ALGORITMIEK: opgaven werkcollege 4

Opgave 1. (Levitin: opgave 2.1.2.a) Consider the following algorithm for finding the difference between two $n \times n$ matrices A and B:

```
for (i=0;i<n;i++)
  for (j=0;j<n;j++)
    diff[i][j] = A[i][j] - B[i][j];
```

What is its basic operation? How many times is it performed as a function of the matrix order n? As a function of the total number of elements in the input matrices?

Opgave 2. (Levitin: 2.2.5)
List the following functions according to their order of growth from the lowest to the highest:

$$(n^2+3)!, \quad 2\log_2((n+50)^5), \quad 3^{3n}, \quad 0.05n^{10}+3n^3+1 \quad (\ln n)^3, \quad \sqrt{n}, \quad 3^{2n}$$

Here, $\ln n$ is the natural logarithm of n, i.e., $\log_e n$.

Opgave 3. (Levitin: 2.3.5)
Consider the following algorithm:

```
ALGORITHM Foo (A[0..n-1])
  // Input: An array A[0..n-1] of n real numbers
  val = 100;
  sumgreater = 0;
  sumless = 0;
  for (i=0;i<n;i++)
  { if (A[i]>val)
      sumgreater = A[i];
    if (A[i]<val)
      sumless = A[i];
  }
  return (sumgreater - sumless);
```

a. What does this algorithm compute?
b. What is its basic operation?
c. How many times is the basic operation executed?
d. What is the efficiency class of this algorithm?
e. Suggest an improvement or a better algorithm altogether, and indicate its efficiency class. If you cannot do it, try to prove that, in fact, it cannot be done.

Opgave 4.
Consider the following algorithm:

```
ALGORITHM doSomething (A[0..n-1])
  // Input: An array A[0..n-1] of n real numbers
  i = 0;
  while (i<n-1)
  { while (i<n-1 && A[i]<=A[i+1])
      i++;

    if (i<n-1)
    { swap (A[i],A[i+1])
      if (i>0)
        i--;
    }
  }
```

a. What does this algorithm compute?
b. What would be a proper basic operation? Explain your answer.
c. How many times is the basic operation executed in the best case? And in the worst case?
d. What is the worst case time complexity of this algorithm?

Opgave 5. (cf. Levitin: 2.5.7)
The Fibonacci numbers $F(n)$ are defined by the following recurrence relation:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n \geq 2 \end{cases}$$

Consider the following recursive function to compute the n-th Fibonacci number.

```
int Fib (int n)
{
  if (n<=1)
    return n;
  else
    return Fib(n-1) + Fib(n-2);
}
```

a. A proper basic operation for this function is the addition performed in the second return statement. Let $A(n)$ be the number of times this basic operation is performed in computing `Fib`(n). Find a recurrence relation for $A(n)$, and express $A(n)$ in terms of the Fibonacci numbers themselves.
b. Let $C(n)$ and $Z(n)$ be the number of times `Fib`(1) and `Fib`(0) are computed respectively in computing `Fib`(n). Find recurrence relations for $C(n)$ and for $Z(n)$, and express both functions in terms of the Fibonacci numbers themselves.

Opgave 6. (Levitin: 2.4.11)
The determinant of an $n \times n$ matrix

$$A = \begin{pmatrix} a_{0,0} & \cdots & a_{0,n-1} \\ a_{1,0} & \cdots & a_{1,n-1} \\ \vdots & & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,n-1} \end{pmatrix}$$

denoted $\det A$, can be defined as $a_{0,0}$ for $n = 1$ and, for $n > 1$, by the recursive formula

$$\det A = \sum_{j=0}^{n-1} s_j \cdot a_{0,j} \cdot \det A_j,$$

where s_j is $+1$ if j is even and -1 if j is odd, $a_{0,j}$ is the element in row 0 and column j, and A_j is the $(n-1) \times (n-1)$ matrix obtained from matrix A by deleting its row 0 and column j.
a. Set up a recurrence relation for the number of multiplications made by the algorithm implementing this recursive definition.
b. Without solving the recurrence, what can you say about the solution's order of growth as compared to $n!$?