Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Übungen zur Vorlesung Informatik II (D-BAUG) FS 2017

D. Sidler, F. Friedrich

http://lec.inf.ethz.ch/baug/informatik2/2017

Solution to exercise sheet # 5 **20.3.2017 – 28.3.2017**

Problem 5.1. Compare sort algorithms.

Consider an array $A[1 \dots n]$ and the following (not necessarily optimal) Java implementations of the sort algorithms *Bubblesort*, *Insertion sort*, *Selection sort* and *Quicksort*.

The functions are called with $l = 1$ and $r = n$ to sort A in ascending order.

```java
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);
}

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);
}

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);
   }
}

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 the elements of A at positions i and j. Provide for each algorithm an asymptotic lower and upper bound on the number of calls to $swap$. Also give the sequence of integers $1, 2, \dots, n$ where these cases occur. Describe the sequence generically depending on n. For instance, describe the sequence of descending numbers as $n, n-1, \dots, 1$.

Submission link: https://codeboard.ethz.ch/inf2baugex05t01

Solution of Problem 5.1.

	bubbleSort		insertionSort	
	min	max	min	max
Comparisons	$\theta(n^2)$	$\theta(n^2)$	$\theta(n)$	$\theta(n^2)$
Sequence	any	any	$1, 2, \dots, n$	$n, n-1, \dots, 1$
Swaps	0	$\theta(n^2)$	0	$\theta(n^2)$
Sequence	$1, 2, \dots, n$	$n, n-1, \dots, 1$	$1, 2, \dots, n$	$n, n-1, \dots, 1$

	selectionSort		quicksort	
	min	max	min	max
Comparisons	$\theta(n^2)$	$\theta(n^2)$	$\theta(nlogn)$	$\theta(n^2)$
Sequence	any	any	$(\star)$	$1, 2, \dots, n$
Swaps	0	$\theta(n)$	$\theta(n)$	$\theta(nlogn)$
Sequence	$1, 2, \dots, n$	$n, n-1, \dots, 1$ $(\star\star)$	$1, 2, \dots, n$	$(\star)$

$(\star)$: It is not easy to write down a compact form. The sequence must be constructed such that every pivot halves the sorting range. For instance for $n = 7$ a sequence is: $4, 5, 7, 6, 2, 1, 3$.

$(\star\star)$: Even more swaps, exactly $n - 1$ and with that the highest possible count, *selectionSort* uses for the sequence $n, 1, 2, 3, \dots, n - 1$.

Problem 5.2. Exceptions

Open the code template at: https://codeboard.ethz.ch/inf2baugex05t02.

Your task is to open the file at the location "./Root/gedicht.txt", read the file and print out the number of lines which are not empty.

For reading the file use the `FileReader` and `BufferedReader` provided by the `java.io` library.

To successfully compile the program you have to catch the `IOException`.

Advice: Look up the method `readLine()` of the class `BufferedReader` in the Java 8 API documentation[1]. To check if a string is empty use the function `isEmpty()` function.

To test your program un-comment the annotation `@RunTests`. Once you pass the test you can submit your program.

Solution of Problem 5.2.

```
/**
 * Main class of the Java program.
 *
 * For TESTING and SUBMITTING: Uncomment the @RunTests annotation
 * (Remove the two slashes at the beginning of line 13)
 *
 */
import java.util.Scanner;
import java.io.IOException;
import java.io.FileReader;
import java.io.BufferedReader;

//@RunTests
public class Main {
    public static void main(String[] args) {
        int nonEmptyLines = 0;
```

[1] http://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html

```
        try (BufferedReader br = new BufferedReader(new
            FileReader("./Root/gedicht.txt"))){
            String line;
            while ((line = br.readLine()) != null) {
                if (!line.isEmpty()){
                    nonEmptyLines++;
                }
            }
        } catch (IOException e){
            e.printStackTrace();
            System.out.println("Datei konnte nicht gelesen werden!");
        }

        System.out.println(nonEmptyLines);
    }
}
```

Problem 5.3. Statistics on a Stream of Data ("Online Algorithm")

Open the code template at: https://codeboard.ethz.ch/inf2baugex05t03a.

Assume you have a continuous stream of data (we simulate this with user input). For each incoming data point of type double, you are asked to compute the mean $\mu_n = \sum_{i=1}^{n} \frac{x_i}{n}$ and sum of squares $\sigma_n^2 = \sum_{i=1}^{n}(x_i - \mu_n)^2$ of the data received until now.

Make sure your algorithm can do this in $\mathcal{O}(1)$ for each new data point arriving. Hint: think about how to compute μ_{n+1} from μ_n and x_{n+1} (write down the sum formula) and try to do the same for σ_{n+1}^2.

Once you have implemented the functionality, you can test your program by un-commenting the annotation `@RunTests`. Once you pass the test you can submit your program.

Question: can you implement a similar algorithm for the median? If so, how? If not, why not?
Provide your answer here: https://codeboard.ethz.ch/inf2baugex05t03b

Solution of Problem 5.3.

a)

```
/**
 * Main class of the Java program.
 *
 * For TESTING and SUBMITTING: Uncomment the @RunTests annotation
 * (Remove the two slashes at the beginning of line 10)
 *
 */
import java.util.Scanner;

@RunTests
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        //Compute mean and sum of squares on data stream
        // Read in first value
        double x = scanner.nextDouble();
```

```
        double mean = x;
        double sumOfSquares = 0;
        int n = 1;
        while (scanner.hasNext()) {
            x = scanner.nextDouble();
            n++;
            double oldMean = mean;
            mean = (((double)1/(double)n) * x) + (((double)(n-1)/(double)n) *
                oldMean);

            sumOfSquares = sumOfSquares + ((x - oldMean) * (x - mean));

            //Print out mean and sum of squares for n
            System.out.printf("n: %d; mean: %.2f; sum of squares: %.2f\n", n,
                mean, sumOfSquares);
        }
    }
}
```

b)

No, there is no similar algorithm. To find the median (50th percentile) you need to store all the values: there is, however, an $\mathcal{O}(\log n)$ (for each data point inserted) algorithm that makes use of a Heap data structure treated later in this course.