Solutions for Exercise Sheet 5

Handout: Oct 17th — Deadline: Oct 24th, 4pm

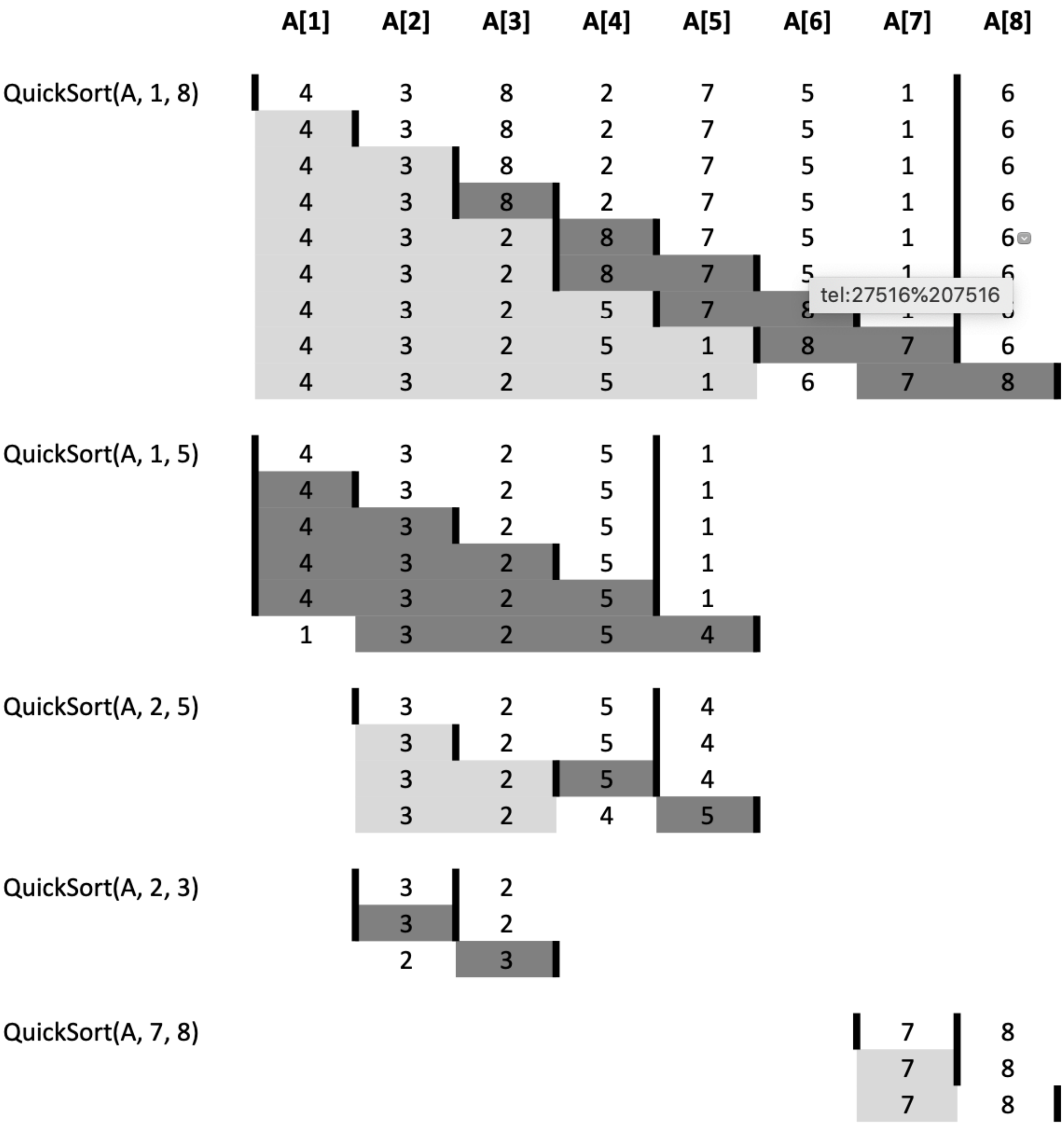
Question 5.1 (Marks: 0.25)

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.

Solution:



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).

Solution: We prove it by induction on the length of the array A.

Base case: n=1

We have $p <= r$ and the algorithm returns the element untouched.

Inductive case: We assume the algorithm works for lengths up to $n - 1$ and prove that it works for length n.

PARTITION returns the array $[p, .., q - 1, q, q + 1, ..r]$ where the elements before q are smaller than q and the elements after q are larger.

Then QUICKSORT$[p, ..q - 1]$ and QUICKSORT$[q, ..r]$ return the respective subarrays sorted by inductive hypothesis so the algorithm is correct as q is in the right place already.

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)

Solution: Partition will always return either the largest or the smallest element. So we get the the same recurrence equation as when the array is increasingly ordered: $T(n) = T(n-1) + \Theta(n)$ leading to a $\Theta(n^2)$ runtime.

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).

Solution: PARTITION will include all equal elements in the left-hand part of the array, increasing i in every iteration of the loop. The loop will terminate with $i + 1 = r$, hence swapping the pivot with itself and returning $q = r$.

The runtime of QUICKSORT is $\Theta(n^2)$ as the size of the larger subarray is only reduced by 1 in each recursive call. An input of n equal values is a worst-case input for QUICKSORT!

Question 5.5 (Marks: 0.5)

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

- $A[p \dots i]$ contains elements smaller than x
- $A[i + 1 \dots k]$ contains elements equal to x and
- $A[k + 1 \dots j - 1]$ 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 $A[q \ldots t]$ 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?

Solution: The idea behind the pseudocode given below is as follows. There are three cases for the new element $A[j]$. If it is larger than x, nothing needs to be done as $A[j]$ is in the right place. If it is equal to x, we put it in the middle part by increasing k and swapping it with $A[k]$. If it is smaller than x, we need to shift the right and the middle parts by 1. This can be achieved by first swapping $A[j]$ with $A[k]$, the last element of the middle part, and then swapping it again with the last element of the left part, $A[i]$ (after increasing i and k).

At the end, the pivot is swapped with $A[k+1]$, the first element amongst those larger than the pivot.

There is only a constant number of swaps and other operations in each execution of the loop, so the runtime for an n-element subarray is still $\Theta(n)$.

```
Partition'(A, p, r)
x = A[r]
i = p − 1
k = p − 1
for j = p to r − 1 do
    if A[j] = x then
        k = k + 1
        exchange A[k] with A[j]
    if A[j] < x then
        i = i + 1
        k = k + 1
        exchange A[k] with A[j]
        exchange A[k] with A[i]
exchange A[k + 1] with A[r]
return i + 1, k + 1
```

The modified QuickSort algorithm then looks as follows:

```
QuickSort'(A, p, r)
if p < r then
    q, t = Partition'(A, p, r)
    QuickSort'(A, p, q − 1)
    QuickSort'(A, t + 1, r)
```

The runtime of QuickSort' on an input of n equal elements is $\Theta(n)$ (essentially the time for Partition'$(A, 1, n)$) as QuickSort'$(A, 1, n)$ leads to recursive calls on two empty subarrays.

Question 5.6 (Marks:0.5)

Implement QuickSort, Randomized-Quicksort and QuickSort$'$ from Question 5.4