בית ספר אפי ארזי למדעי המחשב
המרכז הבינתחומי הרצליה

מבוא למדעי המחשב

מבחן גמר, מועד א'

2010

<u>הנחיות:</u>

- זמן המבחן: שלוש שעות.
- זמן הבחינה מוגבל, ויש לעבוד ביעילות. אם נתקעים בסעיף מסוים, כדאי לעזוב אותו ולרוץ הלאה.
- חומר סגור. השימוש במחשבים או במחשבונים אסור.
- ענו על כל השאלות <u>על טופס המבחן הנוכחי</u>. מומלץ להשתמש במחברות הבחינה כטיוטא. ניתן גם לכתוב תשובות במחברת הבחינה ולהפנות אליהן בצורה ברורה ומפורשת מטופס המבחן. חובה למחוק באופן ברור כל דבר שאינכם רוצים שנבדוק.
- בגמר הבחינה יש להחזיר את טופס המבחן + מחברת המבחן + כל דפי העזר הנלווים.
- אם תרגישו צורך לעשות הנחה מסוימת כדי לענות על כל שאלה שהיא, ניתן לעשות זאת, כל עוד ההנחה היא סבירה ומנוסחת בדיוק ובבהירות.
- אם אתם לא מסוגלים לתת תשובה מלאה לשאלה מסוימת, תנו תשובה חלקית. תשובה נכונה באופן חלקי תקבל ניקוד חלקי.
- התשובות חייבות להיות קצרות ולעניין. כתב היד חייב להיות קריא וברור. תשובות לא קריאות תקבלנה ציון 0.
- אם התבקשתם לכתוב קוד שאמור לפעול על קלט מסוים, אזי הקוד לא אמור לבדוק אם הקלט תקין, אלא אם כן נאמר כך בשאלה במפורש.
- באופן דומה, אם התבקשתם לכתוב מתודה שאמורה לפעול על ארגומנטים מסוימים, אזי המתודה לא אמורה לבדוק את תקינות הארגומנטים, אלא אם כן נאמר כך בשאלה במפורש.
- אין צורך לתעד את הקוד שאתם כותבים, אלא אם אתם רוצים להסביר לנו משהו על הקוד שכתבתם.
- הקוד שתכתבו יישפט, בין היתר, לפי האורך והאלגנטיות שלו. קוד ארוך או מסורבל ללא צורך יקבל פחות נקודות, אפילו אם הוא ממלא את המשימה שהוגדרה בשאלה.
- כל החומרים להם תזדקקו במהלך המבחן מתועדים בדפי העזר שקיבלת.
- לא יורדו נקודות על טעויות סינטקס טריוויאליות.

<u>בהצלחה!</u>

הערות על בדיקת המבחן

- פתרונות התיכנות שמופעים להלן אינם יחידים. כרגיל, יש יותר מתשובה נכונה אחת לרוב השאלות שדורשות כתיבת קוד. יחד עם זאת, תשובות ארוכות ומסורבלות באופן משמעותי מאלה שניתנות כאן גררו הורדת נקודות.
- לא הורדו נקודות על חריגות (exceptions) שאינן מלוות בטקסט שמסביר את החריגה. (האמת, היינו צריכים לדרוש מכם לתעד את החריגות, אבל שכחנו לציין זאת).
- חלק מהפתרונות שלנו מכילים הערות (comments). כמו שציינו בגוף המבחן, לא ציפינו מכם לכתוב הערות בפתרונות שלכם.
- איננו מצפים לקבל פתרון מדויק של כל שאלה, אלא פתרון שמספק את כל המרכיבים של הפתרון הנכון, גם אם הם כוללים טעויות סינטקס ואפילו טעויות לוגיות שוליות.
- ככלל, בדיקת המבחן הייתה מאד סלחנית וליברלית. מומלץ לחשוב טוב טוב לפני שמערערים על ציון המבחן, כי כל ערעור יגרור בדיקה חוזרת ומקיפה של כל המבחן.

