Exercise Sheet 1 Solutions

Fabian Ibanez Avila
7047016

October 18, 2024

Exercise 1: Missing Number

Exercise 1 (Missing Number) — **5+5** points

Let $A[1, \ldots, n]$ be an array such that every number in $\{0, \ldots, n\}$ appears exactly once in A, except for one missing number x. In the MissingNumber problem, we are given such an array A and the goal is to compute the missing number x. For example: given the array $A = [4, 2, 1, 0, 5]$ (of length $n = 5$), the the missing number is $x = 3$.

Prove correctness of each of the following algorithms, that is, argue that these algorithms correctly solve the MissingNumber problem.

a.

```
1: procedure Alg1(A[1...n])
2:     Sort(A)                    ▷ sorts the array such that A[1] ≤ A[2] ≤ ··· ≤ A[n]
3:     for i = 1, ..., n do
4:         if A[i] == i then
5:             return i − 1
6:     return n
```

b.

```
1: procedure Alg2(A[1...n])
2:     y := 0
3:     for i = 1, ..., n do
4:         y := y + i − A[i]
5:     return y
```

Algorithm a

Correctness Proof:
The algorithm a is correct. The algorithm first sorts the array into a increasing order. Afterwards, the process of finding the missing number is divided into two different cases.

- Case 1: The missing number is NOT the last number.

In the sorted algorithm, a number is skipped. As a result, once the for loop arrives to the i index where the number is skipped, the number that should be there is actually the next one. This is exactly what the condition A[i] == i checks. If the condition is true, then the index minus one is returned (since the index starts at 1 and the sequence at 0).

- Case 2: The missing number is the last number.

  The size of the array is n while the size of all possible numbers that could be missing is n+1. Therefore, the array cannot contain all the possible numbers. This means that if the sorted array is a sequence without skipped numbers, the condition inside the for loop will never be triggered. Thus, finally, it should return n (since the index starts at 1 and the sequence at 0).

Algorithm b

Correctness Proof:
The algorithm b is correct. To show its correctness, based on the code provided, I will write y as an explicit formula based on the length of the array n and the missing number x.

$$y = \sum_{i=1}^{n} i - (\sum_{i=0}^{n} i - x)$$

The first sum of the formula represents the sum of the indices done in the for loop (starting from 0 and ending in n).
Then, the expression inside the parenthesis represents the elements of the array. The sum represents the sum of all the possible elements that the array can contain(from 0 to n), while x is the missing number from the array. By subtracting x from this sum of all possible numbers, we get the sum of all the elements inside the array A.
Now, let us simplify it, so it can become obvious why the code is correct:

$$y = \sum_{i=1}^{n} i - (\sum_{i=0}^{n} i - x) \Leftrightarrow y = \sum_{i=1}^{n} i - (0 + \sum_{i=1}^{n} i - x)$$

$$y = \sum_{i=1}^{n} i - \sum_{i=1}^{n} i + x$$

$$y = x$$

Thus, y is indeed the missing number (that is returned).

Exercise 2: Polynomial Evaluation

Exercise 2 (Polynomial Evaluation) — **10** points

We are given integer coefficients $a_0, \ldots, a_n$ of a polynomial $p(X) = \sum_{i=0}^{n} a_i \cdot X^i$, and an evaluation point z. Prove that the following pseudocode correctly evaluates the polynomial at z, that is, argue that it correctly computes the value $p(z)$. (You can assume that all intermediate values computed in the algorithm fit into a memory cell.)

```
1: y := 0
2: for i = n, n − 1, ..., 0 do
3:     y := a_i + z · y
4: return y
```

Solution

To show the correctness of the program, I will transform y from its recursive form to an explicit form:

$$y = a_0 + (a_1 + (a_2 + ... + (a_{n-1} + a_n * z) * ...) * z) * z$$

After this transformation, we notice that the form of y is actually the same as the Horner's method for polynomials (view in MFI II). Thus, we obtain pseudocode that correctly computes the values of $p(z)$.

Exercise 3: Polynomial Evaluation

Exercise 3 (Asymptotic Notation) — **2+2+2+4 points**

Prove the following laws that hold for any functions $f_1, f_2, g_1, g_2, f, g, h \geq_{ae} 0$:

a. **Additivity:** If $f_1 = O(g_1)$ and $f_2 = O(g_2)$ then $f_1 + f_2 = O(\max(g_1, g_2))$.

b. **Multiplicativity:** If $f_1 = O(g_1)$ and $f_2 = O(g_2)$ then $f_1 \cdot f_2 = O(g_1 \cdot g_2)$.

c. **Transitivity:** If $f = O(g)$ and $g = O(h)$ then $f = O(h)$.

d. $f^c = o(b^f)$ for any function $f = \omega(1)$ and constants $c \geq 0, b > 1$.

