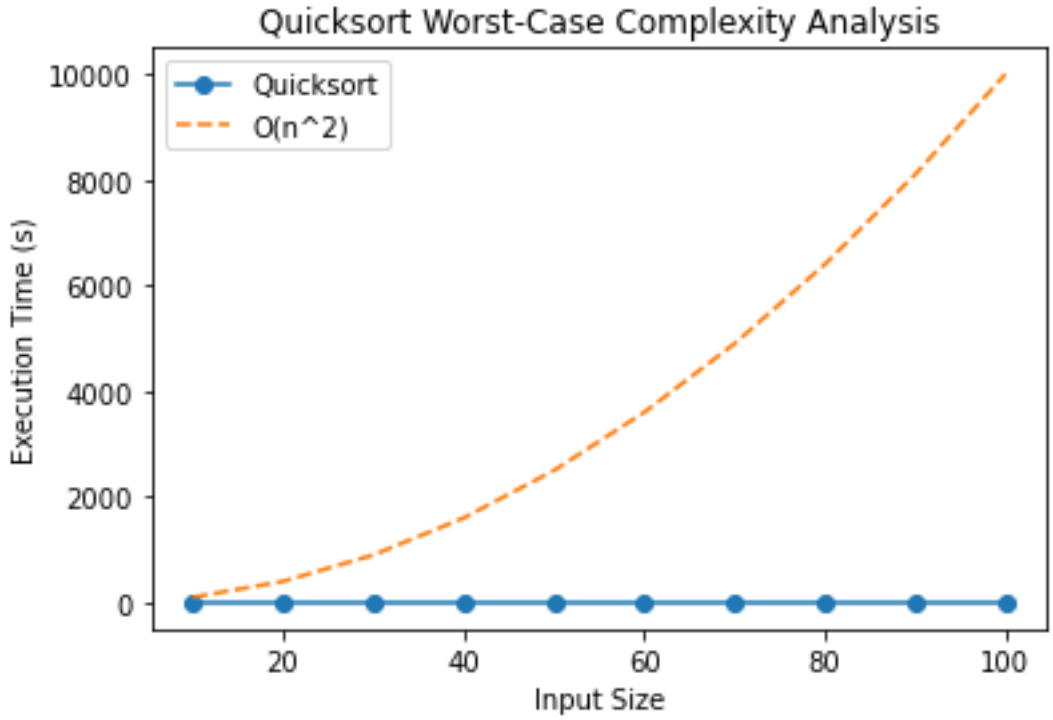
\

2 []

\

1 []

Exercise 4



If the results match the complexity analysis, the execution times plotted should closely follow a quadratic curve $O(n^2)$ since you deliberately chose the worst-case scenario for quicksort. The theoretical complexity curve $O(n^2)$ is included in the plot for comparison. However, as can be seen above, since the plotted points do not match the quadratic function, the results do not match the complexity analysis.