Exercise Sheet 3: Answers
COMS10017 Algorithms 2023/2024

Reminder: $\log n$ denotes the binary logarithm, i.e., $\log n = \log_2 n$.

Example Question: Loop Invariants

Question. Prove that the stated invariant holds throughout the execution of the loop (using the Initialization, Maintenance, Termination approach discussed in the lectures):

Algorithm 1

Require: Array A of length n ($n \geq 2$)

```
1: S ← A[0] − A[1]
2: for i ← 1 . . . n − 2 do
3:     S ← S + A[i] − A[i + 1]
4: end for
5: return S
```

Invariant:

At the beginning of iteration i, the statement $S = A[0] - A[i]$ holds.

Which value is returned by the algorithm (use the Terminiation property for this)?

Solution. Let S_i be the value of S at the beginning of iteration i.

1. *Initialization* ($i = 1$): We need to show that the statement of the loop invariant holds for $i = 1$, i.e., the statement $S_1 = A[0] - A[1]$ holds before iteration $i = 1$. Observe that, in Line 1, S_1 is initialized as $S_1 \leftarrow A[0] - A[1]$. The loop invariant thus holds for $i = 1$.

2. *Maintenance*: Assume that the loop invariant holds for value i, i.e., $S_i = A[0] - A[i]$. We need to show that the loop invariant then also holds for value $i + 1$, i.e., we need to show that $S_{i+1} = A[0] - A[i + 1]$ holds. To this end, observe that in iteration i we execute the operation $S_{i+1} = S_i + A[i] - A[i + 1]$. Since $S_i = A[0] - A[i]$, we obtain $S_{i+1} = A[0] - A[i] + A[i] - A[i + 1] = A[0] - A[i + 1]$.

3. *Termination*: We have that, after the last iteration (or before the $(n-1)$th iteration that is never executed), $S_{n-1} = A[0] - A[n - 1]$ holds. The algorithm thus returns the value $A[0] - A[n - 1]$.

✓

1 Warm up: Proof by Induction

Consider the following sequence: $s_1 = 1, s_2 = 2, s_3 = 3$, and $s_n = s_{n-1} + s_{n-2} + s_{n-3}$, for every $n \geq 4$. Prove that the following holds:

$$s_n \leq 2^n \ .$$

Solution.

Base cases: We need to verify that the statement holds for $n \in \{1, 2, 3\}$, since s_n depends on s_{n-1}, s_{n-2}, and s_{n-3} (in particular, s_4 depends on s_3, s_2, s_1). This is easy to verify: $s_1 = 1 \leq 2^1, s_2 = 2 \leq 2^2$ and $s_3 = 3 \leq 2^3$.

Induction Hypothesis: We complete the proof using strong induction. The induction hypothesis is therefore as follows: For every $n' \leq n$ the statement $s_{n'} \leq 2^{n'}$ holds.

Induction Step: We need to show that the statement also holds for $n + 1$:

$$s_{n+1} = s_n + s_{n-1} + s_{n-2} \leq 2^n + 2^{n-1} + 2^{n-2} = 2^{n-2}(4 + 2 + 1) \leq 2^{n-2} \cdot 8 = 2^{n+1} \ .$$

✓

2 Loop Invariant

Prove that the stated loop invariant holds throughout the execution of the loop (using the Initialization, Maintenance, Termination approach discussed in the lectures):

Algorithm 2

Require: Array A of n positive integers

$B \leftarrow$ empty array of n integers
$B[0] \leftarrow A[0]$
for $i = 1 \ldots n - 1$ **do**
  if $A[i] > B[i - 1]$ **then**
    $B[i] \leftarrow A[i]$
  else
    $B[i] \leftarrow B[i - 1]$
  end if
end for
return $B[n - 1]$

Loop Invariant: At the beginning of iteration i, the following statement holds: For every $0 \leq j < i$: $B[j]$ is the maximum of the subarray $A[0, j]$, i.e., $B[j] = \max\{A[0], \ldots, A[j]\}$.

Which value is returned by the algorithm (use the Terminiation property for this)?

Hint: The Maintenance part requires a case distinction in order to deal with the if-else statement.

Solution.

- **Initialization:** We need to show that the loop invariant holds for $i = 1$. For $i = 1$, the loop invariant translates to "At the beginning of iteration $i = 1$, the following holds: For every $0 \leq j < 1$ (which implies that j only takes on the value 0), $B[0]$ is the maximum of the subarray $A[0]$". This is trivially true since, in Line 2 of the algorithm, we have $B[0] = A[0]$ and, hence, $B[0]$ is also the maximum of $\{A[0]\}$.

- **Maintenance:** We now assume that the loop invariant holds for iteration i, i.e., we have $B[j] = \max\{A[0], A[1], \dots, A[j]\}$, for every $0 \le j < i$, and we need to deduce that the loop invariant then also holds for iteration $i+1$. Observe that in iteration i, only the value of $B[i]$ is updated. Hence, by induction, the statement of the loop-invariant is already trivially true for every $0 \le j < i$, and we only need to consider the remaining case $j = i$.

