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

Institut für Theoretische Informatik
Peter Widmayer
Thomas Tschager
Antonis Thomas

9th March 2016

# Datenstrukturen & Algorithmen — Exercise Sheet 3 — FS 16

## Exercise 3.1 *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 $A$. 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 3.2 *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 (not necessarily different) 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 $a + b = c$?

*Please turn over.*

**Exercise 3.3** *Blum's algorithm (**Programming Exercise**).*

In this exercise we are going to implement *Blum's algorithm* for median computation. Let $x_1, ..., x_n$ be a sequence of $n > 5$ elements (duplicates allowed). The algorithm finds the $k$-th smallest element by performing the following steps.

1) Sequentially, divide the elements into $\lfloor \frac{n}{5} \rfloor$ groups of 5 elements each and at most one group containing the remaining $n$ mod 5 elements. That means the first five elements go in the first group, etc.

2) For each of the above groups, find the median of the group. For a group with 2 elements, the median is the smaller one, and for a group with 4 elements, the median is the 2nd-smallest one.

3) Recursively compute the median $m$ among the above medians. This element is called the *median of medians.*

4) Use the partition step of quickselect to bring the element $m$ to the correct position $p_m$ in the sorted sequence. Then we have $p_m - 1$ elements on the left of $m$ (with value at most $m$), and $n - p_m$ elements on the right of $m$ (with value at least $m$).

5) If $k = p_m$, then we know that the pivot element is on the position we are looking for, and we return $m$. If $k < p_m$, then the $k$-th smallest element is located on the left of $m$, and we search recursively for the $k$-th smallest element among these $p_m - 1$ elements on the left. Otherwise, $k > p_m$, and we search recursively for the $(k - p_m)$-th smallest element among the $n - p_m$ elements on the right.

Our final goal is to compute the median, i.e. the $\lceil n/2 \rceil$-th element in the sorted sequence. For the sequence $3, 4, 2, 6, 4, 7, 1$, the median is 4.

**Input** The first line contains only the number $t$ of test instances. After that, we have exactly one line per test instance containing the numbers $n, x_1, ..., x_n$. While $n \in \mathbb{N}$, $1 \leq n \leq 1000$, describes the number of following integers, $x_i \in \mathbb{Z}$, $-10^8 \leq x_i \leq 10^8$ is the $i$-th number in the sequence.

**Output** For every test instance we output only one line. It contains the first sequence of medians of the groups of at most 5 elements, the first median of medians, and the overall median of the sequence.

**Example**

*Input:*

```
3
5 1 2 3 4 5
6 7 4 3 2 1 2
13 7 3 5 1 9 8 11 21 4 10 2 6 9
```

*Output:*

```
3 3 3
3 2 2 2
5 10 6 6 7
```

**Hand-in:** Wednesday, 16th March 2016 in your exercise group.