מחלקת BigInt

כידוע, טיפוסי נתונים כגון int ו- double מוגבלים בגודל המספרים שניתן לייצגם. למשל, טיפוס הנתונים int מייצג מספרים בעזרת 32 סיביות (ביטים), ויש גבול לגודל המספר שייצוג כזה יכול לייצג. כדי להתמודד עם המגבלה הזאת, קיימת ב Java מחלקה בשם BigInteger שמייצגת מספרים שלמים גדולים כרצון המשתמש. במהלך המבחן נכתוב בהדרגה מחלקה דומה, בשם BigInt, שמייצגת מספרים שלמים חיוביים גדולים כרצוננו ומספקת שירותי אריתמטיקה על מספרים גדולים כאלה.

בדרך כלל נשתמש במחלקות כאלה רק כדי לייצג מספרים גדולים באמת, כמו למשל 669523849932130508876392554713407521319117239637943224980015676156491 (שהוא, דרך אגב, מספר ראשוני). אבל, במהלך המבחן, לצורכי debugging, ובכל השאלות שיינתנו להלן, נוח ומומלץ לחשוב שהמחלקה BigInt מייצגת מספרים קטנים בני כמה ספרות. קיראו את תיעוד המחלקה בדף העזר וענו על השאלות הבאות.

1. (3 נקודות) התבוננו בדף העזר שמתעד את מחלקת BigInt וכיתבו קטע קוד שמשתמש בשירותי המחלקה. בפרט, קטע הקוד צריך לייצר שני אובייקטי BigInt מהמחרוזות "12" ו- "173" ולהדפיס את הסכום שלהם. בונוס: בצעו את הפעולה הזאת בשורת קוד (פקודה) אחת.

```
BigInt x = new BigInt("12");
BigInt y = new BigInt("173");
System.out.println(x.add(y));
// One liner:
System.out.println(((new BigInt("12")).add(new BigInt("173"))));
```

פתרון נכון של שורת קוד (פקודה) אחת זכה ב 4 נקודות במקום 3.

השאלות הבאות עוסקות במימוש מחלקת BigInt. התבוננו בדף העזר שמתעד את מחלקת BigInt ושימו לב לשלושת ההערות הבאות:

- כמו שניתן לראות מדף העזר, המחלקה מייצגת את ספרות המספר במערך בשם digits מסוג []int. למשל, המספר 512 מיוצג ע"י המערך digits כאשר digits[0] = 5, digits[1] = 1, ו- digits[2] = 2.
- implements Comparable רלבנטי רק לחלק מהשאלות משאלה 20 ואילך וניתן להתעלם מהפרט הזה בכל השאלות 19-1.
- כל אחת מהשאלות הבאות מתייחסת למימוש של מתודה ספציפית אחת במחלקה. כשעונים על כל אחת מהשאלות האלה, צריך להניח שכל המתודות האחרות שמתועדות בדף העזר BigInt כבר ממומשות, ואפשר ורצוי להשתמש בהן, לפי הצורך.

2. (6 נקודות) הקונסטרקטור BigInt (int[] val) בונה אובייקט מסוג BigInt מהארגומנט, שאמור להיות מערך של מספרים שכל אחד מהם גדול או שווה לאפס וקטן או שווה מתשע. אם הארגומנט הוא null, או מכיל מספר שאינו בין 0 ל- 9, הקונסטרקטור זורק IllegalArgumentException. כיתבו מימוש של הקונסטרקטור.

```
public BigInt(int[] val) {

    // validate the argument
    if (val == null || val.length == 0) {
        throw new IllegalArgumentException
              ("Array can't be null or empty.");
    }
    // initialize the internal data structure
    digits = new int[val.length];

    // validate and copy the digits
    for (int i = 0; i < val.length; i++) {
        int digit = val[i];
        if (digit < 0 || digit > 9) {
            throw new IllegalArgumentException
                      ("Only array of digits is allowed");
        }
        // the digit is valid
        digits[i] = digit;
    }
}
```

3. (4 נקודות) המתודה toString מחזירה ייצוג טקסטואלי של המספר. כיתבו מימוש של המתודה.

```
public String toString() {
    String result = "";     // Could use StringBuilder instead
    for (int digit : digits) {
        result += digit;
    }
    return result;
}
```

