Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Self-Assessment

Informatik II (D-BAUG)

Felix Friedrich

ETH Zürich, April 2018.

Name, Vorname:	. . . . .
Legi-Nummer:	. . . . .

Diese **Selbsteinschätzung** dient Ihrer und unserer Orientierung. Sie wird eingesammelt, korrigiert und vertraulich behandelt. Sie hat aber keinen Einfluss auf eine spätere Leistungsbewertung. **Sie haben 40 Minuten Zeit.**

Das folgende Kleingedruckte finden Sie auch auf einer "scharfen" Prüfung.

Allgemeine Richtlinien:

1. Dauer der Prüfung: 40 Minuten.
2. Erlaubte Unterlagen: Wörterbuch (für gesprochene Sprachen). 4 A4 Seiten handgeschrieben oder $\geq$ 11pt Schriftgrösse.
3. Benützen Sie einen Kugelschreiber (blau oder schwarz) und keinen Bleistift. Bitte schreiben Sie leserlich. Nur lesbare Resultate werden bewertet.
4. Lösungen sind direkt auf das Aufgabenblatt in die dafür vorgesehenen Boxen zu schreiben (und direkt darunter, falls mehr Platz benötigt wird). Ungültige Lösungen sind deutlich durchzustreichen! Korrekturen bei Multiple-Choice Aufgaben bitte unmissverständlich anbringen!
5. Es gibt keine Negativpunkte für falsche Antworten.
6. Störungen durch irgendjemanden oder irgendetwas melden Sie bitte sofort der Aufsichtsperson.
7. Wir sammeln die Prüfung zum Schluss ein. Wichtig: stellen Sie unbedingt selbst sicher, dass Ihre Prüfung von einem Assistenten eingezogen wird. Stecken Sie keine Prüfung ein und lassen Sie Ihre Prüfung nicht einfach am Platz liegen. Dasselbe gilt, wenn Sie früher abgeben wollen: bitte melden Sie sich lautlos, und wir holen die Prüfung ab. Vorzeitige Abgaben sind nur bis 15 Minuten vor Prüfungsende möglich.
8. Wenn Sie zur Toilette müssen, melden Sie dies einer Aufsichtsperson durch Handzeichen.
9. Wir beantworten keine inhaltlichen Fragen während der Prüfung. Kommentare zur Aufgabe schreiben Sie bitte auf das Aufgabenblatt.

General guidelines:

1. *Exam duration: 40 minutes.*
2. *Permitted examination aids: dictionary (for spoken languages). 4 A4 pages hand written or $\geq$ 11pt font size*
3. *Use a pen (black or blue), not a pencil. Please write legibly. We will only consider solutions that we can read.*
4. *Solutions must be written directly onto the exam sheets in the provided boxes (and directly below, if more space is needed). Invalid solutions need to be crossed out clearly. Provide corrections to answers of multiple choice questions without any ambiguity!*
5. *There are no negative points for false answers.*
6. *If you feel disturbed by anyone or anything, let the supervisor of the exam know immediately.*
7. *We collect the exams at the end. Important: you must ensure that your exam has been collected by an assistant. Do not take any exam with you and do not leave your exam behind on your desk. The same applies when you want to finish early: please contact us silently and we will collect the exam. Handing in your exam ahead of time is only possible until 15 minutes before the exam ends.*
8. *If you need to go to the toilet, raise your hand and wait for a supervisor.*
9. *We will not answer any content-related questions during the exam. Please write comments referring to the tasks on the exam sheets.*

Question:	1	2	3	Total
Points:	22	10	8	40
Score:				

Generelle Anmerkung / *General Remark*

Verwenden Sie die Notation, Algorithmen und Datenstrukturen aus der Vorlesung. Falls Sie andere Methoden verwenden, müssen Sie diese kurz so erklären, dass Ihre Ergebnisse nachvollziehbar sind.

Use notation, algorithms and data structures from the course. If you use different methods, you need to explain them such that your results are reproducible.

