Exercise Sheet 5

Handout: Oct 12th — **Deadline:** Oct 19th, 4pm

Question 5.1 (Marks: 0.25)

Illustrate the operation of QUICKSORT on the array

4	**3**	**8**	**2**	**7**	**5**	**1**	**6**

Write down the arguments for each recursive call to QUICKSORT (e. g. "QUICKSORT(A, 2, 5)") and the contents of the relevant subarray in each step of PARTITION (see Figure 7.1). Use vertical bars as in Figure 7.1 to indicate regions of values "$\leq$ x" and "$>$ x". You may leave out elements outside the relevant subarray and calls to QUICKSORT on subarrays of size 0 or 1.

Answer:

QUICKSORT(A,1,8)
PARTITION(A,1,8)
return q=6

4	**3**	**2**	**5**	**1**	**6**	**7**	**8**
smaller					pivot	larger	

QUICKSORT(A,1,5)
PARTITION(A,1,5)
return q=1

1	**3**	**2**	**5**	**4**	**6**	**7**	**8**
pivot	larger				mask		

QUICKSORT(A,1,0)
QUICKSORT(A,2,5)

PARTITION(A,2,5)
return q=4

1	3	2	4	5	6	7	8
mask	smaller		pivot	larger	mask		

QUICKSORT(A,2,3)
PARTITION(A,2,3)
return q=2

1	2	3	4	5	6	7	8
mask	pivot	larger	mask				

QUICKSORT(A,2,1)
QUICKSORT(A,3,3)
QUICKSORT(A,5,5)
QUICKSORT(A,7,8)
PARTITION(A,7,8)
return q=8

1	2	3	4	5	6	7	8
mask						smaller	pivot

QUICKSORT(A,7,7)
QUICKSORT(A,9,8)

now the array is sorted.

Question 5.2 (Marks: 0.5)

Prove that deterministic QUICKSORT(A, p, r) is correct (you can use that PARTITION is correct since that was proved at lecture).

Answer:

Given PARTITION is correct, we can use induction to prove that QUICKSORT is correct.

BASE CASE: QUICKSORT(A, p, r) can correctly sort an subarray with only one element, since QUICKSORT(A, 1, 1) do not recursively call itself and the subarray is sorted naturally.

INDUCTIVE HIPOTHESIS: Assume that QUICKSORT(A, p, r) can correctly sort subarrays A[p . . . r] for all size n $\leq$ k.

INDUCTIVE STEP: When we call QUICKSORT(A, p, r) with size k+1, first it calls PARTITION(A, p, r) to split the array into two subarrays and returns q, and A[q] (pivot) is not smaller than any elements in A[p, q-1] and not larger than any elements in A[q+1, r]. Then it calss QUICKSORT(A, p, q-1) and QUICKSORT(A, q+1, r) on each two subarrays, whose sizes are at most k, and can be correctly sorted due to the inductive hypothesis. The smaller subarray and the larger subarray are sorted, and the pivot is in the correct position, so QUICKSORT(A, p, r) can correctly sort the array.

Question 5.3 (Marks: 0.25)

What is the runtime of QUICKSORT when the array A contains distinct elements sorted in decreasing order? (Justify your answer)

Answer:

The pivot is the last element of the array, which is the smallest in this scenario. So when partitioning, no elements need to swap and the return value q whould be 1. Then we break the problem into a empty array and a subarray of size n-1. Essentially the recurrence equation is $T(n) = T(n-1) + \Theta(n)$, which gives us $\Theta(n^2)$.

Question 5.4 (Marks: 0.5)

What value of q does PARTITION return when all n elements have the same value?
What is the asymptotic runtime (Θ-notation) of QUICKSORT for such an input? (Justify your answer).

Answer:

If all n elements have the same value, then q would be the pivot's position r, because i increments and do the swap all the way with j, which stop at r-1, then return r.

The runtime of QUICKSORT when all n elements have the same value also $\theta(n^2)$ because we split the problem into size n-1 and size 1, the recurrence equation is $T(n) = T(n-1) + \Theta(n)$, which gives us $\Theta(n^2)$.

Question 5.5 (Marks: 0.5)

Modify PARTITION so it divides the subarray in three parts from left to right:

- A[p . . . i] contains elements smaller than x
- A[i + 1 . . . k] contains elements equal to x and
- A[k + 1 . . . j – 1] contains elements larger than x.

Use pseudocode or your favourite programming language to write down your modified procedure PARTITION' and explain the idea(s) behind it. It should still run in $\theta(n)$ time for every n-element subarray. Give a brief argument as to why that is the case. PARTITION' should return two variables q,t such that A[q . . . t] contains all elements with the same value as the pivot (including the pivot itself).

Also write down a modified algorithm QUICKSORT' that uses PARTITION' and q, t in such a way that it recurses only on strictly smaller and strictly larger elements.

What is the asymptotic runtime of QUICKSORT' on the input from Question 5.4?

Answer:

```
pair<int, int> PARTITION’(vector<int> &arr, int p, int r)
{
    if p >= q{
        return {lt, gt};
    }
    int pivot = arr[r];
    int lt = p;      // pointer for elements < pivot
    int gt = r-1;      // pointer for elements > pivot
    int i = p;       // current pointer

    while (i <= gt) {
        if (arr[i] < pivot) {
            swap(arr[lt], arr[i]);
            lt++;
            i++;
        }
        else if (arr[i] > pivot) {
            swap(arr[i], arr[gt]);
            gt--;
        }
        else {  // arr[i] == pivot
            i++;
        }
    }
    swap(arr[gt], arr[r]);
    return {lt, gt};
}
```

In order to implement the modified algorithm which run in $\Theta(n)$ time for every n-element subarray, we need to finish the procedure only in one pass. Hence, we create two pointers lt and gt, to separate smaller and larger subarrays in the pass while leave the middle of the array for elements equal to pivot. They are initialized as the first and second last element of the array (the last regardless of the pivot). When the current pointer i is smaller than pivot, we swap it with the element at pointer lt, and move both pointers to the next position. When the current pointer i is larger than pivot, we swap it with the element at pointer i and move lt to the previous position. When the current pointer i is equal to pivot, we just move i to the next position. As a result, all elements on the left side of lt are smaller than pivot, all elements on the right side of gt are larger than pivot, and all elements in the middle are equal to pivot. This **loop invariant** maintains util termination, when the whole array is already partitioned into three parts. And the current pointer only scans once through the array so the runtime is $\Theta(n)$. Finally we exchange the pivot with the element at pointer gt, and return the pointers lt and gt.

```
void QUICKSORT'(A, p, r)
{
    if (p < r)
    {
        auto [q, t] = PARTITION'(A, p, r);
        QUICKSORT' (A, p, q - 1);
        QUICKSORT' (A, t + 1, r);
    }
}

```

If all n elements have the same value, the modified algorithm QUICKSORT' runs $\Theta(n)$ because the partitioning process only needs to be done once and return the pointers lt (== p) and gt (== r), and the following recursion calls do nothing because the pointers arguments are at same position.