Exercise Sheet 3

COMS10017 Algorithms 2020/2021

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

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 $s_{n-1}, s_{n-2}, s_{n-3}$ (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 $n + 1$:

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

$\checkmark$

2 Loop Invariant

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

Algorithm 1 Algorithm $\mathcal{A}$

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

$S \leftarrow A[0] - A[1]$
for $i \leftarrow 1 \ldots n - 2$ **do**
    $S \leftarrow S + A[i] - A[i + 1]$
end for
return S

Invariant:

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

What does the algorithm compute?

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

1. *Initialization* ($i = 1$): Observe that S is initialized as $A[0] - A[1]$. The loop invariant claims for $i = 1$ that $S_1 = A[0] - A[1]$, which is thus true.

2. *Maintenance*: Assume that the loop invariant holds in the beginning of iteration i, i.e., $S_i = A[0] - A[i]$. 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) $S = A[0]-A[n-1]$. The algorithm thus computes the value $A[0]-A[n-1]$.

✓

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 $2^{\lceil \log n \rceil} \geq 2^{\log n} = n$. 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:

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

   Here we used the inequality $\lceil \log n \rceil \leq \log(n) + 1$, and the formula $\sum_{j=0}^{k} 2^j = 2^{k+1} - 1$. The runtime therefore is $\Theta(n)$. ✓

4 Mergesort

The Mergesort algorithm uses the MERGE operation, which assumes that the left and the right halves of an array A are already sorted, and merges these two halves so that A is sorted afterwards. The runtime of this operation is $O(n)$.

Suppose that we replaced the MERGE operation in our Mergesort algorithm with a less efficient implementation that runs in time $O(n^2)$ (instead of $O(n)$). What is the runtime of our modified Mergesort algorithm?

Solution. Similar to the analysis in the lecture, we sum up the work in each level of the recursion tree. In level i, there are at most 2^{i-1} nodes, and the arrays in level i are of lengths at most $\lceil \frac{n}{2^{i-1}} \rceil$. The runtime in level i on a single node is then $O(\lceil \frac{n}{2^{i-1}} \rceil^2) = O(\frac{n^2}{2^{2(i-1)}})$. We thus obtain:

$$\sum_{i=1}^{\lceil \log n \rceil + 1} 2^{i-1} O(\frac{n^2}{2^{2(i-1)}}) = \sum_{i=1}^{\lceil \log n \rceil + 1} O(\frac{n^2}{2^{i-1}}) = O(n^2) \sum_{i=1}^{\lceil \log n \rceil + 1} \frac{1}{2^{i-1}} \leq O(n^2) \cdot 2 = O(n^2) \ ,$$

where we used the geometric series $\sum_{i=0}^{\infty} \frac{1}{2^i} = 2$. ✓

5 Runtime Analysis

Algorithm 2

```
Require: Integer n ≥ 2
  x ← 0
  i ← n
  while i ≥ 2 do
    j ← ⌈n^{1/4}⌉ · i
    while j ≥ i do
      x ← x + 1
      j ← j − 10
    end while
    i ← ⌊i/√n⌋
  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. ✓