Aufgabe 1: Verschiedenes (22P)

1) In dieser Aufgabe sollen nur Ergebnisse angegeben werden. Sie können sie direkt bei den Einzelteilen notieren.

   In this task only results have to be provided. You can note them down in the parts.

2) Als Ordnung verwenden wir für Buchstaben die alphabetische Reihenfolge, für Zahlen die aufsteigende Anordnung gemäss ihrer Grösse.

   As the order of characters we take the alphabetical order and for numbers we take the ascending order according to their sizes.

/1P (a) Gegeben sind Daten der Länge n. Wie viele Vergleiche sind minimal nötig, um das Maximum des Datensatzes zu finden?

Given data with length n. How many comparisons are minimally required to determine the maximum of the data set?

Minimale Anzahl Vergleiche/*Minimal number of comparisons*:

/3P (b) Nachfolgend sehen Sie drei Folgen von Momentaufnahmen (Schritten) der Algorithmen (a) Sortieren durch Einfügen, (b) Sortieren durch Auswahl und (c) Bubblesort. Geben Sie unter den Folgen jeweils den Namen des zugehörigen Algorithmus an.

Consider the following three sequences of snap-shots (steps) of the algorithms (a) Insertion Sort, (b) Selection Sort and (c) Bubblesort. Below each sequence provide the corresponding algorithm name.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

(c) Im folgenden ist eine Hashtabelle dargestellt (nachdem die Schlüssel 7, 9, 10 und 19 bereits eingefügt wurden). Die Hashfunktion ist $h(k) = k \bmod 11$. Kollisionen werden mit linearem Sondieren aufgelöst. Die Sondierung läuft immer nach rechts. Fügen Sie die folgenden Schlüssel in die (teilweise schon belegte) Hashtabelle in. Wie viele Kollisionen treten dabei (beim Einfügen der folgenden drei Zahlen) auf?

In the following a hash table is displayed (after keys 7, 9, 10 and 19 had been inserted previously). The hash function is $h(k) = k \bmod 11$. Collisions are resolved using linear probing. Probing always goes right. Insert the following keys into the (partially occupied) hash table. In doing so (in inserting the following three keys), how many collisions take place?

/4P

Einzufügende Schlüssel/*Keys to be inserted*: 18, 22, 20

							7	19	9	10
0	1	2	3	4	5	6	7	8	9	10

Anzahl Kollisionen/*Number of collisions*: ◯ 7 ◯ 8 ◯ 9 ◯ 10 ◯ 11

(d) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. Korrekte Antworten ergeben 1 Punkt. Inkorrekte oder fehlende Antworten ergeben keinen Punkt.

Mark in the following if the statements are true or false. Correct answers provide 1 points. Incorrect or missing answers provide no point.

/2P

Jeder vergleichsbasierte Sortieralgorithmus benötigt im schlechtesten Fall $O(n \log n)$ Vergleiche.

Any comparison based sorting algorithm requires $O(n \log n)$ key comparisons in the worst case.

◯ wahr/*true* ◯ falsch/*false*

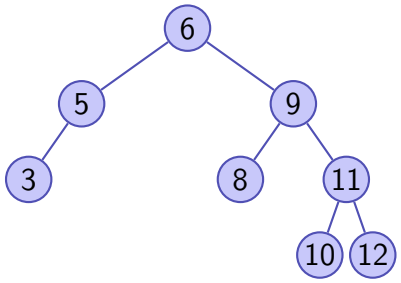
Werden in einen binären Suchbaum ausschliesslich Schlüssel eingefügt und keine entfernt, so entspricht die Preorder-Reihenfolge genau der Einfügereihenfolge.

If in a binary search tree keys are only inserted but none is removed, then the pre-order matches the insertion order exactly.

◯ wahr/*true* ◯ falsch/*false*

/4P (e) Fügen Sie in untenstehendem binären Suchbaum zuerst den Schlüssel 4 ein und löschen Sie danach im entstandenen Suchbaum den Schlüssel 6.

