Programming Assignment #3

Quiz, 3 questions

3/3 points (100.00%)

Congratulations! You passed!

Next Item

1 / 1 point

1.

GENERAL DIRECTIONS:

Download the following text file:

QuickSort.txt

The file contains all of the integers between 1 and 10,000 (inclusive, with no repeats) in unsorted order. The integer in the i^{th} row of the file gives you the i^{th} entry of an input array.

Your task is to compute the total number of comparisons used to sort the given input file by QuickSort. As you know, the number of comparisons depends on which elements are chosen as pivots, so we'll ask you to explore three different pivoting rules.

You should not count comparisons one-by-one. Rather, when there is a recursive call on a subarray of length m, you should simply add $m-1$ to your running total of comparisons. (This is because the pivot element is compared to each of the other $m-1$ elements in the subarray in this recursive call.)

WARNING: The Partition subroutine can be implemented in several different ways, and different implementations can give you differing numbers of comparisons. For this problem, you should implement the Partition subroutine *exactly* as it is described in the video lectures (otherwise you might get the wrong answer).

DIRECTIONS FOR THIS PROBLEM:

For the first part of the programming assignment, you should always use the first element of the array as the pivot element.

HOW TO GIVE US YOUR ANSWER:

Type the numeric answer in the space provided.

So if your answer is 1198233847, then just type 1198233847 in the space provided without any space / commas / other punctuation marks. You have 5 attempts to get the correct answer.

(We do not require you to submit your code, so feel free to use the programming language of your choice, just type the numeric answer in the following space.)

1 / 1 point

3/3 points (100.00%)

2.

GENERAL DIRECTIONS AND HOW TO GIVE US YOUR ANSWER:

See the first question.

DIRECTIONS FOR THIS PROBLEM:

Compute the number of comparisons (as in Problem 1), always using the final element of the given array as the pivot element. Again, be sure to implement the Partition subroutine *exactly* as it is described in the video lectures.

Recall from the lectures that, just before the main Partition subroutine, you should exchange the pivot element (i.e., the last element) with the first element.

---

1 / 1 point

3.

GENERAL DIRECTIONS AND HOW TO GIVE US YOUR ANSWER:

See the first question.

DIRECTIONS FOR THIS PROBLEM:

Compute the number of comparisons (as in Problem 1), using the "median-of-three" pivot rule. [The primary motivation behind this rule is to do a little bit of extra work to get much better performance on input arrays that are nearly sorted or reverse sorted.] In more detail, you should choose the pivot as follows. Consider the first, middle, and final elements of the given array. (If the array has odd length it should be clear what the "middle" element is; for an array with even length $2k$, use the k^{th} element as the "middle" element. So for the array 4 5 6 7, the "middle" element is the second one ---- 5 and not 6!) Identify which of these three elements is the median (i.e., the one whose value is in between the other two), and use this as your pivot. As discussed in the first and second parts of this programming assignment, be sure to implement Partition *exactly* as described in the video lectures (including exchanging the pivot element with the first element just before the main Partition subroutine).

EXAMPLE: For the input array 8 2 4 5 7 1 you would consider the first (8), middle (4), and last (1) elements; since 4 is the median of the set {1,4,8}, you would use 4 as your pivot element.

SUBTLE POINT: A careful analysis would keep track of the comparisons made in identifying the median of the three candidate elements. You should NOT do this. That is, as in the previous two problems, you should simply add $m - 1$ to your running total of comparisons every time you recurse on a subarray with length m.
