Eidgenössische
Technische Hochschule
Zürich

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

Department Informatik
Markus Püschel David Steurer Peter Widmayer
Chih-Hung Liu
Stefano Leucci

# Datenstrukturen & Algorithmen Blatt P7 HS 17

**Solution for Exercise P7.1** *Inversions.*

The problem can be solved by a variant of the Mergesort algorithm. In particular, we design a recursive procedure `SortAndCount` that takes an array $A = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ of $n$ distinct integers as input, sorts $A$, and returns the number of inversions in $A$. If $n \leq 1$ then `SortAndCount` is trivial since $A$ is already sorted and contains 0 inversions. If $n \geq 2$, Algorithm `SortAndCount` *splits* $A$ into two arrays $A_1 = \langle a_0, \ldots, a_{\lfloor \frac{n-1}{2} \rfloor} \rangle$ and $A_2 = \langle a_{\lceil \frac{n}{2} \rceil}, \ldots, a_{n-1} \rangle$ containing the first $\lceil \frac{n}{2} \rceil$ and the last $\lfloor \frac{n}{2} \rfloor$ elements of $A$, respectively.[1] Then, `SortAndCount` recursively invokes itself on $A_1$ (resp. $A_2$) to compute the number of inversions $\eta_1$ of $A_1$ (resp. $\eta_2$ of $A_2$) and the sorted version $A'_1$ of $A_1$ (resp. the sorted version $A'_2$ of $A_2$). Finally, $A'_1$ and $A'_2$ are *merged* into a single sorted vector $A'$ and the number of inversions of $A$ is computed as described in the following.

Observe that an inversion between two elements $a_i$ and $a_j$ (with $i < j$) in $A$ falls into one of three categories:

1. $i \leq \lfloor \frac{n-1}{2} \rfloor$ and $j \leq \lfloor \frac{n-1}{2} \rfloor$, i.e., $a_i$ and $a_j$ both belong to $A_1$.
2. $i \geq \lceil \frac{n}{2} \rceil$ and $j \geq \lceil \frac{n}{2} \rceil$, i.e., $a_i$ and $a_j$ both belong to $A_2$.
3. $i \leq \lfloor \frac{n-1}{2} \rfloor$ and $j \geq \lceil \frac{n}{2} \rceil$, i.e., $a_i$ belongs to $A_1$ and $a_j$ belongs to $A_2$.

Since the number of inversions of $A$ that fall into category 1 (resp. 2) is exactly $\eta_1$ (resp. $\eta_2$) we only need to find the number $\eta_3$ of inversions that fall in category 3. This can be done by a modification of the `Merge` procedure of Mergesort: initially $\eta_3 = 0$ and $A'$ is empty, then `Merge` iteratively compares the smallest element $x$ in $A'_1$ with the smallest element $y$ in $A'_2$.[2] If $x < y$ then $x$ is removed from $A'_1$ and added to $A'$. Notice that, in this case, all the elements in $A'_2$ are larger than $x$ and hence there are no inversions between $x$ and the remaining elements in $A'_2$. If $x > y$ then $y$ is removed from $A'_2$, $y$ is added to $A'$, and $\eta_3$ is incremented by $|A'_1|$. This is because $y$ is smaller than all the elements in $A'_1$ and therefore there are exactly $|A_1|$ inversions of category 3 between $y$ and the elements in $A'_1$.

Finally, `SortAndCount` copies $A'$ into $A$ and returns $\eta_1 + \eta_2 + \eta_3$.

[1] Notice that this operation creates no new inversions in $A_1$ and $A_2$.

[2] The case in which one of $A'_1$ and $A'_2$ is empty is easily handled by adding all the missing elements to $A'$.

**Solution for Exercise P7.2** *Mountain Trip.*

We first consider the following auxiliary problem: given a sorted vector $A = \langle a_1, a_2, \ldots, a_\eta \rangle$ of $m$ distinct integers and two additional numbers $x, y$, compute the number $N(A, x, y)$ of elements $a$ in $A$ such that $x \leq a \leq y$. This problem can be solved in $O(\log \eta)$ time by performing two binary searches on $A$: the first binary search looks for the largest index $i \leq m$ such that $a_i < x$, while the second binary search looks for the smallest index $j > 0$ such that $a_j > y$.[3]. The elements of $A$ between $x$ and $y$ are exactly the ones in the sub-array $\langle a_{i+1}, a_{i+2} \ldots, a_{j-2} \rangle$ and hence $N(A, x, y) = j - i + 1$.

To solve the original problem we first sort the arrays $\langle s_1, \ldots, s_S \rangle$ and $\langle m_1, \ldots, m_M \rangle$ containing the positions of the sea and mountain cities, respectively. Let $S'$ and $M'$ be the sorted vectors, respectively, and notice that this step requires $O(S \log S + M \log M)$ time (e.g., using Mergesort). Then, we examine one trip ad a time: when the $i$-th trip is considered we check whether $N(S', b_i, e_i) > 0$ and, if this is the case, the trip is ignored. Otherwise, if $N(S', b_i, e_i) = 0$, we compute the number $N(M', b_i, e_i)$ of mountain cities visited by the trip and we keep track of the best trip examined so far. The time required by this step is $O(\log S + \log M)$ per trip, therefore the total time spent by the algorithm is $O((T + S) \log S + (T + M) \log M) = O((M + S + T) \log(M + S))$.

[3] If $a_0 \geq x$ then let $i = 0$. If $a_m \leq y$ then let $j = m + 1$.