RWTH Aachen **SS 2012**
Lehrgebiet Theoretische Informatik Problem Set 2
Langer—Reidl—Rossmanith—Sikdar 23.4.2012

Analysis of Algorithms — Tutorial

Problem 2-1

In the analysis of the quicksort algorithm, the term S_N represents the expected number of pushes to the stack. Find out an expression for S_N.

Solution:

If the array has less than $2M+3$ entries, then the partitioning phase will produces at least one subarray of size less than $M+1$ and nothing will be put on the stack. An array of size exactly $2M+3$ might be paritioned into two subarrays of size $M+1$, in which case one of them will be pushed onto the stack. The probabiliy of this even is exactly $\frac{1}{2M+3}$ as there is exactly one element in the array which, if chosen as the pivot, produces this partition.

In general an interval will be pushed on the stack if the parition produces to intervals of size at least $M+1$. Assuming an array of size larger than $2M+3$, there are $N-2M-2$ numbers k which fall into the interval $M+1 < k < N-M$, which in turn will partition the array into large enough subarrays. Choosing such an element and thus producing a push to the stack then happens with probability $\frac{N-2M-2}{N}$. We therefore obtain

$$S_N = \frac{2}{N}\sum_{k=0}^{N-1} S_k + \frac{N-2M-2}{N}$$

for $N > 2M+3$. By considering the difference of two consecutive elements of the sequence S_N, we can get rid of the sum an obtain the recurrence

$$\begin{aligned} NS_N - (N-1)S_{N-1} &= 2S_{N-1} + 1 \\ NS_N &= (N+1)S_{N-1} + 1 \end{aligned}$$

which we can solve by substitution:

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

As this formula resolves to $\frac{1}{2M+3}$ for $N = 2M+3$, we finally obtain

$$S_N = \begin{cases} 0 & \text{if } N < 2M+3 \\ \frac{N+1}{2M+3} - 1 & \text{else} \end{cases}$$

Problem 2-2

Consider the following algorithm. The array $a[\,]$ contains a random permutation of the numbers $1, \ldots, N$.

```
void doSomething(int a[ ], int N) {
  int i;

  for (i = 0; i < N − 1; i++)
    while (a[i] < a[i + 1])
      a[i]++;
}
```

What is the expected number of executions of the instruction $a[i]{+}{+}$?

Solution

We first calculate the expected value of the random variable

$$Z_k = \max\{0, a[k] - a[k+1]\},$$

which describes how often line 3 is executed in the above program if the variable i takes value k. In total then line 3 is exectued $Z_1 + Z_2 + \cdots + Z_{N-1}$ times.

Consider the probablity that a certain pair $1 \le x, y \le N, x \ne y$ appears consecutively in the array at position k and $k+1$. As there are $(N-2)!$ possible permutations for the remaining elements we obtain

$$\Pr[x = a[k], y = a[k+1]] = \frac{(N-2)!}{N!} = \frac{1}{N(N-1)}$$

For such a pair line 3 is executed $\max\{0, x - y\}$ times. Therefore the expected value of Z_k is given by

$$E(Z_k) = \sum_{1 \le y < x \le N} \frac{x-y}{N(N-1)} = \sum_{x=1}^{N} \sum_{y=x+1}^{N} \frac{x-y}{N(N-1)} = \frac{N+1}{6}.$$

Using the linearity of expected values the total sum can be calculated easily:

$$E(Z_1 + Z_2 + \cdots + Z_{N-1}) = \sum_{k=1}^{N-1} E(Z_k) = \frac{(N-1)(N+1)}{6} = \frac{N^2-1}{6}$$

Thus, the average number line 3 is executed is exactly $(N^2-1)/6$.

Homework Assignment 2-1 (10 Points)

We already analyzed C_n, the *total* expected number of comparisons in the two innermost **while**-loops of the quicksort algorithm (see the program fragment below).

What is the expected number of executions of the single comparison $a[i] < k$?

```
[...]
    i = l − 1; j = r; k = a[j];
    do{
      do{i++;}while(a[i] < k);
      do{j−−;}while(k < a[j]);
      t = a[i]; a[i] = a[j]; a[j] = t;
    } while(i < j);
[...]
```

Homework Assignment 2-2 (10 Points)

Last week we analysed the running of a program that computes *amicable numbers*. Recall that two natural numbers $m \neq n$ are called *amicable*, if the sum of all proper factors of m equals n — and the other way around.

This week we will turn to the father's program. Determine the expected number of executions of the instruction $p += i$ as a function of N of the form $f(N) + O(N)$.

Which of the two programs is faster?

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
      << "  " << a << "\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: 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 $N-1$):

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

We can rewrite this as follows

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

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 $NH_N = N \log N + O(N)$