First insert into the Binary Search Tree below the key 4. After insertion, delete the key 6 from the created binary search tree.



Nach Einfügen von 4 / *After insertion of 4*

Nach Löschen von 6 / *After deletion of 6*

/4P (f) Zeichnen Sie einen binären Suchbaum, dessen Post-Order Traversierung die folgende Folge ergibt:

Draw a binary search tree where a post-order traversing would provide the following sequence:

2, 5, 3, 9, 11, 10, 8, 7

(g) Die folgende Tabelle enthält einen Min-Heap mit 9 Einträgen (als Array gespeichert), welcher die geraden Zahlen von 2 bis 18 enthält. Geben Sie an, wie die der Min-Heap aussieht, wenn man mit dem Algorithmus der Vorlesung die Zahl 0 hinzugefügt hat.

The following table contains a Min-Heap with 9 entries (represented as array). The Min-Heap contains the even numbers from 2 to 18. Specify the min-heap after the number 0 has been inserted according to the algorithm from the lectures.

/4P

[index]	1	2	3	4	5	6	7	8	9	10
initial	2	4	6	8	10	12	14	16	18	–
nach/*after* Insert(0)										

Aufgabe 2: Asymptotik (10P)

/4P (a) Geben Sie für die untenstehenden Funktionen eine Reihenfolge an, so dass folgendes gilt: Wenn eine Funktion f links von einer Funktion g steht, dann gilt $f \in \mathcal{O}(g)$.
Beispiel: die drei Funktionen n^3, n^5 und n^7 sind bereits in der richtigen Reihenfolge, da $n^3 \in \mathcal{O}(n^5)$ und $n^5 \in \mathcal{O}(n^7)$.

Provide an order for the functions below such that the following holds: If a function f is left of a function g then it holds that $f \in \mathcal{O}(g)$. Example: the functions n^3, n^5 and n^7 are already in the correct order because $n^3 \in \mathcal{O}(n^5)$ and $n^5 \in \mathcal{O}(n^7)$.

$$n^2 + 2n + 1, \quad n \sum_{i=0}^{n} i, \quad n \log n^n, \quad \sqrt{n} \log n, \quad n \log n^2, \quad n!$$

In den folgenden Aufgabenteilen wird jeweils angenommen, dass die Funktion g mit $g(n)$ aufgerufen wird. Geben Sie jeweils die asymptotische Anzahl von Aufrufen der Funktion $f()$ in Abhängigkeit von $n \in \mathbb{N}$ mit Θ-Notation möglichst knapp an. Die Funktion f ruft sich nicht selbst auf. Sie müssen Ihre Antworten nicht begründen.

In the following parts of this task we assume that the function g is called as $g(n)$. Provide the asymptotic number of calls of $f()$ depending on $n \in \mathbb{N}$ using Θ notation as tight as possible. The function f does not call itself. You do not have to justify your answers.

(b) /1P

```
void g(int n){
    for (int i = 0; i<n ; ++i ){
        if (i < 10){
            f();
        }
    }
}
```

Anzahl Aufrufe von f / *Number of calls of f*

(c) /2P

```
void g(int n){
    for (int i = 0; i<n ; ++i ){
        for (int j=0; j<i; ++j){
            f();
        }
    }
}
```

Anzahl Aufrufe von f / *Number of calls of f*

(d) /3P

```
void g(int n){
    if (n > 1){
        g(n/2);
    }
    f();
}
```

Anzahl Aufrufe von f / *Number of calls of f*

Aufgabe 3: Code Schreiben (8P)

Vervollständigen Sie die folgende Methode `filteredList`, welche eine Liste von Strings `list` entgegennimmt und einen Teil der Liste zurückgibt, sodass:

/6P (a) diejenigen Strings als Liste zurückgegeben werden, welche mit Kleinbuchstaben 'a' beginnen und aus genau drei Buchstaben bestehen, und

/2P (b) zusätzlich zu (a) alle Strings, welche mit dem Buchstaben 'a' beginnen, in einer eigenen Zeile auf System.out geschrieben werden (auch wenn sie mehr oder weniger als drei Buchstaben enthalten).

