Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department of Computer Science
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

20th October 2016

Data Structures & Algorithm — Solutions to Sheet 4 — AS 16

Solution 4.1 *Various exercises.*

a) There are two possible solutions: $x = 1511$, $y = 2609$ as well as $x = 926$, $y = 1115$.

b) After one iteration:

2	5	9	10	15	1	6	11	12	8	7	20
1	2	3	4	5	6	7	8	9	10	11	12

After two iterations:

1	2	5	9	10	15	6	11	12	8	7	20
1	2	3	4	5	6	7	8	9	10	11	12

c) As we can assume that n is a power of 3, we know that $n = 3^k$ for some $k \in \mathbb{N}$. By telescoping we get:

$$\begin{aligned}
T(n) &= 9 + 4T(n/3) \\
&= 9 + 4(9 + 4T(n/3^2)) \\
&= 9 + 4(9 + 4(9 + 4T(n/3^3))) \\
&= 9 + 4 \cdot 9 + 4^2 \cdot 9 + 4^3 \cdot T(n/3^3) \\
&= 9 \cdot (4^0 + 4^1 + 4^2) + 4^3 \cdot T(n/3^3) \\
&= \dots \\
&\overset{!}{=} 9 \cdot \sum_{i=0}^{k-1} 4^i + 4^k \cdot 6 = 9 \cdot \frac{4^k - 1}{4 - 1} + 4^k \cdot 6 = 9 \cdot 4^k - 3 = 9 \cdot 4^{\log_3(n)} - 3.
\end{aligned}$$

We prove using mathematical induction over n:

Base case ($n = 1$): For $n = 1$ it holds that $T(1) = 6 = 9 \cdot 4^{\log_3(1)} - 3$.

Induction hypothesis: We assume that $T(n) = 9 \cdot 4^{\log_3(n)} - 3$.

Inductive step ($n \to 3n$): For $n > 1$ it holds that

$$T(3n) = 9 + 4 \cdot T(n) \overset{\text{I.H.}}{=} 9 + 4 \cdot (9 \cdot 4^{\log_3(n)} - 3) = 9 + 9 \cdot 4^{\log_3(n)+1} - 12 = 9 \cdot 4^{\log_3(3n)} - 3.$$

Hint: Alternatively, we could have proven the hypothesis using mathematical induction over k.

Lösung 4.2 *Algorithm Design: Sums of Numbers.*

a) We check for every possible choice of a (there are n candidates) if $z - a$ is in the array (in this case we would have found two numbers with $a + b = z$). As A is sorted, we can check if $z - a$ is in A in $\mathcal{O}(\log n)$ Steps using binary search. If this number is in A, we have found a solution. Otherwise, we check the next choice of a. This leads to an algorithm with cost in $\mathcal{O}(n \log n)$.

b) We can get an algorithm with running time in $\mathcal{O}(n)$ by considering the following: Let l, r be the two indices of the left and the right end of the array (initially $l = 1$ and $r = n$). If $A[l] + A[r] = z$, we output $A[l]$ and $A[r]$ and terminate the algorithm. On the other hand, if $A[l] + A[r] > z$, we know that $A[k] + A[r] > z$ for every k with $l \leq k \leq r$ (it holds that $A[k] \geq A[l]$, as A is sorted). Hence, there is no number in the array from position l to position r that sums up to z together with $A[r]$. Therefore, it is sufficient to consider only elements with index smaller or equal $r - 1$. We set $r \leftarrow r - 1$ and repeat the procedure.

   If $A[l] + A[r] < z$, we know for sure that $A[l] + A[k] < z$ for every k with $l \leq k \leq r$. Hence, no number in the array from position l to position r leads to sum z when added to $A[l]$ (we know that $A[k] \leq A[r]$, as A is sorted). Therefore, it is sufficient to consider only elements with an index larger or equal to $l + 1$. We set $l \leftarrow l + 1$ and repeat the procedure.

   We terminate the procedure if we have either found two numbers that sum up to z or if $l = r$. We will always find a pair a, b with $a + b = z$ if such a pair exists. We can show the running time in $\mathcal{O}(n)$ by considering how $r - l$ changes: In every step that does not terminate the procedure, we either decrease r or increase l by one (but we never do both in the same step). That is, $r - l$ is decreased by one in every step. Initially, we have $r - l = n - 1$, and therefore the procedure terminates in at most $n - 1$ steps.