4. (6 נקודות) הקונסטרקטור BigInt (int[] val, int n) בונה מספר בן n ספרות מהארגומנט val. אם n קטן מהאורך של val, הקונסטרקטור זורק IllegalArgumentException. אם n גדול מהאורך של val, הקונסטרקטור מרפד את המספר עם אפסים מובילים. למשל, התבוננו בקטע הקוד הבא:

```
int[] xData = {1, 7, 3};
BigInt x = new BigInt(xData, 5);
System.out.println(x);  // Prints "00173"
```

כיתבו מימוש של הקונסטרקטור. כדי לפשט את החיים, בקונסטרקטור הזה אין צורך לעשות שום בדיקות נוספות על הארגומנטים פרט לבדיקת הערך של n.

```
public BigInt(int[] val, int n) {
     if (n < val.length) {
         throw new IllegalArgumentException("n too small");
     }
     digits = new int[n];
     for (int i = 0; i < n; i++) {
         digits[i] = 0;
     }
     int i = val.length - 1;
     int j = n - 1;
     while (!(i < 0)) {
         digits[j--] = val[i--];
     }
}
```

5. (5 נקודות) המתודה add (BigInt other) מחזירה את הסכום של המספר הנוכחי ו- other. כיתבו במילים כיצד תממשו את המתודה הזאת. שימו לב: אין צורך לכתוב קוד, ואין צורך להיכנס לפרטים כמו אינדקסים. יחד עם זאת, צריך לכתוב תיאור עקרוני קצר וקולע של גישת חישוב הסכום – תיאור ממנו מתכנת יוכל לממש את המתודה.

תשובה: יש לנו שני מערכים של ספרות שכל אחד מהם מייצג מספר BigInt. נתחיל בכך שנדאג ששניהם יהיו באותו אורך – אפשר לעשות זאת ע"י ייצור מספר BigInt חדש מהקטן שבין שני המספרים בעזרת קריאה לקונסטרקטור BigInt(int[] val, int n). כעת, כשיש לנו שני מערכים באותו אורך, אפשר לרוץ על כל הספרות שלהם בלולאה ובכל צעד לחבר את שתי הספרות הנוכחיות פלוס השארית (carry) מחיבור שתי הספרות הקודמות. מתחילים עם carry = 0.

6. (8 נקודות) המתודה compareTo (BigInt other) מחזירה 1, -1, או 0 אם המספר הנוכחי גדול מ-, קטן מ-, או שווה לארגומנט other, בהתאמה. כיתבו מימוש של המתודה. אין צורך לבדוק את תקינות הארגומנט, ואפשר להניח שאין אפסים בתחילת המספר. כלומר, אין מספרים כמו 00173 – מספר כזה ייוצג ע"י 173.

```
public int compareTo(Object other) {
       BigInt x = (BigInt) other;
       // check length conditions
       if (digits.length !=  x.digits.length) {
            return ((digits.length > x.digits.length) ? 1 : -1);
       }
       // Both numbers have the same number of digits
       for (int i = 0; i < this.digits.length; i++) {
            if (digits[i] != x.digits[i]) {
                 return ((digits[i] > x.digits[i]) ? 1 : -1);
            }
       }
       // the numbers are equal
       return 0;
}
```

7. (3 נקודות) המתודה min (BigInt other) משווה את other למספר הנוכחי ומחזירה את הקטן או השווה ביניהם. כיתבו מימוש של המתודה.

```
public BigInt min (BigInt other) {
    return ((this.compareTo(other) < 0) ? this : other);
}
```

8. (14 נקודות) המתודה intValue () מחזירה את ערך ה- int של המספר הנוכחי. אם המספר הנוכחי גדול מדי, המתודה מחזירה 1-. כיתבו מימוש רקורסיבי של המתודה intValue. עצה: אפשר להגדיר את המתודה הרקורסיבית כמתודה פרטית שהמתודה intValue קוראת לה עם פרמטר (אחד או יותר – תלוי בעיצוב שתבחרו לממש) מסוים. הערה: אפשר לכתוב מימוש לא רקורסיבי של intValue() תמורת 7 נקודות במקום 14.