Hinweis: Benutzen Sie Java 8 Lambdas, Streams und die angehängte Dokumentation.

Complete the following method `filteredList`, which takes as argument a List of Strings `list` and returns a part of that List such that:

the returned subset list consists of exactly those strings which start with the letter 'a' (lower case) and have exactly 3 letters, and

additionally to (a) all Strings starting with the lower case letter 'a' are printed to individual lines of System.out (even if these strings contain less or more than three characters).

TIP: Use Java 8 Lambdas, Streams API's and consult the docs.

```
// pre: a List<String> list
// post: a subset List<String> containing all strings
// of the form 'a..' (a + two characters)
public List<String> filteredList(List<String> list) {
    return list.stream()
}
```

java.util.stream

Interface Stream<T>

Type Parameters:

`T - the type of the stream elements`

All Superinterfaces:

`AutoCloseable, BaseStream<T,Stream<T>>`

```
public interface Stream<T>
extends BaseStream<T,Stream<T>>
```

A sequence of elements supporting sequential and parallel aggregate operations. The following example illustrates an aggregate operation using `Stream` and `IntStream`:

```
int sum = widgets.stream()
                 .filter(w -> w.getColor() == RED)
                 .mapToInt(w -> w.getWeight())
                 .sum();
```

In this example, `widgets` is a `Collection<Widget>`. We create a stream of `Widget` objects via `Collection.stream()`, filter it to produce a stream containing only the red widgets, and then transform it into a stream of `int` values representing the weight of each red widget. Then this stream is summed to produce a total weight.

Method Summary

All Methods **Static Methods** **Instance Methods**

Abstract Methods **Default Methods**

Modifier and Type	Method and Description
`<R,A> R`	**`collect(Collector<? super T,A,R> collector)`** Performs a **mutable reduction** operation on the elements of this stream using a `Collector`.
`static <T> Stream<T>`	**`concat(Stream<? extends T> a, Stream<? extends T> b)`** Creates a lazily concatenated stream whose elements are all the elements of the first stream followed by all the elements of the second stream.
`long`	**`count()`** Returns the count of elements in this stream.
`Stream<T>`	**`distinct()`** Returns a stream consisting of the distinct elements (according to **`Object.equals(Object)`**) of this stream.
`static <T> Stream<T>`	**`empty()`** Returns an empty sequential `Stream`.
`Stream<T>`	**`filter(Predicate<? super T> predicate)`** Returns a stream consisting of the elements of this stream that match the given predicate.
`Optional<T>`	**`findAny()`** Returns an **`Optional`** describing some element of the stream, or an empty `Optional` if the stream is empty.
`Optional<T>`	**`findFirst()`** Returns an **`Optional`** describing the first element of this stream, or an empty `Optional` if the stream is empty.
`<R> Stream<R>`	**`flatMap(Function<? super T,? extends Stream<? extends R>> mapper)`** Returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element.
`void`	**`forEach(Consumer<? super T> action)`** Performs an action for each element of this stream.
`void`	**`forEachOrdered(Consumer<? super T> action)`** Performs an action for each element of this stream, in the encounter order of the stream if the stream has a defined encounter order.
`static <T> Stream<T>`	**`generate(Supplier<T> s)`** Returns an infinite sequential unordered stream where each element is generated by the provided `Supplier`.
`static <T> Stream<T>`	**`iterate(T seed, UnaryOperator<T> f)`** Returns an infinite sequential ordered `Stream` produced by iterative application of a function `f` to an initial element `seed`, producing a `Stream`

