Assignment 13

Discussion on the correctness and complexity of each version.

For the first version

The correctness of this version is discussed in the textbook, so we will not elaborate on it here.

Time Complexity: $O(n^2)$ when the input is an already sorted array (in any order).

Space Complexity: $O(1)$

For the second version

Algorithm Description:

I Arbitrarily, select the three rightmost elements in each sub-array and choose the element that is neither the minimum nor the maximum. This element will serve as the pivot, and its index will be returned.

Select Pivot(A, end)

```
pivotIndex = end, max = end, min = end
 for i ← 1 to 2      # We find the indices of the max and min
  if A[end − i] > A[max] then
   max = end − i
  if A[end − i] < A[min] then
   min = end − i
 end of for loop

 for i ← 1 to 2   # Determine which index does not point to the max and min
  if end − i ≠ max and end − i ≠ min then
        pivotIndex = end − i
 end of for loop
return pivotIndex
```

II Move this element to the rightmost position and sort the sub-array accordingly.

Partition 2 $(\mathbf{A}, \mathbf{left}, \mathbf{right})$

$\mathrm{p} = \text{Select Pivot}(\mathrm{A}, \text{right})$

$\text{Swap}(\mathrm{A}, \mathrm{p}, \text{right})$

$\text{return Partition}(\mathrm{A}, \text{left}, \text{right})$

III Repeat steps I and II until the sub-arrays contain at most two elements. Sort them (if necessary) and backtrack. Ultimately, the output will be a sorted array.

Quick Sort 2 $(\mathbf{A}, \mathbf{left}, \mathbf{right})$

$\text{if right} - \text{left} + 1 \geq 3 \text{ then}$

$\text{pivot} = \text{Partition 2}(\mathrm{A}, \text{left}, \text{right})$

$\text{Quick Sort 2}(\mathrm{A}, \text{left}, \text{pivot} - 1)$

$\text{Quick Sort 2}(\mathrm{A}, \text{pivot} + 1, \text{right})$

$\text{if right} - \text{left} + 1 = 2 \text{ and A[left]} > \text{A[right]}$

$\text{Swap}(\mathrm{A}, \text{left}, \text{right})$

The correctness of $Select\ Pivot$ stems from the fact that, in the first loop, the indices of the minimum and maximum elements among the three are identified. In the second loop, the index of the remaining element is determined. As a result, the position of the element that is neither the maximum nor the minimum among the three will be returned, ensuring it is also not the maximum or minimum in the sub-array being processed in the current call.

The correctness of $Partition\ 2$ follows from the correctness of $Select\ Pivot$, the operation of swapping the pivot element with the rightmost element in the sub-array, and the correctness of $Partition$, as discussed in the textbook.

The correctness of $Quicksort\ 2$ is derived from the fact that this version behaves like $Quicksort\ 1$ for arrays of size 3 or larger. When the sub-array contains at most two elements (noting that with each call to $Quicksort\ 2$, the sizes of the sub-arrays indeed decrease, as shown in steps 3 and 4 of section III, their sizes will eventually reach 2), the elements are compared, swapped if necessary, and backtracked.

Time Complexity:

The worst-case scenario occurs when, in every call to $Select\ Pivot$, the pivot chosen is the second smallest or second largest element in the sub-array. As a result, in each call, $Partition\ 2$ divides the sub-array into one region containing $n-2$ elements and another containing a single element (the maximum or minimum element of the sub-array).

Thus, the recurrence formula is:

$$T(n) = T(n-2) + T(1) + O(n) + O(1)$$

Note that the time complexity of $Select\ Pivot$ is $O(1)$.

The runtime efficiency of $Partition\ 2$ is the same as $Partition$ from the textbook, which is $O(n)$.

When the size of the array is less than three, the condition is handled in $O(1)$, so:

$$T(1) = O(1)$$

Thus, the recurrence formula becomes:

$$\mathrm{T(n) = T(n-2) + O(n)}$$

We will use the recursion tree method to analyze the recurrence formula:

$$\mathrm{n}$$
$$\downarrow$$
$$\mathrm{n-2}$$
$$\downarrow$$
$$\mathrm{n-4}$$
$$\downarrow$$
$$\dots$$
$$\downarrow$$
$$0$$

We start with a problem of size n, where n units of work are done.
At the next level, the problem size is $n-2$, and $n-2$ units of work are done.
After that, the problem size is $n-4$, and $n-4$ units of work are done, and so on.

The size of the tree is $\frac{n}{2}$ (from the number of steps from the main problem until its completion). In each step j, when $\frac{n}{2} \geq j \geq 0$, we invest $n-2j$ units of work.
In conclusion, the total amount of work done is:

$$\sum_{\mathrm{j=0}}^{\frac{\mathrm{n}}{2}} (\mathrm{n-2j}) = \frac{\frac{\mathrm{n}}{2} * \left(\mathrm{n-2*0+n-2*\frac{n}{2}}\right)}{2} = \frac{\mathrm{n^2}}{4} < \mathrm{n^2} \rightarrow \mathrm{T(n) = O(n^2)}$$

Space Complexity: $O(1)$

as the auxiliary variables are independent of the size of the array.

