RWTH Aachen **WS 2025**
Theoretical Computer Science Exercise 1
Rossmanith–Gehnen 30.10.2025

Exercise for Analysis of Algorithms

Exercise 1

Consider the following algorithm that computes the maximum element in an array of positive integers. We assume that all elements are pairwise different and that each permutation occurs with equal probability.

```
int maxElem(int *a, int N)
{
  int i,max;

  max = -1;              /*  1  */
  for (i=0; i<N; i++)    /*  2  */
    if (a[i] > max)      /*  3  */
      max = a[i];        /*  4  */
  return max;            /*  5  */
}
```

- How often are the lines 3 and 4 executed in the *worst case* (in the *best case*)?
- What is the probability that this worst case occurs?
- How often are the lines 3 and 4 executed in the *average case*?

Solution:

- Line 3 is executed N times. Since `max` is negative at the beginning, it must be overwritten in line 4 at least once even in the best case, when the largest element is at the beginning. In the worst case, the array is sorted and increasing, the condition in line 3 is always true and line 4 is executed N times.
- The best case occurs if the first element is the maximum element. The probability for this event is $1/N$. The worst case happens with a probability of only $1/N!$, since among the $N!$ permutations of the array only one is the increasing sequence.
- Line 3 is executed N times. The estimation for line 4 is more complicated, since the probability that line 4 is executed for $a[i]$ depends on the previous $a[j]$, $j < i$, and is thus not independent for all i. We can tackle this problem with the linearity of the expected value.
  The probability that the kth element is larger than its $k-1$ predecessors is $1/k$, since among the $k!$ permutations of the first k elements there are exactly $(k-1)!$, s.t. the largest element is at position k. The expected number of executions of line 4 is therefore
  $$\sum_{i=0}^{N-1} \frac{1}{i+1} = \sum_{k=1}^{N} \frac{1}{k} = H_N = \ln N + \gamma + o(1),$$
  where H_N is the Nth harmonic number and $\gamma \approx 0.78$.

Exercise 2
Let S_N be the expected number of pushs on the stack in the Quicksort program, where the input consists of a random permutation of N distinct keys. Analyze S_N for all values of N and M.

Solution:
We start by giving a recurrence relation for S_N, note that we only push something on the stack if *both* halves are larger than M, that is, $M+1 \le k \le N-1-k$.

$$\begin{aligned} S_N &= \frac{1}{N}\sum_{k=0}^{N-1}\big(S_k + S_{N-1-k} + (M+1 \le k \le N-1-k)\big) \\ &= \frac{2}{N}\sum_{k=0}^{N-1} S_k + \frac{N-2M-2}{N} \end{aligned}$$

This equation is only valid for $N \ge 2M+2$ otherwise $S_N = 0$.
From the lecture we know that for $N > M$,

$$X_N = \frac{2}{N}\sum_{k=0}^{N-1} X_k + f_N = \frac{N+1}{M+2}X_{M+1} + (N+1)\sum_{k=M+2}^{N}\frac{kf_k - (k-1)f_{k-1}}{k(k+1)}$$

We need to define a function f_N that is valid for smaller values of N and get

$$f_N = \begin{cases} \frac{N-2M-2}{N} & N \ge 2M+2 \\ 0 & \text{else} \end{cases}$$

Now we can solve S_N. Since $S_{M+1} = 0$ and $f_k = 0$ for $k \le 2M+2$ the expression simplifies a lot.

$$\begin{aligned} S_N &= (N+1)\sum_{k=2M+3}^{N}\frac{1}{k(k+1)} \\ &= (N+1)\Big(\frac{1}{2M+3} - \frac{1}{N+1}\Big) \\ &= \frac{N+1}{2M+3} - 1 \end{aligned}$$