	consisting of `seed`, `f(seed)`, `f(f(seed))`, etc.
Stream<T>	**limit**(long maxSize) Returns a stream consisting of the elements of this stream, truncated to be no longer than `maxSize` in length.
<R> **Stream**<R>	**map**(**Function**<? super **T**,? extends R> mapper) Returns a stream consisting of the results of applying the given function to the elements of this stream.
DoubleStream	**mapToDouble**(**ToDoubleFunction**<? super **T**> mapper) Returns a `DoubleStream` consisting of the results of applying the given function to the elements of this stream.
IntStream	**mapToInt**(**ToIntFunction**<? super **T**> mapper) Returns an `IntStream` consisting of the results of applying the given function to the elements of this stream.
Optional<**T**>	**max**(**Comparator**<? super **T**> comparator) Returns the maximum element of this stream according to the provided `Comparator`.
Optional<**T**>	**min**(**Comparator**<? super **T**> comparator) Returns the minimum element of this stream according to the provided `Comparator`.
boolean	**noneMatch**(**Predicate**<? super **T**> predicate) Returns whether no elements of this stream match the provided predicate.
static <T> **Stream**<T>	**of**(T... values) Returns a sequential ordered stream whose elements are the specified values.
Stream<**T**>	**peek**(**Consumer**<? super **T**> action) Returns a stream consisting of the elements of this stream, additionally performing the provided action on each element as elements are consumed from the resulting stream.
Optional<**T**>	**reduce**(**BinaryOperator**<**T**> accumulator) Performs a **reduction** on the elements of this stream, using an **associative** accumulation function, and returns an `Optional` describing the reduced value, if any.
Stream<**T**>	**skip**(long n) Returns a stream consisting of the remaining elements of this stream after discarding the first `n` elements of the stream.
Stream<**T**>	**sorted**() Returns a stream consisting of the elements of this stream, sorted according to natural order.
Stream<**T**>	**sorted**(**Comparator**<? super **T**> comparator) Returns a stream consisting of the elements of this stream, sorted according to the provided `Comparator`.
Object[]	**toArray**() Returns an array containing the elements of this stream.

java.util.stream

Class Collectors

java.lang.Object
    java.util.stream.Collectors

```
public final class Collectors
extends Object
```

Implementations of `Collector` that implement various useful reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria, etc. Example:

```
// Accumulate names into a TreeSet
Set<String> set = people.stream().map(Person::getName).collect(Collectors.toCollection(TreeSet::new));
```

