Assignment 2 – Selected Model Answers

Exercise 1.

(i) Generalise the mergesort algorithm splitting a list into k sublists instead of just 2. Show that the complexity remains the same.

(ii) Modify the mergesort algorithm adding a threshold value t such that for lists with a length at most t an elementary sorting algorithm (bubblesort, insertion sort, selection sort) is used instead of mergesort. Implement the modified algorithm (choose one of the three elementary sorting algorithms).

(iii) Provide a theoretical and experimental analysis to determine a good value for the threshold t, which optimises performance.

Exercise 2. Suppose you have to process n advance bookings of rooms for a hotel with k identical rooms. Bookings contain an arrival date and a departure date. You have to find out whether there are enough rooms in the hotel to satisfy the demands.

(i) Design an algorithm that solves this problem in time $O(n \log n)$.

Hint. Sort the set of all arrivals and departures and process it in sorted order. Choose the most appropriate sorting algorithm for this problem.

(ii) Implement your algorithm.

Solution. We only consider part (i). Let a booking be a pair (a, d) with an arrival date a and a departure date d. We can map dates to $\mathbb{N}$, so without loss of generality assume $a, d \in \mathbb{N}$ with $a < d$. We can then view the i'th booking as a pair comprising $(a_i, 1)$ (for arrival) and $(d_i, 0)$ for departure. The set of all advance bookings is given by the list $[(a_1, 1), (d_1, 0), (a_2, 1), (d_2, 0), \dots, (a_n, 1), (d_n, 0)]$.

We can sort this list using the order $(x_1, y_1) \leq (x_2, y_2)$ iff $x_1 < x_2$ or ($x_1 = x_2$ and $y_1 \leq y_2$). As the components are natural numbers, the most appropriate sorting algorithm is radix sort.

If we now have a sorted list, we can iterate over it using a counter H, which is initialised as 0. A list element $(x, 1)$ increments H, while a list element $(x, 0)$ decrements H. After each change of H we have to check, if $H \leq k$ holds. If true, we can continue. If false, the capacity of the hotel has been exceeded.

Exercise 3. Explain how to implement a FIFO queue using two stacks so that each FIFO operation takes amortised constant time.

Solution. Let the two stacks be $s1$ and $s2$. For the *pushback* operation—accordingly also for *back*—we use only $s1$. If we have a *popfront* operation, we have to distinguish two cases:

Case 1. If s_2 is empty, we first move all elements from $s1$ to $s2$, where a move consists of a *pop* operation on $s1$ followed by a *push* operation on $s2$. Then proceed as in Case 2.

Case 2. If $s2$ is not empty, then the *popfront* operation becomes simply a *pop* operation on $s2$.

The operation *topfront* is handled analogously without changing $s2$ is the second case. The emptiness check amounts to checking, if both stacks are empty.

For the amortised complexity let each *pushback* operation make a constant contribution $2c$ to a counter C, where c is the cost required by any *push* or *pop* operation. Thus, moving all elements from $s1$ to $s2$ in Case 1 above requires the cost of $2nc$, where n is the number of elements moved.

When a move becomes necessary, there have been n *pushback* operations since the latest move, which altogether contributed $2nc$ to the counter C. By executing the move the counter is reset to 0. The constant contribution of the *pushback* operation guarantees that its amortised complexity remains in $\Theta(1)$. No other operation is affected.

EXERCISE 4. It is easy to check to check whether an algorithm produces a sorted output. It is less easy to check whether the output is also a permutation of the input. However, for integers there exists a fast and simple algorithm:

(i) Show that $[e_1, \dots, e_{n_i}]$ is a permutation of $[e'_1, \dots, e'_{n_i}]$ iff the polynomial

$$P(x) \;=\; \prod_{i=1}^{n}(x - e_i) - \prod_{i=1}^{n}(x - e'_i)$$

in the variable x is identically zero.

(ii) For any $\varepsilon > 0$ let p be a prime with $p > \max\{n/\varepsilon, e_1, \dots, e_{n_i}, e'_1, \dots, e'_{n_i}\}$. The idea is to evaluate the above polynomial $P(x)$ modulo p for a random value $x \in [0, p-1]$.

Show that if $[e_1, \dots, e_{n_i}]$ is not a permutation of $[e'_1, \dots, e'_{n_i}]$, then the result of the evaluation is zero with probability at most ε.

Hint. A non-zero polynomial of degree n has at most n zeroes.

SOLUTION.

(i) Clearly, if $[e_1, \dots, e_n]$ is a permutation of $[e'_1, \dots, e'_n]$, the factors in both products are the same, so the polynomial is identical zero.

Conversely, both products are polynomials of degree n with roots $\{e_1, \dots, e_n\}$ and $\{e'_1, \dots, e'_n\}$. If $P(x)$ is identically zero, these two polynomials are the same, and hence have the same roots with the samne multiplicities, which in other words means that $[e_1, \dots, e_n]$ is a permutation of $[e'_1, \dots, e'_n]$.

(ii) $P(x)$ is a polynomial $x^n + a_{n-1}x^{n-1} + \dots + a_0$ of degree n, and as such has at most n roots, unless it is identically zero. As p is a prime and all e_i, e'_i are in $[0, p-1]$, the same holds modulo p.

Taking random values over the interval $[0, p-1]$ means we have p possible values for x, and the outcome $P(x) = 0$ can appear at most n times. So the probability that $P(x) = 0$, i.e. the probability that x is one of the at most n roots, is at most $\frac{n}{p} < \varepsilon$.