  To this end, we conduct a case distinction that reflects the if-else statement in the algorithm.

  - First, assume that $A[i] > B[i-1]$ holds. By induction, we know that the statement $B[i-1] = \max\{A[0], \dots, A[i-1]\}$ holds, which, together with the assumption $A[i] > B[i-1]$ implies $A[i] = \max\{A[0], \dots, A[i]\}$. In Line 5, we compute $B[i] \leftarrow A[i]$, and, thus, $B[i] = \max\{A[0], \dots, A[i]\}$ holds, which implies the loop invariant for $i+1$.
  - Next, suppose that $A[i] \le B[i-1]$ is true. Again, by induction, we know that the statement $B[i-1] = \max\{A[0], \dots, A[i-1]\}$ holds, which, together with the assumption $A[i] \le B[i-1]$ implies $B[i-1] = \max\{A[0], \dots, A[i-1], A[i]\}$. In Line 7, we compute $B[i] \leftarrow B[i-1]$, and, thus, $B[i] = \max\{A[0], \dots, A[i-1], A[i]\}$ holds, which implies the loop invariant for $i+1$.

- **Termination:** We evaluate the loop-invariant for $i = n$, which corresponds to the state of the algorithm after iteration $i = n-1$ (or before a virtual iteration $i = n$ that is never executed). We obtain that $B[j]$ is the maximum of $A[0, j]$, and, in particular, $B[n-1]$ is the maximum of A. The algorithm thus returns the maximum of the elements in A.

✓

3 Insertionsort

What is the runtime (in Θ-notation) of Insertionsort when executed on the following arrays of lengths n:

1. $1, 2, 3, 4, \dots, n-1, n$

   Solution. The runtime is $\Theta(n)$ since the inner loop of Insertionsort always requires time $\Theta(1)$ on this instance (no moves are needed). ✓

2. $n, n-1, n-2, \dots, 2, 1$

   Solution. The runtime is $\Theta(n^2)$. An easy way to see this is as follows: Consider the last $n/2$ elements of the input array. Each of these elements is moved at least $n/2$ positions to the left, i.e., the inner loop requires time $\Theta(n)$ for each of these elements. The total runtime is therefore $\Omega(\frac{n}{2} \cdot \frac{n}{2}) = \Omega(n^2)$. Since the runtime of Insertionsort is $O(n^2)$ on any instance, the runtime has to be $\Theta(n^2)$. ✓

3. The array A such that $A[i] = 1$ if $i \in \{1, 2, 4, 8, 16, \dots\}$ (i.e., when i is a power of two) and $A[i] = i$ otherwise.

Solution. Observe that Insertionsort does not move any of the elements (i.e., executes the inner loop) that are outside the positions $i \in \{1, 2, 4, 8, 16, \dots\}$. We thus only need to count the number of iterations of the inner loop for these positions. Observe further that the element at position 2^j, for some integer j, is moved at most 2^j steps to the left. Furthermore, we have that $2^{\lceil \log n \rceil} \ge 2^{\log n} = n$. Hence, there are at most $\lceil \log n \rceil$ positions in A with value 1. The total number of iterations the inner loop of Insertionsort is executed is therefore at most:

$$\sum_{j=0}^{\lceil \log n \rceil} 2^j = 2^{\lceil \log n \rceil + 1} - 1 \le 2^{\log n + 2} - 1 = 4n - 1 = \Theta(n) \ .$$

Here we used the inequality $\lceil \log n \rceil \le \log(n) + 1$, and the formula $\sum_{j=0}^{k} 2^j = 2^{k+1} - 1$.

The runtime therefore is $O(n)$. However, since our aim is give the runtime in Θ notation, we still need to argue that Insertionsort cannot be faster than $\Theta(n)$. This, however, we already know: As discussed in the lectures, the best-case runtime of Insertionsort is $\Theta(n)$. Hence, Insertionsort on array A has a runtime of $\Theta(n)$. ✓

4. The array B such that $B[i] = 1$ if $i \in \{10, 20, 30, 40 \dots\}$ (i.e., when i is a multiple of 10) and $B[i] = i$ otherwise.