Parts a-c also holds after replacing O by o, Ω or ω, but you don't need to prove this.

Hint: It might help to refresh your knowledge of logarithm laws, see e.g. sections 1-4 of `https://en.wikipedia.org/wiki/List_of_logarithmic_identities`

Solution

a)

We know that:

$$f_1 \in O(g_1) \Leftrightarrow O(g_1) := \{f_1 \geq_{ae} 0 | \exists c_1 > 0 : f_1 \leq_{ae} c_1 * g_1\}$$
$$f_2 \in O(g_2) \Leftrightarrow O(g_2) := \{f_2 \geq_{ae} 0 | \exists c_2 > 0 : f_1 \leq_{ae} c_2 * g_2\}$$

Then, assume $f_1 = O(g_1)$ and $f_2 = O(g_2)$, then:

$$f_1 + f_2 \leq_{ae} c_1 * g_1 + c_2 * g_2 \leq_{ae} (c_1 + c_2) * (g_1 + g_2) \leq_{ae} (c_1 + c_2) * \max(g_1, g_2$$

Now, if we set the constant c to $c_1 + c_2$, we obtain the form of the claim. Thus the claim is correct.

b)

Following the same logic as before, assume $f_1 = O(g_1)$ and $f_2 = O(g_2)$, then:

$$f_1 * f_2 \leq_{ae} c_1 * g_1 * c_2 * g_2 \Leftrightarrow (c_1 * c_2) * (g_1 * g_2)$$

By setting $c := c_1 * c_2$, we obtain the same form as in the claim. Thus, property is indeed correct.

c)

We know that:

$$f \in O(g) \Leftrightarrow O(g) := \{f \geq_{ae} 0 | \exists c_g > 0 : f \leq_{ae} c_g * g\}$$
$$g \in O(h) \Leftrightarrow O(h) := \{g \geq_{ae} 0 | \exists c_h > 0 : h \leq_{ae} c_h * h\}$$

Now, assume $f \in O(g)$ and $g \in O(h)$, then:

$$f \leq_{ae} c_g * g \leq c_h * h \Leftrightarrow f \leq_{ae} c_h * h$$

Thus, the claim is proved.

c)

Exercise 4: Asymptotic Growth

Exercise 4 (Asymptotic Growth) — **10** points

Order the following functions according to their asymptotic growth. Prove your answer.

$2n^2 - 4n, \quad n^2 \log n, \quad n \log n, \quad 2^n, \quad \log n, \quad n \log \log n.$

Hint: Use the laws from exercise 3.

Solution

Solution:

$log(n) \leq_{ae} n * log(log(n)) \leq_{ae} n * log(n) \leq_{ae} 2n^2 - 4n \leq_{ae} n^2 * log(n) \leq_{ae} 2^n$

First, from multiplicativity, we notice that $log(n) \leq_{ae} n$ and that $1 \leq_{ae} log(log(n))$, then $log(n) \leq_{ae} n * log(log(n))$.
Second, we know that $logn \leq_{ae} n$ and as log is an increasing function, we can apply the function to both sides without changing the growth, so $log(logn) \leq_{ae} log(n)$. Now, by applying multiplicativity with n, we obtain $nlog(log(n)) \leq_{ae} nlog(n)$.
Third, following multiplicativity once again with $log(n) \leq_{ae} n$, it results in $n * log(n) \leq_{ae} n^2$. Then by more multiplicativity and additivity, we obtain $n * log(n) \leq_{ae} 2n^2 - 4n$ Fourth, we notice from additivity that $2n^2 - 4n \in O(2n^2)$ and from this, that $2 \leq_{ae} log(n)$. So, from multiplicativity, we can observe that $2n^2 \leq_{ae} n^2 * log$, this means that $2n^2 - 4n \leq_{ae} n^2 * log(n)$
From the last property of the exercise 3, we can notice that $n^2 \in 2^n$, this means that n^2 is asymptotically much smaller than 2^n. Thus,$n^2 * log(n) \in 2^n$