תשובה: הפתרון מבוסס על תבנית החישוב של ערך מספרי. למשל, ערך המספר 2735 הוא (((2) * 10 + 7) * 10 + 3) * 10 + 5 . את תבנית החישוב הזאת ניתן לבטא באופן רקורסיבי, כי התבנית חוזרת על עצמה. במהלך החישוב, צריך לבדוק שהערך המחושב לא גולש מעבר לערך המקסימלי שניתן להכניס במשתנה מסוג int.

איך בודקים מתי הערך המחושב נהייה גדול מדי? בגלל שיטת ה 2's complement, המספר החיובי הגדול ביותר שניתן לייצג הוא 0111…1. לכן, הגלישה קורה כשמקבלים מספר בינארי כמו 111…1 או כל מספר בינארי אחר שמתחיל ב 1. בגלל שיטת ה 2's-complement, כל מספר שמתחיל ב 1 הוא שלילי. לכן, הבדיקה צריכה להיות כזאת: אם הערך המחושב נהיה שלילי, החזר 1-.

יחד עם זאת, לצורכי המבחן, פתרונות אחרים כמו לבדוק אם הערך המחושב לא יותר גדול מ Integer.MAX_VALUE (או כל קבוע דומה אחר שתמציאו) התקבלו גם כן. הנה מימוש אחד שעושה את העבודה:

```
public int intValue() {
    return recursiveIntValue(0, 0);
}

private int recursiveIntValue(int sofar, int i) {

    // Check for integer overflow
    if (sofar < 0)
        return -1;   // throwing an exception is OK also.

    // Terminate the recursion
    if (i == digits.length)
        return sofar;

    // Tail-recursive call
    return recursiveIntValue(sofar * 10 + digits[i], i + 1);
}
```

בבדיקת המבחן לא ציפינו לקבל פתרון מושלם. כדי לזכות ברוב הנקודות בשאלה הזאת, לקטע הקוד צריכים להיות שלושה אלמנטים: ניסוח סביר של הצעד הרקורסיבי, ניסוח סביר של תנאי העצירה, וניסוח סביר של בדיקה הגלישה.

9. (14 נקודות) כיתבו קטע קוד שעושה שלושה דברים, אחד אחרי השני. (א) בונה מערך של n מספרים גדולים כרצוננו. (ב) מאתחל כל מספר במערך בעזרת הקונסטרקטור ()BigInt שמחזיר מספר BigInt רנדומי. (ג) מיין את המערך בעזרת אחד מאלגוריתמי המיון שנלמדו בקורס (או כל אלגוריתם מיון אחר שעולה בדעתכם). כדי להסיר ספקות, הקוד שלכם צריך לממש את האלגוריתם הזה. מה סיבוכיות זמן הריצה של האלגוריתם שמימשתם?

```
// Declares an array of n BigInts and populates it
BigInt[] a = new BigInt[n];
for (int i = 0; i < n; i++) {
    a[i] = new BigInt();
}
// Sorts the array using selection sort
BigInt temp;
for (int i = 0; i < n - 1; i++) {
    for (int j = i + 1; j < n; j++) {
        if ((a[i].compareTo(a[j]) > 0)) {
            // Switch
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
}
```

בסריקה הראשונה האלגוריתם עושה n-1 השוואות; בסריקה השנייה האלגוריתם עושה n-2 השוואות; וכן הלאה. סך הכל האלגוריתם עושה (n-1) + (n-2) + … + 1 = ½ n (n - 1) צעדים. לכן סיבוכיות זמן הריצה היא סדר גודל של n^2.

לא הגבלנו באיזה אלגוריתם להשתמש. יחד עם זאת, אלגוריתמים מסורבלים ללא צורך גרמו להורדת נקודות, אפילו אם הם נכונים.

10. (3 נקודות) המחלקה BigInt, במימוש הנוכחי שלה, מייצגת מספרים גדולים כ immutable objects (אובייקטים שלא ניתנים לשינוי). נכון / לא נכון? נמקו את תשובתכם במשפט קצר וקולע.

תשובה: לשאלה הזאת יש שתי תשובות סבירות. תשובה סבירה אחת היא נכון, כי השדה digits הוא פרטי, והמחלקה לא מספקת מתודות שאמורות לשנות אותו. תשובה שנייה (יותר מתוחכמת) היא כדלקמן: תלוי איך המתודה getDigits ממומשת. אם היא מחזירה reference למערך עצמו, אזי ניתן לשנות את המערך מחוץ למחלקה ולכן זה mutable. אם המתודה מחזירה clone, אזי אי אפשר לשנות את המערך ולכן זה immutable.

11. (3 נקודות) המחלקה BigInt, במימוש הנוכחי שלה, מייצגת את המספר הנוכחי באמצעות מערך. הציעו דרך סבירה אחרת לייצג את המספר. אין צורך לכתוב קוד, אבל צריך לתאר באופן קצר וקולע את המימוש המוצע.

תשובה: במקום מערך ניתן לייצג את הספרות בעזרת רשימה מקושרת, או StringBuilder. האמת, השאלה הזאת לא מוצלחת כי החלופה הסבירה ביותר היא מערך, וכל חלופה אחרת פחות טובה באופן משמעותי (כולל רשימה מקושרת או StringBuilder). לכן החלטנו שבמקרה הגרוע ביותר נוריד בשאלה הזאת רק נקודה אחת, ונעשה זאת רק במקרים בהם החלופה המוצעת ממש לא לעניין, כמו למשל מחסנית אוArrayList. אלה מבני נתונים יקרים שהפונקציונליות שלהם לא מתאימה למשימה שלנו.

12. (3 נקודות) נניח שביצעתם את שינוי המימוש שמתואר בשאלה 11. האם השינוי הזה צריך לעניין לקוחות שמשתמשים במחלקה BigInt? איך נקרא העיקרון הכללי בעיצוב מונחה-עצמים שמאפשר שינויים כאלה במימוש של מחלקות?

תשובה: מימוש השדה digits לא צריך לעניין מחלקות אחרות כי מדובר בשדה פרטי שמוסתר, או "מוכמס" ממחלקות אחרות. העקרון שמאפשר את גמישות המימוש הזאת נקרא "הכמסה", אנקפסולציה, או encapsulation.

נניח שאנו מעוניינים לייצג מספרים בינאריים גדולים כרצוננו, למשל – מספר בן כמה מאות או אלפים של אפסים ואחדים. לצורך זה נבנה מחלקה ששמה BigBinary. כדי לפשט את החיים, ובלי להתחשב בשיקולי יעילות של זיכרון, נעשה זאת ע"י ירושה מהמחלקה BigInt. עיינו במבנה המחלקה BigBinary שמתואר בדף עזר.

<u>ענו על אחת מתוך שאלות 13, 14:</u>

13. (6 נקודות) הקונסטרקטור BigBinary (int[] val) בונה עצם מסוג BigBinary מהארגומנט, שאמור להיות מערך של מספרים שכל אחד מהם הוא 0 או 1. אם הארגומנט הוא null, או מכיל מספר שאינו 0 או 1, הקונסטרקטור זורק IllegalArgumentException. כיתבו מימוש של הקונסטרקטור.

```
public BigBinary (int[] val) {
    super(val);
    for (int digit : val)
        if ((digit != 0) && (digit != 1)) {
            throw new IllegalArgumentException
                  ("The argument does not represent a binary number");
        }
}
```

14. (6 נקודות) המתודה and(BigBinary other) מחזירה מספר BigBinary שבכל סיבית (ביט) שלו נמצאת התוצאה של הפעולה הלוגית and על כל אחד מהסיביות הבודדות (כזכור, הפעולה הזאת מחזירה 1 אם שני הביטים הם 1 ומחזירה 0 אחרת). למשל, התבוננו בקטע הקוד הבא:

```
int[] xData = {1, 1, 0};
int[] yData = {0, 1, 1};
BigBinary x = new BigBinary(xData);
BigBinary y = new BigBinary(yData);
System.out.println(x.and(y));  // Prints 010.
```

כיתבו מימוש של המתודה and (BigBinary other). אין צורך לבצע שום בדיקה של הארגומנט.

```
public BigBinary and (BigBinary other) {
    int[] a = this.getDigits();
    int[] b = other.getDigits();
    int[] c = new int[a.length];
    for (int i = 0; i < a.length; i++)
        c[i] = (a[i] * b[i] == 1) ? 1 : 0;
    return new BigBinary(c);
}
```

<u>ענו על אחת מתוך שאלות 15, 16:</u>

15. (6 נקודות) המתודה toString של BigBinary מחזירה ייצוג טקסטואלי של המספר הבינארי. כיתבו מימוש של המתודה.

```
public String toString () {
    return super.toString();
}
```

מספר סטודנטים כתבו מימוש מלא של המתודה הזאת. המימוש הזה הוא לא לעניין משתי סיבות: (א) כתבנו אותו כבר בשאלה 3, ו- (ב) הוא מיותר, בגלל הירושה. לכן, מימוש מלא של המתודה הזאת גרם להורדת נקודות.

16. (6 נקודות) התבוננו בפקודה `System.out.println(x.and(y));` מתוך קטע הקוד שהוצג בשאלה 14. הסבירו בדיוק ובקיצור נמרץ מה מתרחש כאשר הפקודה הזאת מתבצעת.

תשובה: הקריאה למתודה x.and(y) מחזירה אובייקט חדש מסוג BigInt שהספרות שלו הן תוצאה של הפעלת and על האובייקטים x ו- y. הקריאה למתודה System.out.println על האובייקט הזה קוראת באופן חבוי למתודה toString שמחזירה מחרוזת שמהווה את הייצוג הטקסטואלי של האובייקט החדש. המחרוזת מודפסת ע"י המתודה System.out.println.

<u>ענו על 2 מתוך שאלות 17, 18, 19, 20:</u>

17. (4 נקודות) המחלקה BigInt מממשת את טיפוס הנתונים שהיא מייצגת בעזרת מערך מטיפוס []int. טענה: את המחלקה הזאת ניתן לממש באופן יותר חסכוני בעזרת מערך מטיפוס []char. נכון? לא נכון? הסבר ונמק באופן קצר וקולע.

נכון. כדי לייצג 9 ספרות שונות לא צריך 32 סיביות – מספיק 4 סיביות. טיפוס הנתונים char מספק 50% חסכון כי הוא צורך "רק" 16 סיביות.

18. (4 נקודות) אם באמת טיפוס הנתונים של המערך יוחלף מ []int ל []char, אזי הקוד של מחלקות שעושות שימוש במחלקת BigInt צריך להשתנות בהתאם. נכון? לא נכון? הסבר ונמק באופן קצר וקולע.

השינוי מתרחש בשדה פרטי שאינו חשוף לעולם ולכן לא תהיה לו השפעה על מחלקות אחרות.

19. (4 נקודות) למחלקת BigInt יש כמה קונסטרקטורים שיש להם אותו שם. היכולת לעשות זאת היא דוגמא לתכונה כללית של שפת Java שנקראת method overriding. נכון? לא נכון? הסבר ונמק באופן קצר וקולע.

לא נכון. התכונה נקראת method overloading, או העמסה.

20. (4 נקודות) המחלקה BigInt תתקמפל ללא טעות אם נוריד את implements Comparable. נכון? לא נכון? הסבר ונמק באופן קצר וקולע.

נכון, היא תתקמפל ללא טעות. זה שיש במחלקה מתודה ששמה זהה למתודה שנמצאת בממשק (interface) כלשהו לא אומר שחייבים לציין זאת בהגדרת המחלקה. מחויבות הקומפיילר לממשק היא הפוכה: אם הגדרת המחלקה כוללת implements Comparable, אזי חוסר מימוש של מתודה שהחתימה שלה מופיעה בממשק יגרום לטעות קומפילציה.

