Exercise 3.1  

Comparison of Sorting Algorithms.

Let $A[1..n]$ be an array. Consider the following naive implementations of the sorting algorithms bubble sort, insertion sort, selection sort, and quicksort. These algorithms are called with the parameters $l = 1$ and $r = n$ to sort $A$ in ascending order.

```java
public void bubbleSort(int[] A, int l, int r) {
    for(int i=r; i>l; i--)
        for(int j=l; j<i; j++)
            if(A[j]>A[j+1])
                swap(A, j, j+1);
}

public void insertionSort(int[] A, int l, int r) {
    for(int i=l; i<r; i++) {
        int minJ = i;
        for(int j=i+1; j<=r; j++)
            if(A[j]<A[minJ])
                minJ = j;
        if(minJ != i)
            swap(A, i, minJ);
    }
}

public void selectionSort(int[] A, int l, int r) {
    for(int i=l; i<r; i++)
        for(int j=i-1; j>=l && A[j]>A[j+1]; j--)
            swap(A, j, j+1);
}

public void quicksort(int[] A, int l, int r) {
    if(l<r) {
        int i=l+1, j=r;
        do {
            while(i<j && A[i]<=A[l]) i++;
            while(i<=j && A[j]>=A[l]) j--;
            if(i<j) swap(A, i, j);
        } while(i<j);
        swap(A, l, j);
        quicksort(A, l, j-1);
        quicksort(A, j+1, r);
    }
}
```

The function `swap(A, i, j)` exchanges (swaps) the elements $A[i]$ and $A[j]$. For each of the above algorithms, estimate asymptotically both the minimum and the maximum number of performed swaps and comparisons of elements of $A$. For each of these cases, give an example sequence of the numbers $1, 2, \ldots, n$ for which the particular case occurs. The sequence should be described in such a way that any $n$ can be chosen arbitrarily (i.e., the descending sorted sequence can be described as $n, n-1, \ldots, 1$). Enter your results in a table of the following form.

<table>
<thead>
<tr>
<th></th>
<th>bubbleSort</th>
<th>insertionSort</th>
<th>selectionSort</th>
<th>quicksort</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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input sequence</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Permutations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input sequence</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please turn over.
Exercise 3.2  Questions About Sorting Algorithms.

Answer the following questions and give a brief explanation of your answer.

a) Does the sorted sequence 1, 2, ..., n implicitly represent a Min-Heap?

b) When all the elements in a Min-Heap are different, at which positions could the largest element be found?

c) A comparison-based algorithm is called stable if the relative order of identical elements is not changed. If the key ’5’, for example, occurs twice in an array, then the first 5 is never moved past the second 5. Which of the comparison-based sorting methods that you know are stable, or can easily be adapted accordingly?

d) A sorting algorithm is called in-situ if it requires only constant space in addition to the input sequence. Which sorting algorithms that you know are in-situ, or can easily be adapted accordingly?

e) The worst-case running time of Quicksort is $\Theta(n^2)$, while the worst-case running time of Mergesort is $\Theta(n \log n)$. Provide two reasons why, despite this fact, Quicksort is the more popular solution in practice.

Exercise 3.3  Extended Heaps.

In this exercise we are considering an array $A[1..|A|]$ representing a Min-Heap. We want to use $A$ to maintain a set of $n$ keys. Describe how the following operations can be implemented efficiently (i.e., with a running time in $O(\log n)$).

a) MIN: Computes the smallest key.

b) REPLACE($i$, $k$): Removes the key $A[i]$ and replaces it by $k$.

c) INSERT($k$): Inserts a new key with value $k$ in the heap.

d) DELETE($i$): Removes the key $A[i]$ from the heap.

Note: Of course you have to make sure that the heap property is maintained after each operation. You can assume that $A$ was chosen large enough to store all occurring keys.

Exercise 3.4  Algorithm Design: Sums of Numbers.

Let $A[1..n]$ be an array of natural numbers. For each of the following problems, provide an algorithm that is as efficient as possible, and determine its running time in the worst case.

a) Given a natural number $z$, does the array $A$ contain two entries $a$ and $b$ such that $a + b = z$?

b) Suppose that $A$ is sorted in ascending order. How efficiently can the problem from a) be solved now? Hint: In this case it is possible to achieve a better running time than in the previous case.

c) Does the array $A$ contain any three different entries $a$, $b$ and $c$ such that $a + b = c$?