Lab #03

Exercise 4

1. Let T(n) be the time complexity of quicksort for an array of size n. Assuming a worst-case complexity for a quicksort, a bad pivot point was presumably chosen, making it so that the values of the arrays are either entirely on the left or right side of the pivot (smallest or largest value). Thus, the partition will have a size of n-1. Therefore:

$$T(n) = T(n-1) + O(n)$$

Where T(n-1) is the recursive call of quicksort onto the partition and O(n) is the cost of partitioning. Given that T(n-1) must continue to recursively call itself until the partition is divided into it's smallest form, and assuming as with the first case a bad pivot is always chosen, the following is also true:

$$\begin{aligned}
&T(n-1) = T(n-2) + O(n-1)\\
&T(n-2) = T(n-3) + O(n-2)\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&T(1) = O(1)
\end{aligned}$$

The above implies that when there exists only one single value, the only cost present is the partition cost. No need to call upon quicksort recursively at this point. Adding all these terms together:

$$\begin{aligned}
&T(n) = O(n) + O(n-1) + O(n-2) + \ldots + O(2) + O(1)\\
&T(n) = O(1 + 2 + 3 + \ldots + (n-1) + n)
\end{aligned}$$

The arithmetic series above can be simplified into the form:

$$T(n) = O\left(\frac{n(n+1)}{2}\right)$$
$$T(n) = O\left(\frac{n^2+n}{2}\right)$$

Removing the constants and leaving only the highest order of n, we get:

$$T(n) = O(n^2)$$

This is the same as the worst-case complexity for quicksort.

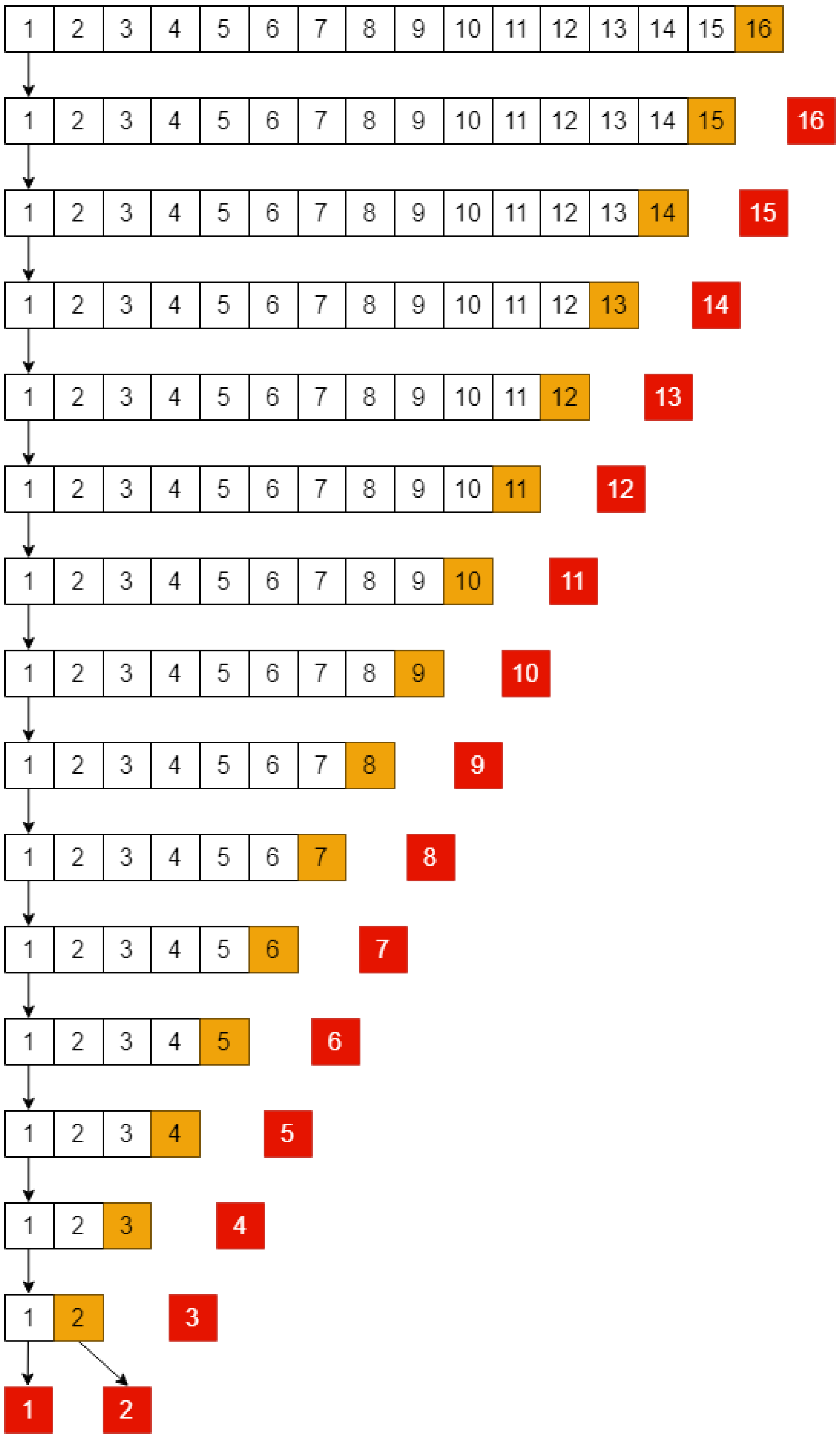
2.

