Aufgabe 3

Die folgende Seite enthält Software-Quellcode, der einen Algorithmus zur binären Suche implementiert. Dieser ist durch Inspektion zu überprüfen. Im Folgenden sind die Regeln der Inspektion angegeben.

RM1	(Dokumentation)	Jede Quellcode-Datei beginnt mit einem Kommentar, der den Klassennamen, Versionsinformationen, Datum und Urheberrechtsangaben enthält.
RM2	(Dokumentation)	Jede Methode wird kommentiert. Der Kommentar enthält eine vollständige Beschreibung der Signatur so wie eine Design-by-Contract-Spezifikation.
RM3	(Dokumentation)	Deklarationen von Variablen werden kommentiert.
RM4	(Dokumentation)	Jede Kontrollstruktur wird kommentiert.
RM5	(Formatierung)	Zwischen einem Schlüsselwort und einer Klammer steht ein Leerzeichen.
RM6	(Formatierung)	Zwischen binären Operatoren und den Operanden stehen Leerzeichen.
RM7	(Programmierung)	Variablen werden in der Anweisung initialisiert, in der sie auch deklariert werden.
RM8	(Bezeichner)	Klassennamen werden groß geschrieben, Variablennamen klein.

(a) Überprüfen Sie durch Inspektion, ob die obigen Regeln für den Quellcode eingehalten wurden. Erstellen Sie eine Liste mit allen Verletzungen der Regeln. Geben Sie für jede Verletzung einer Regel die Zeilennummer, Regelnummer und Kommentar an, z. B.(07, RM4, while nicht kommentiert).[...]

(b) Entspricht die Methode `binarySearch` ihrer Spezifikation, die durch Vor- und Nachbedingungen angeben ist? Geben Sie gegebenenfalls Korrekturen der Methode an.

(c) Beschreiben alle Kommentare ab Zeile 24 die Semantik des Codes korrekt? Geben Sie zu jedem falschen Kommentar einen korrigierten Kommentar mit Zeilennummer an.

(d) Geben Sie den Kontrollflussgraphen für die Methode `binarySearch` an.

(e) Geben Sie maximal drei Testfälle für die Methode `binarySearch` an, die insgesamt eine vollständige Anweisungsüberdeckung leisten.

```
3  /**
4   * BinarySearch.java
5   *
6   * Eine Implementierung der "BinaereSuche" mit einem iterativen Algorithmus
7   */
8  class BinarySearch {
```

```java
  /**
   * BinaereSuche
   *
   * a: Eingabefeld
   *
   * item: zusuchendesElement
   *
   * returnValue: der Index des zu suchenden Elements oder -1
   *
   * Vorbedingung:
   *
   * a.length > 0
   *
   * a ist ein linear geordnetes Feld:
   *
   * For all k: (1 <= k < a.length) ==> (a[k-1] <=a [k])
   *
   * Nachbedingung:
   *
   * Wenn item in a, dann gibt es ein k mit a[k] == item und returnValue == k
   * Genau dann wenn returnValue == -1 gibt es kein k mit 0 <= k < a.length und
   * a[k]==item.
   */
  public static int binarySearch(float a[], float item) {
    int End;// exklusiver Index fuer das Ende des
    // zudurchsuchenden Teils des Arrays
    int start = 1; // inklusiver Index fuer den Anfang der Suche
    End = a.length;
    // Die Schleife wird verlassen, wenn keine der beiden Haelften das
    // Element enthaelt.
    while (start < End) {
      // Teilung des Arrays in zwei Haelften
      // untere Haelfte: [0,mid[
      // obere Haelfte: ]mid,End[
      int mid = (start + End) / 2;
      if (item > a[mid]) {
        // Ausschluss der oberen Haelfte
        start = mid + 1;
      } else if (item < a[mid]) {
        // Ausschluss der unteren Haelfte
        End = mid - 1;
      } else {
        // Das gesuchte Element wird zurueckgegeben
        return (mid);
      }
    } // end of while
      // Bei Misserfolg der Suche wird -1 zurueckgegeben
    return (-1);
  }
}
```