Finding Nearest Larger Neighbors  
A Case Study in Algorithm Design and Analysis

Tetsuo Asano\textsuperscript{1}, Sergey Bereg\textsuperscript{2}, and David Kirkpatrick\textsuperscript{3}

\textsuperscript{1} School of Information Science, JAIST, Japan  
\textsuperscript{2} Department of Computer Science, University of Texas at Dallas, USA  
\textsuperscript{3} Department of Computer Science, University of British Columbia, Canada

Abstract. Designing and analysing efficient algorithms is important in practical applications, but it is also fun and frequently instructive, even for simple problems with no immediate applications. In this self-contained paper we try to convey some of fun of algorithm design and analysis. Hopefully, the reader will find the discussion instructive as well.

We focus our attention on a single problem that we call the All Nearest Larger Neighbors Problem. Part of the fun in designing algorithms for this problem is the rich variety of algorithms that arise under slightly different optimization criteria. We also illustrate several important analytic techniques, including amortization, and correctness arguments using non-trivial loop invariants.

We hope, in this modest way, to reflect our deep admiration for the many contributions of Kurt Mehlhorn to the theory, practice and appreciation of algorithm design and analysis.

1 What Is the ANLN Problem?

Here is a general definition of our All Nearest Larger Neighbors (ANLN) Problem: Given a set $S$ of $n$ objects, find, for each object $x$ in $S$, an object $y$ in $S$ (if one exists) that is (i) larger than $x$ and (ii) at least as close to $x$ as any other object $z$ that is larger than $x$. We implicitly assume that any two objects are comparable, that is, one of them is larger than the other or they are equal. As we shall see, the problem is simplified considerably if we can assume that all elements are distinct.

Although the ANLN problem seems very natural and worthy of study even without specific applications, it is easy to imagine scenarios in which it could arise. For example, in emergency situations we often rely on protocols for sending messages from all individuals to some specified leader (or the reverse). It is easy to see the desirability of a tree-like protocol where (i) individual links are “short” and (ii) nodes closer to the root (leader) have greater authority (measured,
perhaps, by transmission power/capacity). To enact such a protocol, where each node has an assigned authority, it suffices to associate with every node its closest neighbor with higher authority.

2 The ANLN Problem for Linear Arrays

We begin with the simplest possible situation: the objects are \(n\) real numbers presented in an array \(A[1..n]\). For each element \(A[i]\), we want to determine an element \(A[j]\) among those with keys larger than \(A[i]\) that is closest to \(A[i]\), that is for which \(|j - i|\) is minimized, with ties broken in favor of the lower indexed neighbor.

Of course, an element with the largest key has no such larger neighbor. For this reason, as well as to simplify the presentation of some of our algorithms, we will assume that the array \(A\) has been implicitly extended to \(A[-n..3n]\), with \(A[-n] = A[3n] = \infty\) and \(A[j] = -\infty\) for \(j \in [-n + 1..0] \cup [n + 1..3n - 1]\). It should be clear that with this extension the nearest larger neighbors of all non-maximal elements of \(A[1..n]\) are unaltered and the nearest larger neighbor of all maximal elements of \(A[1..n]\) is \(A[-n]\).

[ANLN Problem for a Linear Array]

Input: An array \(A[1..n]\) of \(n\) real numbers.
Output: An array \(NLN[1..n]\) such that \(A[NLN[i]]\) is the nearest larger neighbor of \(A[i]\). If \(A[i]\) is a largest element in \(A[1..n]\) then \(NLN[i] = -n\).

Figure 1 shows an example of an array \(A\) containing 10 numbers together with its associated \(NLN\) array.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>87</td>
<td>32</td>
<td>12</td>
<td>54</td>
<td>28</td>
<td>35</td>
<td>14</td>
<td>61</td>
<td>18</td>
<td>53</td>
</tr>
<tr>
<td>(NLN)</td>
<td>-10</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Fig. 1. An example of an array \(A\) containing 10 elements together with its associated \(NLN\) array

2.1 A Simple Linear-Time Stack-Based Algorithm

The ANLN problem has a very straightforward linear-time solution by computing NLN values to both the left (LNLN) and right (RNLN), using a stack: