Solution 3.1  *Comparison of Sorting Algorithms.*

<table>
<thead>
<tr>
<th></th>
<th>bubbleSort</th>
<th></th>
<th>insertionSort</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>Comparisons</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>Input sequence</td>
<td>every</td>
<td>every</td>
<td>1,2,...,n</td>
<td>n,n-1,...,1</td>
</tr>
<tr>
<td>Permutations</td>
<td>0</td>
<td>$\Theta(n^2)$</td>
<td>0</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>Input Sequence</td>
<td>1,2,...,n</td>
<td>n,n-1,...,1</td>
<td>1,2,...,n</td>
<td>n,n-1,...,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>selectionSort</th>
<th></th>
<th>quicksort</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>Comparisons</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n \log n)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>Input sequence</td>
<td>every</td>
<td>every</td>
<td>(*)</td>
<td>1,2,...,n</td>
</tr>
<tr>
<td>Permutations</td>
<td>0</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n \log n)$</td>
</tr>
<tr>
<td>Input Sequence</td>
<td>1,2,...,n</td>
<td>n,n-1,...,1</td>
<td>1,2,...,n</td>
<td>(*)</td>
</tr>
</tbody>
</table>

\((\ast)\): An appropriate sequence is not easy to write. The sequence must be designed such that every chosen pivot halves the area that will be stored. For \(n = 7\), the sequence will be 4, 5, 7, 6, 2, 1, 3.

Solution 3.2  *Algorithm Design: Sums of Numbers.*

a) First, we sort \(A\) in ascending order. This takes \(O(n \log n)\) steps (using, for example, Mergesort/Heapsort). Then we check for any choice of \(a\) in the array (we have \(n\) candidates), whether the number \(z - a\) occurs in the array (in this case, we would have found two numbers such that \(a + b = z\)). Since \(A\) is sorted, we use binary search to determine whether \(z - a\) occurs in \(A\). This takes only \(O(\log n)\) steps. If this number occurs in \(A\), we found a solution, otherwise we try the next candidate, etc. In overall we get a running of \(O(n \log n)\).

*Note:* The above algorithm is asymptotically optimal. One could show that *every* algorithm for this problem requires \(\Omega(n \log n)\) steps (details can be found in “Refined Upper and Lower Bounds for 2-SUM”, A.C. Chan, W.I. Gasarch, C.P. Kruskal, 1997).

b) We can of course proceed exactly as in part a), except that we do not have to sort anymore. However, the running time would still be \(O(n \log n)\) because we have to invoke a binary search exactly \(n\) times to find the number \(z - a\).

A running time of \(O(n)\) can be achieved using the following consideration: let \(l, r\) be the indices of the left and right ends of the array. If \(A[l] + A[r] = z\), then we return \(A[l]\) and \(A[r]\) and terminate the procedure. If \(A[l] + A[r] > z\), then we have \(A[k] + A[r] > z\) for every \(k\) with \(l \leq k \leq r\) (we have \(A[k] \geq A[l]\) since \(A\) is sorted). Therefore, there are no elements in the array between positions \(l\) and \(r\) that sum to \(z\) with \(A[r]\). Thus it suffices...
to consider only elements in the array with indices $\leq r - 1$. We set $r := r - 1$ and repeat the procedure.

If, however, we have $A[l] + A[r] < z$, then we have $A[l] + A[k] < z$ for every $k$ with $l \leq k \leq r$. Therefore, there are no elements in the array between positions $l$ and $r$ that sum to $z$ with $A[l]$ (since the sequence is sorted). We can then consider only elements in the array with index $\geq l + 1$. We set $l := l + 1$ and repeat the procedure.

We stop the procedure if we find a solution, or if $l = r$. In this way, we will always find a pair $a, b$ with $a + b = z$ if such a pair exists. The running time of $O(n)$ can be proved by considering the development of $r - l$: every time the procedure does not terminate, either $r$ is decreased by one or $l$ is increased by one (but never both at the same time). That is, $r - l$ decreases by one in every step. In the first step we start with $r - l = n - 1$, and therefore the procedure terminates after at most $n - 1$ steps.

Note: This procedure could of course also be applied in part a), and it would be more efficient than the binary search we used there. However, since sorting alone requires $\Omega(n \log n)$ steps, the total asymptotic running time would not improve.

c) We obtain a running time of $O(n^2)$ by sorting the array first and using part b) after that: we try every of the $n$ possible numbers in $A$ as a candidate for $c$, and apply the algorithm from part b) for each of them. Additionally we have to make sure that $a$ and $b$ are different.

Note: It is unknown whether there exists an algorithm with a better running time.

Solution 3.3  Blum’s Median-of-Median Strategy.

a) We divide the first sequence in $\lfloor N/5 \rfloor$ groups of exactly 5 elements, and one group with 2 elements:

$$7, 12, 17, 3, 10 \quad 1, 6, 2, 4, 8 \quad 11, 9, 9, 6, 5 \quad 14, 20, 13, 1, 7 \quad 19, 8.$$  

The first recursive call is invoked on the medians of these groups. For the last group, the median is 19 by definition. Therefore the first sequence is

$$\begin{array}{c}
10, 4, 9, 13, 19
\end{array}$$

The result of this call is the median-of-medians 10. We use 10 as pivot element for a pivoting step similarly to the one in Quicksort (we first interchange 10 with 8, do the pivoting step and finally interchange 11 with 10):

$$7, 7, 1, 3, 8, 1, 6, 2, 4, 8, 5, 9, 9, 6 \quad 10 \quad 14, 20, 13, 17, 12, 19, 11$$

The first sequence has more elements than the second one has. Since we look for the element on the position $\lceil N/2 \rceil$, the second recursive call of Auswahl is invoked on the longer sequence

$$\begin{array}{c}
7, 7, 1, 3, 8, 1, 6, 2, 4, 8, 5, 9, 9, 6
\end{array}$$

Note: Depending on the implementation of the pivoting step, the elements of the sequence could be in a different order.

b) The first recursive call of the procedure Auswahl is always invoked on exactly $\lfloor N/5 \rfloor$ elements. Let $\nu$ be the median-of-medians. In the best case, $\nu$ is exactly the median we are looking for. Then the second call is not needed at all. Otherwise the recursion is invoked on the longer of the two subsequences, i.e. on a sequence having at least $\lceil N/2 \rceil$ elements.
For an upper bound we observe that at least \( \left\lceil \frac{1}{2} \left\lceil \frac{N}{5} \right\rceil \right\rceil - 1 \) many medians are smaller than \( \nu \). Considering that there might be group having less than 5 elements, then at least \( \left\lceil \frac{1}{2} \left\lceil \frac{N}{5} \right\rceil \right\rceil - 2 \) many groups have at least 3 elements smaller than \( \nu \). A group with less than 5 elements has at least one element smaller than \( \nu \), and the group containing \( \nu \) has at least two elements smaller than \( \nu \). Thus, there exist at least

\[
3 \left( \left\lceil \frac{1}{2} \left\lceil \frac{N}{5} \right\rceil \right\rceil - 2 \right) + 2 + 1 = 3 \left( \left\lceil \frac{1}{2} \left\lceil \frac{N}{5} \right\rceil \right\rceil - 1 \right)
\]

many elements smaller than \( \nu \). Analogously there exists at least the same number of elements that are greater than \( \nu \). Therefore the number of elements in the second recursive call is at most

\[
N - 3 \left( \left\lceil \frac{1}{2} \left\lceil \frac{N}{5} \right\rceil \right\rceil - 1 \right) \approx \frac{7}{10} N + 3.
\]