Alternative Solution: It is also possible to solve S_N directly without using the formula for X_N

$$\begin{aligned} NS_N - (N-1)S_{N-1} &= 2S_{N-1} + 1 \\ NS_N &= (N+1)S_{N-1} + 1 \\ \sigma_N &:= \frac{S_N}{N+1} = \sigma_{N-1} + \frac{1}{N(N+1)} \\ \sigma_N &= \sum_{k=2M+4}^{N}\frac{1}{k(k+1)} + \sigma_{2M+3} = \frac{1}{2M+4} - \frac{1}{N+1} + \sigma_{2M+3} \\ S_N &= \frac{N+1}{2M+4} - 1 + \frac{(N+1)}{(2M+3)(2M+4)} = \frac{N+1}{2M+3} - 1 \end{aligned}$$

Exercise 3

As we discussed at the first discussion session, we would assume that C' —the number of comparisons done in quicksort during partitioning phases *on the left side* of the array — should be around half the number of comparions, i.e., around $C/2$.

This is as all comparisons are distributed among the following two lines in the program. We split the number of comparisons into C' for the first line and C'' for the second line.

```
do {i++;} while(a[i]<k);
do {j--;} while(k<a[j]);
```

Can we motivate the recurrence relation for C'_N, which shows this?

Solution:

We get the recurrence

$$C'_N = \frac{1}{N}\sum_{k=0}^{N-1}(C_k + C_{N-1} + k + 1) = \frac{2}{N}\sum_{k=0}^{N-1} C_k + \frac{N+1}{2}$$

because $k+1$ comparisons take place of left side of the pivot element under the conditions that the pivot element will be left at position $k+1$ in the array (when the positions are $1, \ldots, n$). This recurrence is exactly half of the recurrence we had for C_N. Because of linearity the result is then $C'_N = C_N/2$.

Exercise 4

Two natural numbers $m \neq n$ are called *friendly*, if the sum of all factors of m equals n — or the other way round. A son and a father wrote the following programs, that compute friendly numbers. What is their running time?

Son }

```
#include <iostream>
int e[150000];
int realdiv(int a) {
int n=0;
for(int i=1; i+i<=a; i++)
if(a%i==0) n+=i;
e[a] = n;
return n;
}
main() {
for(int i=0; i<150000; i++) {
int a = realdiv(i);
if(a >= i) continue;
if(e[a]==i) {
std::cout << i << " "
<< realdiv(i) << "\n";
}
}
```

Father

```
#include <stdio.h>
#define N 1000000
int factorsum[N];
int main() {
int i;
for(i=1; i<N; i++) {
int p=i;
while(p<N) {
factorsum[p] += i;
p += i;
}
}
for(i=1; i<N; i++) {
int a = factorsum[i]-i;
if(a<i && i==factorsum[a]-a)
printf("%d %d\n", a, i);
}
return 0;
}
```

Solution:

The son's program requires $O(N^2)$ steps. In the father's program, the first for-loop dominates the running time. In the following table, lines represent the outer for-loop, columns represent the inner while-loop, and each 1 represents an execution of the inner loop (for the ease of presentation we write N instead of $N-1$):

$$\overbrace{\begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\ & 1 & & 1 & & 1 & & 1 & & \cdots & 1 \\ & & 1 & & & 1 & & & 1 & \cdots & \\ & & & 1 & & & & 1 & & \cdots & \\ & & & & 1 & & & & & \cdots & 1 \\ \vdots & & & & & \ddots & & & & \cdots & \vdots \\ & & & & & & & & & & 1 \end{array}}^{N}$$

We can rewrite this as follows

$$\overbrace{\begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\ 1/2 & 1/2 & 1/2 & 1/2 & 1/2 & 1/2 & 1/2 & 1/2 & 1/2 & \cdots & 1/2 \\ 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & \cdots & 1/3 \\ 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & \cdots & 1/4 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & \cdots & 1/5 \\ \vdots & & & & & & \ddots & & & \cdots & \vdots \\ 1/N & 1/N & 1/N & 1/N & 1/N & 1/N & 1/N & 1/N & 1/N & \cdots & 1/N \end{array}}^{N}$$

It is not hard to see that the sum over all entries of both tables differ by at most N (due to missing values in the last colums). Since the sum of each column is H_N, it is easy to see that the sum over the lower table is $NH_N = O(N \log N)$.