DM582 Exercises - Sheet 6

Mads Anker Nielsen Tobias Samsøe Sørensen

March 5, 2025

This document contains exercises from the course DM582 (spring 2025). Most exercises are from the book *Introduction to Algorithms, 4th edition* by Cormen, Leiserson, Rivest, and Stein (CLRS), the book *Algorithm Design, 1st edition* by J. Kleinberg and E. Tardos (KT), and the book *Discrete Mathematics and its Applications, 8th edition* by K. Rosen.

References to CLRS refer to the book *Introduction to Algorithms, 4th edition* by Cormen, Leiserson, Rivest, and Stein.

References to KT refer to the book *Algorithm Design, 1st edition* by J. Kleinberg and E. Tardos.

References to Rosen refer to the book *Discrete Mathematics and its Applications, 8th edition* by K. Rosen.

This document will inevitably contain mistakes. If you find some, please report them to your TA so that we can correct them.

Sheet 6

CLRS, 7.4-5

Exercise

Coarsening the recursion, as we did in Problem 2-1 for merge sort, is a common way to improve the running time of quicksort in practice. We modify the base case of the recursion so that if the array has fewer than k elements, the subarray is sorted by insertion sort, rather than by continued recursive calls to quicksort. Argue that the randomized version of this sorting algorithm runs in $O(nk + n\log(n/k))$ expected time. How should you pick k, both in theory and in practice?

CLRS, 7-1 (a-b)

Exercise

The version of **PARTITION** given in this chapter is not the original partitioning algorithm. Here is the original partitioning algorithm, which is due to C. A. R. Hoare.

```
HOARE-PARTITION(A, p, r)
 1  x = A[p]
 2  i = p − 1
 3  j = r + 1
 4  while TRUE
 5      repeat
 6          j = j − 1
 7      until A[j] ≤ x
 8      repeat
 9          i = i + 1
10      until A[i] ≥ x
11      if i < j
12          exchange A[i] with A[j]
13      else return j
```

a. Demonstrate the operation of **HOARE-PARTITION** on the array $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21\rangle$, showing the values of the array and the indices i and j after each iteration of the while loop.

b. Describe how the **PARTITION** procedure in Section 7.1 differs from **HOARE-PARTITION** when all elements in $A[p : r]$ are equal. Describe a practical advantage of **HOARE-PARTITION** over **PARTITION** for use in quicksort.

CLRS, Problem 7-2

Exercise

The analysis of the expected running time of randomized quicksort in Section 7.4.2 assumes that all element values are distinct. This problem examines what happens when they are not.

a. Suppose that all element values are equal. What is randomized quicksort's running time in this case?

b. The `PARTITION` procedure returns an index q such that each element of $A[p : q-1]$ is less than or equal to $A[q]$ and each element of $A[q+1 : r]$ is greater than $A[q]$. Modify the `PARTITION` procedure to produce a procedure `PARTITION'` (A, p, r), which permutes the elements of $A[p : r]$ and returns two indices q and t, where $p \le q \le t \le r$, such that

   - all elements of $A[q : t]$ are equal,
   - each element of $A[p : q-1]$ is less than $A[q]$, and
   - each element of $A[t+1 : r]$ is greater than $A[q]$.

   Like `PARTITION`, your `PARTITION'` procedure should take $O(r-p)$ time.

c. Modify the `RANDOMIZED-PARTITION` procedure to call `PARTITION'`, and name the new procedure `RANDOMIZED-PARTITION'`. Then modify the `QUICKSORT` procedure to produce a procedure `QUICKSORT'`(A, p, r) that calls `RANDOMIZED-PARTITION'` and recurses only on partitions where elements are not known to be equal to each other.

d. Using `QUICKSORT'`, adjust the analysis in Section 7.4.2 to avoid the assumption that all elements are distinct.

CLRS, Problem 7-4

Exercise

Professors Howard, Fine, and Howard have proposed a deceptively simple sorting algorithm, named `stooge-sort` in their honor, appearing on the following page.

```
STOOGE-SORT(A, p, r)
1  if A[p] > A[r]
2      exchange A[p] with A[r]
3  if p + 1 < r
4      k = ⌊(r − p + 1)/3⌋          // round down
5      STOOGE-SORT(A, p, r − k)     // first two-thirds
6      STOOGE-SORT(A, p + k, r)     // last two-thirds
7      STOOGE-SORT(A, p, r − k)     // first two-thirds again
```

a. Argue that the call `STOOGE-SORT(A, 1, n)` correctly sorts the array $A[1 : n]$.

b. Give a recurrence for the worst-case running time of `STOOGE-SORT` and a tight asymptotic (Θ-notation) bound on the worst-case running time.

c. Compare the worst-case running time of `STOOGE-SORT` with that of insertion sort, merge sort, heapsort, and quicksort. Do the professors deserve tenure?

CLRS, 9.2-1

Exercise

Show that `RANDOMIZED-SELECT` never makes a recursive call to a 0-length array.

CLRS, Problem 9-1

Exercise

You are given a set of n numbers, and you wish to find the i largest in sorted order using a comparison-based algorithm. Describe the algorithm that implements each of the following methods with the best asymptotic worst-case running time, and analyze the running times of the algorithms in terms of n and i.

1. Sort the numbers, and list the i largest.
2. Build a max-priority queue from the numbers, and call `EXTRACT-MAX` i times.
3. Use an order-statistic algorithm to find the i-th largest number, partition around that number, and sort the i largest numbers.

Exercise from course webpage

Exercise

Assume that for an oral exam, there are k questions that students can draw from. ~~Kim~~ The hypothetical lecturer finds it tiring to hear the same topic several times, so when a student draws a question, he doesn't put it back again for the next student. Thus, when the first student in the exam sequence draws, there are k questions laid out on the table, when the second student draws, there are only $k-1$ questions on the table, etc. The ith student sees a table with $k-i+1$ questions. At some point, the lecturer restarts the process with all k questions.

The students employ different algorithms for selecting; some try to choose uniformly at random, some take the nearest, some the one furthest away, etc. The lecturer, on the other hand, places the questions in an order chosen uniformly at random, and either does not change the order in between questions disappearing, or sometimes collects all the questions currently on the table and places them uniformly at random again.

Are the students treated fairly in the sense that the ith student gets any of the questions with probability $1/k$, i.e., independent of the person's placement in the exam sequence?