# Übungsblatt 3, Vorkurs Informatik, WiSe 2024/25

(Maike Buchin & Michael Walter, mit Lösungen)

In dieser Übung können Sie mit den Suchalgorithmen aus der Vorlesung in Java experimentieren und sich auch eigene Algorithmen ausdenken.

## Aufgabe 1: Lineare vs. binäre Suche

Betrachten Sie das folgende Java-Programm:

```
public class Main {
    static int search(int[] array, int x) {
        for (int i = 0; i < array.length; ++i)
            if (array[i] == x)
                return i;
        return -1;
    }

    public static void main(String[] args) {
        final int N = 1000;
        int[] zahlen = new int[N];
        for (int i = 0; i < N; ++i)
            zahlen[i] = i;
        for (int i = 0; i < N; ++i) {
            if (search(zahlen, i) != i) {
                System.out.println("Fehler bei der Suche nach " + i);
                return;
            }
        }
        if (search(zahlen, -42) != -1) {
            System.out.println("Fehler bei der Suche nach -42");
            return;
        }
        System.out.println("Alles OK!");
    }
}
```

Die Funktion `search` implementiert den Algorithmus "Lineare Suche" aus der Vorlesung, um in einem Array nach einer Zahl zu suchen. Der Rückgabewert ist entweder der Index der Zahl im Array, oder -1, falls die Zahl nicht im Array enthalten ist. Die Funktion `main` testet anhand eines Beispielarrays mit $N = 1000$ Einträgen, ob der Algorithmus auch wirklich korrekt funktioniert.

a) Führen Sie das Programm auf www.programiz.com/java-programming/online-compiler aus. Diskutieren Sie mit Ihren Banknachbar*innen darüber, wie das Programm genau funktioniert.

b) Ändern Sie die Größe des Arrays nach $N = 1\,000\,000$ und führen Sie das Programm erneut aus. Was beobachten Sie? Erstaunt Sie, was Sie sehen?

c) Ändern Sie die Funktion `search`, so dass anstatt der "Linearen Suche" die "Binäre Suche" aus der Vorlesung implementiert wird.

*Lösung.* b) Das Programm ist ziemlich langsam – es gibt erst nach langem Warten eine Erfolgsmeldung aus (falls man geduldig genug ist).

Grund: Die Testroutine testet jeden Eintrag. Bei einem $O(N)$ Suchalgorithmus führt das also zu einer Laufzeit von $O(N^2)$. Bei $N = 1\ 000\ 000$ wartet man da eine Weile...

c)
```java
static int search(int[] array, int x) {
    int begin = 0, end = array.length;
    while (begin != end) {
        int i = (begin + end) / 2;
        int z = array[i];
        if (x == z)
            return i;
        if (x < z)
            end = i;
        else
            begin = i + 1;
    }
    return -1;
}
```

□

## Aufgabe 2: Ihr eigener Suchalgorithmus

a) Entwerfen Sie einen Algorithmus, der ein sortieres Array von Zahlen als Eingabe bekommt, und die erste Zahl zurückgibt, die echt größer als Null ist.

*Beispiel: Für das Array* $\{-500, -320, -40, 0, 30, 50, 80, 1200\}$ *sollte die Rückgabe* 30 *sein.*

b) Implementieren Sie Ihren Algorithmus in Java und testen Sie, ob dieser auch wirklich korrekt funktioniert.

c) Falls Ihr Algorithmus das Array einfach von links nach rechts durchläuft: Können Sie sich eine effizientere Lösung vorstellen, die ähnlich wie die "Binäre Suche" vorgeht?

*Lösung.*

```java
public class Main {
    // throws if no positive element in array
    static int findFirstPositive(int[] array) {
        int begin = 0, end = array.length;
        while (begin != end) {
            int i = (begin + end) / 2;
            int z = array[i];
            if (z > 0)
                end = i;
            else
                begin = i + 1;
        }
        return array[begin];
    }

    public static void main(String[] args) {
        int[] zahlen = {-500, -320, -40, 0, 30, 50, 80, 1200};
        System.out.println(findFirstPositive(zahlen));
    }
}
```

□

## Bonusaufgabe: Unendliche Arrays?

In der Vorlesung haben wir die "Exponentielle Suche" besprochen, die prinzipiell unendlich große sortierte Arrays durchsuchen kann. Es ist gar nicht so klar, wie man so etwas in einem Computer darstellen kann.

a) Diskutieren Sie mit Ihren Banknachbar*innen, wie man in Java ein Array modellieren könnte, das "unendlich viele" Einträge hat. So ein Array könnte beispielsweise alle Quadratzahlen enthalten (1, 4, 9, 16, 25, usw.).

b) Implementieren Sie den Algorithmus "Exponentielle Suche" aus der Vorlesung, um ein solches Array effizient nach Einträgen zu durchsuchen.

*Lösung.* Eine Möglichkeit wäre so etwas:

```java
// Aufgabenteil a)
interface InfiniteArray {
    int get(int index);
}

class Squares implements InfiniteArray {
    public int get(int index) { return index * index; }
}

// Aufgabenteil b)
public class Main {
    static int search(InfiniteArray array, int x) {
        // determine upper bound
        int end = 1;
        while (true) {
            int z = array.get(end);
            if (x == z) return end;
            if (x < z) break;
            end *= 2;
        }

        // binary search
        int begin = 0;
        while (begin != end) {
            int i = (begin + end) / 2;
            int z = array.get(i);
            if (x == z)
                return i;
            if (x < z)
                end = i;
            else
                begin = i + 1;
        }
        return -1;
    }

    public static void main(String[] args) {
        System.out.println(search(new Squares(), 36));
    }
}
```

□