Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

27th October 2016

Datenstrukturen & Algorithmen Exercise Sheet 6 AS 16

Hand-in: Thursday, 3rd November 2016 before the start of the lecture at 10:00 in the entrance area of ML D28. Please staple all sheets together and use this sheet as the cover page. Fill out the first two fields of the form below.

Exercise class (Room & Day):	
Submitted by:	
Corrected by:	
Bonus points:	

Exercise 6.1 *Comparison of Sorting Algorithms.*

Let $A[1..n]$ be an array. Consider the following Java 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.

```
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 selectionSort(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 insertionSort(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. Moreover, study how much space is used at least and at most. 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 preferably described in such a way that any n can be chosen arbitrarily. For example, the descending sorted sequence can be described as $n, n-1, \ldots, 1$.

Exercise 6.2 *Questions About Sorting Algorithms.*

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

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

b) 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?

c) 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?

d) 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 6.3 *Median-of-three Quicksort.*

We use quicksort with the *median-of-three strategy* (see the book, section 2.2.2 for a detailed description) to sort $A[l..r]$: First, we calculate the median of the elements a_l, a_m and a_r where $m = \lfloor (l+r)/2 \rfloor$, then we swap a_r with the median and use a_r as the pivot. Given n_0, provide a sequence of length $n \geq n_0$ for which the median-of-three quicksort compares $\Omega(n^2)$ keys. Prove that this sequence really leads to a quadratic number of key comparisons.

Exercise 6.4 *Algorithm design and Lower bounds.*

Given two arrays $A = (A[1], \ldots, A[n])$ and $B = (B[1], \ldots, B[n])$ both sorted in ascending order, we want to merge them to an array of size $2n$ that exactly contains the keys stored in A and B.

a) Analyze the exact number of comparisons that the merge routine of Mergesort needs.

b) Draw the process of the merge routine of Mergesort for two arrays of length 2 as a decision tree.

c) Show that *every* general deterministic algorithm for merging two sequences of length n needs at least $2n - \mathcal{O}(\log n)$ key comparisons in the worst case. Consider how many possible arrays can arise during the merging procedure. The estimate of exercise 2.2 might be helpful:

$$n \ln n - n \leq \ln(n!) \leq n \ln n - n + \mathcal{O}(\ln n).$$

d) **Tricky question:** Show that every general deterministic algorithm for merging two sequences needs actually $2n - 1$ key comparisons in the worst case. Consider the arrays $A = (1, 3, 5, \ldots)$ and $B = (2, 4, 6, \ldots)$.

Hint: It is not sufficient to show that the merge routine of Mergesort needs $2n - 1$ key comparisons for these two arrays. You have to prove the statement for a general algorithm.