Total comparisons of $Quicksort\ 2$ divided by the total comparisons of $Quicksort\ 1$	Average number of comparisons for the worst-case scenarios $Quicksort1$	Average number of comparisons for the worst-case scenarios $Quicksort2$	Average number of comparisons for the best-case scenarios $Quicksort1$	Average number of comparisons for the best-case scenarios $Quicksort2$	Average number of comparisons for the average-case scenarios $Quicksort1$	Average number of comparisons for the average-case scenarios $Quicksort2$	n
$\frac{12}{16} = \frac{3}{4}$	$\frac{6}{16} = \frac{3}{8}$	0	$\frac{2}{16} = \frac{1}{8}$	1	$\frac{1}{2} = \frac{8}{16}$	0	3
$\frac{672}{888} = \frac{28}{37}$	$\frac{20}{888} = \frac{5}{222}$	$\frac{432}{672} = \frac{9}{14}$	$\frac{6}{888} = \frac{1}{148}$	$\frac{240}{672} = \frac{5}{14}$	$\frac{862}{888} = \frac{431}{444}$	0	5
$\frac{183}{236}$	$\frac{7}{11328}$	$\frac{12}{61}$	$\frac{5}{33984}$	$\frac{15}{61}$	0.999234	$\frac{34}{61}$	7
$\frac{1852}{2369}$	$\frac{7}{85284}$	$\frac{93}{926}$	$\frac{13}{682272}$	$\frac{375}{3704}$	$\frac{9887}{9888}$	$\frac{2957}{3706}$	8

The table describes the average number of comparisons for the average, best, and worst cases. In each case, the total comparisons for all scenarios of the given type were divided by the total number of comparisons performed.

It can be observed that in all the cases examined, $Quicksort\ 2$ is more efficient than $Quicksort\ 1$, but the trade-off is that the second version has significantly more worst-case scenarios.

Note:

In special cases (as seen for $n = 3$), the given case might simultaneously be both the best and the worst case. This happens when the algorithm divides every sub-array into two equal-sized sub-arrays, but the chosen pivot is the second smallest or second largest value in the sub-array (as explained earlier in the time complexity analysis). In such a situation, we consider the case to be a best-case scenario.

In the best case, the expected runtime for both versions of $Quicksort$ is $nlogn$, while in the worst case, it is n^2.

For $n = 5$

Best Case:

The expected number of comparisons is: $5 \cdot log(5) \approx 12$

Worst Case:
The expected number of comparisons is: $5^2 = 25$

For *Quicksort* 1:

There was 1 best-case scenario and 2 worst-case scenarios, so the total expected comparisons are:

Best-case scenario: 12

Worst-case scenario: $2 * 25 = 50$

Actual results (in comparisons):

Best-case scenario: 6

Worst-case scenario: 20

***For Quicksort* 2**:

There was 48 best-case scenario and 72 worst-case scenarios, so the total expected comparisons are:

Best-case scenario: $12 * 48 = 576$

Worst-case scenario: $72 * 25 = 1800$

Actual results (in comparisons):

Best-case scenario: 240

Relation: $\frac{240}{576} = \frac{5}{12}$

Worst-case scenario: 432

Relation: $\frac{432}{1800} = 0.24$

***For* $n = 7$**

Best Case:
The expected number of comparisons is: $7 \cdot log(7) \approx 20$

Worst Case:
The expected number of comparisons is: $7^2 = 49$

For *Quicksort* 1:

There was 1 best-case scenario and 2 worst-case scenarios, so the total expected comparisons are:

Best-case scenario: 20

Worst-case scenario: $2 * 49 = 98$

Actual results (in comparisons):

Best-case scenario: 10

Worst-case scenario: 42

For Quicksort 2:

There was 1296 best-case scenario and 864 worst-case scenarios, so the total expected comparisons are:

Best-case scenario: $20 * 1296 = 25920$

Worst-case scenario: $49 * 864 = 42336$

Actual results (in comparisons):

Best-case scenario: 12960

Relation: $\frac{12960}{25920} = \frac{1}{2}$

Worst-case scenario: 10368

Relation: $\frac{10368}{42336} = \frac{12}{49}$

For n=8

Best Case:
The expected number of comparisons is: $8 \cdot log(8) \approx 24$

Worst Case:
The expected number of comparisons is: $8^2 = 64$

For *Quicksort* 1:

There was 1 best-case scenario and 2 worst-case scenarios, so the total expected comparisons are:

Best-case scenario: 24

Worst-case scenario: $2 * 64 = 128$

Actual results (in comparisons):

Best-case scenario: 13

Worst-case scenario: 56

For $Quicksort$ 2:

There was 4320 best-case scenario and 3456 worst-case scenarios, so the total expected comparisons are:

Best-case scenario: $24 * 4320 = 103680$

Worst-case scenario: $64 * 3456 = 221184$

Actual results (in comparisons):

Best-case scenario: 54000

Relation: $\frac{54000}{103680} = \frac{25}{48}$

Worst-case scenario: 53568

Relation: $\frac{53568}{221184} = \frac{31}{128}$

Conclusion:

We've observed in the best-case scenarios that both versions of $Quicksort$ achieve $O(nlogn)$ with the ratio of the actual comparisons and the expected comparisons being approximately 0.5. We had also observed that for the worst-case scenarios, both versions achieve $O(n^2)$ with the ratio of the actual comparisons and the expected comparisons being approximately 0.24.