Solution. Similar as in the previous exercise, only the elements at positions i that are a multiple of 10 are moved, and such an element is moved at most i steps. It is also important to note that each such element is moved at least $i/2$ steps. Hence, the runtime can be bounded from above by:

$$\sum_{i=10,20,30,\dots (i \le n-1)} i = \sum_{j=1}^{\lfloor \frac{n-1}{10} \rfloor} 10j = 10 \sum_{j=1}^{\lfloor \frac{n-1}{10} \rfloor} = 10 \cdot \frac{(\lfloor \frac{n-1}{10} \rfloor + 1) \lfloor \frac{n-1}{10} \rfloor}{2}$$

$$\le 10 \cdot \frac{(\frac{n-1}{10} + 1) \frac{n-1}{10}}{2} = \Theta((n-1)^2 + (n-1)) = \Theta(n^2) \ .$$

Similarly, the runtime can be bounded from below by:

$$\sum_{i=10,20,30,\dots (i \le n-1)} i/2 = \dots = \Theta(n^2) \ ,$$

where the calculation is almost identical to the previous calculation. Since the runtime is bounded from above and from below by $\Theta(n^2)$, the runtime therefore is $\Theta(n^2)$. ✓

5. The array C such that $C[i] = 1$ if $i \in \{n^{\frac{1}{10}}, 2 \cdot n^{\frac{1}{10}}, 3 \cdot n^{\frac{1}{10}}, \dots\}$ (i.e., when i is a multiple of $n^{\frac{1}{10}}$) and $C[i] = i$ otherwise. We assume here that $n^{\frac{1}{10}}$ is an integer.

Solution. $\Theta(n^{\frac{19}{10}})$. The approach is identical to the previous exercise, but the maths is slightly different. ✓

4 Runtime Analysis

Algorithm 3

Require: Integer $n \geq 2$
  $x \leftarrow 0$
  $i \leftarrow n$
  while $i \geq 2$ **do**
    $j \leftarrow \lceil n^{1/4} \rceil \cdot i$
    while $j \geq i$ **do**
      $x \leftarrow x + 1$
      $j \leftarrow j - 10$
    end while
    $i \leftarrow \lfloor i/\sqrt{n} \rfloor$
  end while
  return x

Determine the runtime of Algorithm 3 in Θ-notation.

Solution. Let us first determine the number of times x the inner loop is executed. The value of j evolves as follows:

$$\lceil n^{1/4} \rceil \cdot i, \lceil n^{1/4} \rceil \cdot i - 10, \lceil n^{1/4} \rceil \cdot i - 20, \ldots$$

until it reaches a value that is smaller than i. We thus have $\lceil n^{1/4} \rceil \cdot i - x \cdot 10 < i$ which yields $\frac{(\lceil n^{1/4} \rceil - 1) \cdot i}{10} < x$ and thus implies $x = \Theta(n^{1/4} i)$.

Next, concerning the outer loop, we see that the parameter i evolves as follows (disregarding the floor operation): $n, n/\sqrt{n} = \sqrt{n}, 1$. In fact, the iteration with $i = 1$ is never executed. The inner loop is thus executed only twice. The overall runtime therefore is:

$$\Theta(n^{1/4} n) + \Theta(n^{1/4} \sqrt{n}) + = \Theta(n^{5/4})$$

i.e., the runtime is dominated by the first iteration of the outer loop. ✓

5 Optional and Difficult Questions

Exercises in this section are intentionally more difficult and are there to challenge yourself.

5.1 Proof by Induction

Let n be a positive number that is divisible by 23, i.e., $n = k \cdot 23$, for some interger $k \geq 1$. Let $x = \lfloor n/10 \rfloor$ and let $y = n \ \% \ 10$ (the rest of an integer division). Prove by induction on k that 23 divides $x + 7y$.

Example: Consider $k = 4$. Then $n = 92$, $x = 9$ and $y = 2$. Observe that the quantity $x + 7y = 9 + 7 \cdot 2 = 23$ is divisible by 23.

Solution. We prove the statement by induction over k. To this end, let x_i be the value of x when $n = i \cdot 23$, and similarly, let y_i be the value of y when $n = i \cdot 23$.

Base case: $(k = 1)$
In this case, $n = 1 \cdot 23$, $x_1 = 2$ and $y_1 = 3$. The quantity $x_1 + 7y_1 = 23$, which is divisible by 23. ✓

Induction Hypothesis: Suppose that $x_i + 7y_i$ is divisible by 23.

Induction Step: We will show that $x_{i+1} + 7y_{i+1}$ is also divisible by 23. We conduct a case distinction:

- Suppose that $y_i \leq 6$. Then $y_{i+1} = y_i + 3$ and $x_{i+1} = x_i + 2$. We obtain:

  $$x_{i+1} + 7y_{i+1} = x_i + 2 + 7(y_i + 3) = x_i + 7y_i + 2 + 21 = x_i + 7y_i + 23 \ .$$

  Since $x_i + 7y_i$ is divisible by 23 and 23 is of course divisible by 23, we have $x_{i+1} + 7y_{i+1}$ is divisible by 23.

- Suppose that $y_i > 6$. Then, $y_{i+1} = y_i - 7$ and $x_{i+1} = x_i + 3$. We obtain:

  $$x_{i+1} + 7y_{i+1} = x_i + 3 + 7(y_i - 7) = x_i + 7y_i + 3 - 49 = x_i + 7y_i - 46 \ .$$

  Again, since $x_i + 7y_i$ is divisible by 23 and 46 is divisible by 23, we have $x_{i+1} + 7y_{i+1}$ is divisible by 23.

✓