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

This is a vector with 16 elements. Utilizing quicksort, if we were to make either the smallest or largest point the pivot, it would incur the worst-case complexity. This is because the array is already sorted and therefore would always ensure that one partition would remain empty, while the other has n-1 elements (where n is the size of the array/partition if called recursively). For the sake of this example, say we choose the last element [which will always be the largest one] as the pivot point. Thus, for the first pass:

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

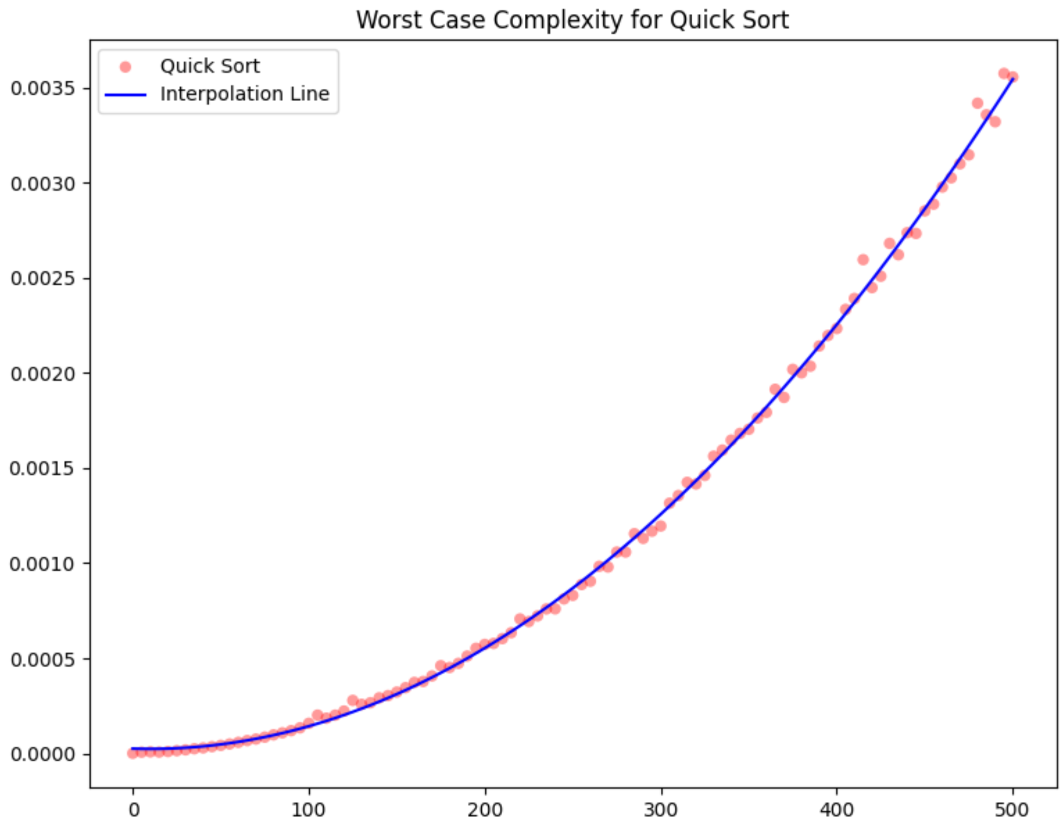
The diagram on page #3 is the result of the full quicksort algorithm being applied. The numbers highlighted in red belong to the array, in their proper order. The numbers highlighted in orange are selected to be pivot points:


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

Once again, note that the order of the array elements remain unchanged, and that a partition was only formed on the left side of the pivot point at all times.

3. See code for implementation

4. The graph below showcases the measurement data recorded, when compared to an interpolation line of $c.n^2$; where c is a constant coefficient and n is the input size. With said coefficient applied, the data recorded matches with the interpolation line, and therefore matches the complexity analysis from earlier. This is an example of $O(n^2)$, the worst-case complexity of quicksort.


Worst Case Complexity for Quick Sort
Quick Sort
Interpolation Line