German University in Cairo
Department of Computer Science
Dr. Nourhan Ehab

CSEN 703 Analysis and Design of Algorithms, Winter Term 2024
Practice Assignment 5

Exercise 5-1

Recall the following QuickSort algorithm discussed in class.

```
1  QuickSort($A, p, r$)
2  if $p < r$ then
3      $q$ = Partition($A, p, r$);
4      QuickSort($A, p, q - 1$) ;
5      QuickSort($A, q + 1, r$) ;
6  end
```

Suppose that the Partition function at line 2 was replaced by the following function Modified_Partition.

```
1  Modified_Partition($A, p, r$)
2  $x = A[p]$;
3  $i = p$;
4  $j = r$;
5  while $TRUE$ do
6      while $j > p$ and $A[j] \geq x$ do
7          $j = j - 1$;
8      end
9      while $i < r$ and $A[i] \leq x$ do
10         $i = i + 1$;
11     end
12     if $i < j$ then
13         Exchange $A[i]$ with $A[j]$;
14     else
15         Exhange $A[p]$ with $A[j]$;
16         return $j$;
17     end
18 end
```

(a) Demonstrate the operation of Modified_Partition when called with $A = [13, 19, 9, 5, 14, 8, 7, 4, 21]$, $p = 1$, and $r = 9$. Show the values of the array after each iteration of the while loop in lines 5-18 and the final return value.

(b) Explain the functionality of Modified_Partition.

Solution:

The `Modified_Partition` function is a variation of the standard partitioning algorithm used in QuickSort. It partitions the array around the pivot element $A[p]$ such that:

- All elements to the left of the pivot (from p to $j - 1$) are less than or equal to the pivot.
- All elements to the right of the pivot (from $j + 1$ to r) are greater than or equal to the pivot.

The function moves i from the left and j from the right, ensuring that smaller elements are on the left and larger ones on the right. Once $i \geq j$, the pivot is swapped with $A[j]$ and j is returned as the partition index.

Correctness Proof (You do not need to prove correctness here but the proof is provided as supplementary material.):

Invariant: $A[p..i-1] \leq$ pivot, $A[j+1..r] \geq$ pivot, and if $i \geq j$, $A[j] =$ pivot.

- Initialization: Before the first iteration of the (outer) while loop, $i = p$ and $j = r$, so $A[p..p-1]$ and $A[r+1..r]$ are the empty subarray, so the invariant is not violated.
- Maintenance: i is incremented as long as $A[i] \leq$ pivot and j is decremented as long as $A[j] \geq$ pivot. Therefore, when the two inner loops terminate and $i < j$, the array is divided into the 3 subarrays described below, such that i is pointing at the first element greater than the pivot and j is pointing at the last element smaller than the pivot.
  - $A[p..i-1]$ which contains elements smaller than or equal to the pivot,
  - $A[j+1..r]$ which contains elements greater than or equal to the pivot, and
  - $A[i..j]$ which contains elements that are not scanned yet.

  The elements at i and j are then swapped together to prepare for the next iteration.
- Termination: When $i \geq j$, the loop terminates as j points to the location of the pivot in the sorted array; this is because the inner while loop that moves j terminates when j is pointing to the last element smaller than the pivot (or to the first duplicate of the pivot if the array if filled with duplicates). The pivot is then swapped with the element at index j and the loop invariant remains true.

(c) Is the modified QuickSort algorithm correct when it uses Modified_Partition? Explain your reasoning.

Solution:

Yes, the modified QuickSort is correct. The `Modified_Partition` ensures that the array is partitioned into two subarrays, where:

- The left partition contains elements smaller than or equal to the pivot.
- The right partition contains elements larger than or equal to the pivot.

This is sufficient for QuickSort to work, as recursive calls on both partitions will sort the array correctly.

(d) What are the best and worst cases of the modified QuickSort?

Solution:

- **Best case:** The best case occurs when the pivot chosen divides the array into two nearly equal halves. This happens when the pivot is close to the median of the subarray.
- **Worst case:** The worst case occurs when the pivot is the smallest or largest element in the array, leading to highly unbalanced partitioning and recursion depth equal to the size of the array.

(e) Write the recurrences representing the best and worst case running times of the modified QuickSort (you don't need to solve them).

Solution:

- **Best case recurrence:**
$$T(n) = 2T(n/2) + O(n)$$
- **Worst case recurrence:**
$$T(n) = T(n-1) + O(n)$$