Method Summary

All Methods **Static Methods** **Concrete Methods**

Modifier and Type	Method and Description
static <T> `Collector<T,?,Double>`	**averagingDouble**(`ToDoubleFunction<? super T> mapper`) Returns a `Collector` that produces the arithmetic mean of a double-valued function applied to the input elements.
static <T,A,R,RR> `Collector<T,A,RR>`	**collectingAndThen**(`Collector<T,A,R> downstream, Function<R,RR> finisher`) Adapts a `Collector` to perform an additional finishing transformation.
static <T> `Collector<T,?,Long>`	**counting**() Returns a `Collector` accepting elements of type `T` that counts the number of input elements.
static <T,K> `Collector<T,?,Map<K,List<T>>>`	**groupingBy**(`Function<? super T,? extends K> classifier`) Returns a `Collector` implementing a "group by" operation on input elements of type `T`, grouping elements according to a classification function, and returning the results in a `Map`.
static `Collector<CharSequence,?,String>`	**joining**(`CharSequence delimiter`) Returns a `Collector` that concatenates the input elements, separated by the specified delimiter, in encounter order.
static <T,U,A,R> `Collector<T,?,R>`	**mapping**(`Function<? super T,? extends U> mapper, Collector<? super U,A,R> downstream`) Adapts a `Collector` accepting elements of type `U` to one accepting elements of type `T` by applying a mapping function to each input element before accumulation.
static <T> `Collector<T,?,Optional<T>>`	**maxBy**(`Comparator<? super T> comparator`) Returns a `Collector` that produces the maximal element according to a given `Comparator`, described as an `Optional<T>`.
static <T> `Collector<T,?,Optional<T>>`	**minBy**(`Comparator<? super T> comparator`) Returns a `Collector` that produces the minimal element according to a given `Comparator`, described as an `Optional<T>`.
static <T> `Collector<T,?,Map<Boolean,List<T>>>`	**partitioningBy**(`Predicate<? super T> predicate`) Returns a `Collector` which partitions the input elements according to a `Predicate`, and organizes them into a `Map<Boolean, List<T>>`.
static <T> `Collector<T,?,Optional<T>>`	**reducing**(`BinaryOperator<T> op`) Returns a `Collector` which performs a reduction of its input elements under a specified `BinaryOperator`.
static <T> `Collector<T,?,DoubleSummaryStatistics>`	**summarizingDouble**(`ToDoubleFunction<? super T> mapper`) Returns a `Collector` which applies an `double`-producing mapping function to each input element, and returns summary statistics for the resulting values.
static <T> `Collector<T,?,Double>`	**summingDouble**(`ToDoubleFunction<? super T> mapper`) Returns a `Collector` that produces the sum of a double-valued function applied to the input elements.
static <T,C extends `Collection<T>>` `Collector<T,?,C>`	**toCollection**(`Supplier<C> collectionFactory`) Returns a `Collector` that accumulates the input elements into a new `Collection`, in encounter order.
static <T,K,U> `Collector<T,?,ConcurrentMap<K,U>>`	**toConcurrentMap**(`Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper`) Returns a concurrent `Collector` that accumulates elements into a `ConcurrentMap` whose keys and values are the result of applying the provided mapping functions to the input elements.
static <T> `Collector<T,?,List<T>>`	**toList**() Returns a `Collector` that accumulates the input elements into a new `List`.
static <T,K,U> `Collector<T,?,Map<K,U>>`	**toMap**(`Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper`) Returns a `Collector` that accumulates elements into a `Map` whose keys and values are the result of applying the provided mapping functions to the input elements.
static <T> `Collector<T,?,Set<T>>`	**toSet**() Returns a `Collector` that accumulates the input elements into a new `Set`.

java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Method Summary

All Methods **Static Methods** **Instance Methods**

Modifier and Type	Method and Description
char	**charAt**(int index) Returns the char value at the specified index.
int	**compareTo**(**String** anotherString) Compares two strings lexicographically.
String	**concat**(**String** str) Concatenates the specified string to the end of this string.
boolean	**contains**(**CharSequence** s) Returns true if and only if this string contains the specified sequence of char values.
boolean	**equals**(**Object** anObject) Compares this string to the specified object.
static **String**	**format**(**String** format, **Object**... args) Returns a formatted string using the specified format string and arguments.
int	**hashCode**() Returns a hash code for this string.
int	**indexOf**(**String** str) Returns the index within this string of the first occurrence of the specified substring.
int	**lastIndexOf**(**String** str) Returns the index within this string of the last occurrence of the specified substring.
int	**length**() Returns the length of this string.
boolean	**matches**(**String** regex) Tells whether or not this string matches the given **regular expression**.
String[]	**split**(**String** regex) Splits this string around matches of the given **regular expression**.
boolean	**startsWith**(**String** prefix) Tests if this string starts with the specified prefix.
String	**substring**(int beginIndex, int endIndex) Returns a string that is a substring of this string.
char[]	**toCharArray**() Converts this string to a new character array.
String	**toLowerCase**() Converts all of the characters in this String to lower case using the rules of the default locale.
String	**toString**() This object (which is already a string!) is itself returned.
String	**toUpperCase**() Converts all of the characters in this String to upper case using the rules of the default locale.
String	**trim**() Returns a string whose value is this string, with any leading and trailing whitespace removed.
static **String**	**valueOf**(char c) Returns the string representation of the char argument.