ענו על 2 מתוך שאלות 21, 22, 23:

21. (4 נקודות) מה מדפיס קטע הקוד הבא (הקף בעיגול את אחת מארבעת התשובות):

```
int[] xData = {1, 1, 0};
int[] yData = {1, 0, 1};
BigInt x = new BigBinary(xData);
BigInt y = new BigBinary(yData);
System.out.println(x.add(y));
```

(א) טעות קומפילציה

(ב) טעות run-time

(ג) הפלט 211 (*)

(ג) אף תשובה לא נכונה

22. (4 נקודות) מה מדפיס קטע הקוד הבא (הקף בעיגול את אחת מארבעת התשובות):

```
int[] xData = {1, 1, 0};
int[] yData = {1, 0, 1};
BigBinary x = new BigBinary(xData);
BigBinary y = new BigBinary(yData);
System.out.println((x.and(y)).add(y));
```

(א) טעות קומפילציה

(ב) טעות run-time

(ג) הפלט 201 (*)

(ג) אף תשובה לא נכונה

23. (4 נקודות) מה מדפיס קטע הקוד הבא (הקף בעיגול את אחת מארבעת התשובות):

```
int[] xData = {1, 1, 0};
Comparable x = new BigInt(xData);
System.out.println(x);
```

(א) טעות קומפילציה

(ב) טעות run-time

(ג) הפלט 110 (*)

(ג) אף תשובה לא נכונה

(סוף המבחן)

דף עזר: מחלקת BigInt

מחלקת BigInt מייצגת מספר שלם חיובי גדול כרצוננו.

```
// Represents a positive integer value as large as needed.
public class BigInt implements Comparable {

  private int[] digits  // an array representation of the BigInt's digits

   // Constructs a BigInt object from a given array.
   // If the array is null, or the array contains a number
   // which is not in the range 0...9, throws an
   // IllegalArgumentException.
   public BigInt (int[] val)

   // Constructs a BigInt object from a given string.
   // The length of the string is not restricted.
   // If the string contains one or more non-digit characters,
   // throws an IllegalArgumentException.
   public BigInt (String val)

   // Constructs a BigInt object from a given array. If n is less
   // than the size of the array, throws an IllegalArgumentException.
   // If n is larger than the size of the array, pads the number with
   // leading zeros as necessary.
   public BigInt (int[] val, int n)

   // Constructs a random BigInt. Both the number of digits
   // and the digits themselves are randomly chosen.
   public BigInt()

   // Returns the digits of this number.
   public int[] getDigits()

   // Returns a BigInt whose value is this BigInt + the other.
   public BigInt add (BigInt other)

   // Returns true if this BigInt equals the other, false otherwise.
   public boolean equals (BigInt other)

   // Returns 1, -1, or 0 if this BigInt is greater than, less than,
   // or equal to the other.
   public int compareTo (BigInt other)

   // Converts this BigInt to an int value (32 bit). If this BigInt
   // is too large to fit in the int range, returns -1.
   public int intValue ()

   // Returns the minimum of this BigInt and the other.
   public BigInt min (BigInt other)

   // Returns the textual representation of this BigInt.
   public String toString ()

}
```

דף עזר: מחלקת BigBinary

```
public class BigBinary extends BigInt {

    // Constructs a new BigBinary object from a given array.
    // If the array is null, or the array contains a number
    // which is not 0 or 1, throws an IllegalArgumentException.
    public BigBinary (int[] val)

    // Computes the bit-wise "and" function of this and other.
    // For example, if this is 110 and other is 011, returns
    // the number 010.
    public BigBinary and (BigBinary other)

    // Returns a textual description of this BigBinary number.
    public String toString ()
}
```

דף עזר: ממשק Comparable (כפי שנלקח מה Java API):

Interface Comparable<T>

Type Parameters:

`T` - the type of objects that this object may be compared to

Method Summary	
`int`	**`compareTo`**`(T o)` Compares this object with the specified object for order.

Method Detail

`int` **`compareTo`**`(T o)`

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Parameters:

`o` - the object to be compared.

Returns:

a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Throws:

`ClassCastException` - if the specified object's type prevents it from being compared to this object.