<u>מבחן מסכם מועד א' ב"מבוא למדעי המחשב" 202-1-101-1</u>

סמסטר א' תשס"ב
8.2.2002

פרופ' אורי אברהם
פרופ' דניאל ברנד
ד"ר שמואל ספרוני
פרופ' מיכאל קודיש
ד"ר חן קיסר

משך הבחינה שלוש שעות.
חומר עזר אסור.
אין להשתמש במחשבון.

במבחן זה 6 שאלות.

אנא רשמו את תשובותיכם בדף התשובות בלבד. המחברת שקיבלתם היא **מחברת טיוטה והיא לא תיסמר כלל לבדיקה.** בסיום הבחינה נשמור אך ורק את דף התשובות. כל שאר החומר יועבר לגריסה. הקפידו לרשום בדף התשובות גם את מספר הנבחן ומספר החדר שבו אתם נבחנים.

בשאלות התכנות, מספר השורות העומדות לרשותכם בדף התשובות רומז על אורך הקוד הנדרש. הקפידו על כתב יד ברור. תשובות מסורבלות או ארוכות מדי לא יזכו בניקוד מלא. **אין צורך להעתיק את שורות הקוד הנתונות בשאלון לדף התשובות.**

בהצלחה !

<u>שאלה 1</u> (30 נקודות)

בשאלה זו נניח שנתון ממשק עבור רשימות דומה לזה שתואר בכיתה:

```
public interface List{
        boolean isEmpty( ) ;            //  check if the list is empty
        boolean addFront(Object a);     //  adds object a at the front of the list
        boolean addEnd(Object a);       //  adds object a at the end of the list
        Object removeFront( );          //  removes and returns the first element of the list
}
```

נניח גם שהמחלקות הבאות הוגדרו (בדומה לאלו שתוארו בכיתה):

```
class LinkedList implements List{…}             /*  מממש את הממשק בעזרת שרשרת חוליות  */
class SortableList extends LinkedList {…}       /* מגדיר רשימה של עצמים שניתנים להשוואה */
class SortedList extends SortableList {…}       /*  מגדיר רשימה ממוינת של עצמים  */
```

כמו כן נתייחס לשלוש השיטות הבאות שתיארנו בכיתה:

```
public static SortedList mergesort(SortableList list){
        if (list.getFirstLink() == null || list.getFirstLink().get_next() == null)
        // if the list is empty or has only one element
                return new SortedList(list);
        else{   SortableList[] pair = split(list);
                return  merge(mergesort(pair[0]), mergesort(pair[1]));
        }
}
public static SortableList[] split (SortableList list){
        SortableList[] pair = {new SortableList( ), new SortableList( )};
        int i = 0;
        while ( !list.isEmpty() ){
                pair[i%2].addFront(list.removeFront());
                i = i+1;
        }
        return pair;
}
public static SortedList merge(SortedList list1, SortedList list2){
        SortedList list;
        Object head;
        if (list1.isEmpty()) return list2;
        else if (list2.isEmpty()) return list1;
        else {  Object data1 = list1.head(),  data2 = list2.head();
                if (((Comparable) data1).compareTo((Comparable) data2) < 0)
                        head = list1.removeFront();
                else
                        head = list2.removeFront();
                list = merge(list1,list2);
                list.addFront(head);
                return list;
        }
}
```

סטודנט בקורס מבוא למדעי המחשב הציע לממש מיון על-ידי מיזוג (merge sort) בעזרת התוכנית הבאה בשפת java, אשר מקבלת בפרמטר X רשימת חוליות של עצמים שניתנים להשוואה ומחזירה אותן כרשימה ממוינת.

```
public static SortedList mergesort2(SortableList X){
        return multimerge(multisplit(X));
}
```

הרעיון הוא שתחילה נפצל (בעזרת השיטה multisplit) את איברי הרשימה X הנתונה לרשימה של רשימות, כל אחת בעלת איבר יחיד ובתור שכזאת גם ממוינת; אחר כך נעבור על רשימת הרשימות הממוינות (בעזרת השיטה multimerge) ונמזג זוג רשימות מתוכה. כך נעשה עד שתישאר רק רשימה ממוינת אחת ברשימת הרשימות, ואותה נחזיר.

הסטודנט הציע את המימוש הבא בשפת java, כאשר הקריאה ל-merge בשורה המסומנת ב-N היא לתוכנית שתיארנו בכיתה, אשר ממזגת שתי רשימות ממוינות ומחזירה רשימה ממוינת.

```
public static List multisplit (SortableList X){          //A
        List ans = new LinkedList( );                    //B
        while (!X.isEmpty( )){                           //C
                SortedList single = new SortedList( ); //D
                single.addFront(X.removeFront( ));       //E
                ans.addFront(single);                    //F
        }
        return ans;                                      //G
}

public static SortedList multimerge (List list){         //H
         if (list.isEmpty( )) return (SortedList) list;  //I
        SortedList s1 =  (List) list.removeFront( );     //J
        if (list.isEmpty( )) return s1;                  //K
        else{                                            //L
                SortedList s2 =  (List) list.removeFront( ); //M
                list.addFront(merge(s1,s2));             //N
                return multimerge (list);                //O
        }
}
```

התלמיד הסביר שהצעתו – mergesort2 – יעילה מזו שלמדנו בכיתה מכיוון שהיא מבצעת רק קריאה אחת ל-multisplit, וזו כרוכה במעבר אחד בלבד על איברי הרשימה הנתונה. כזכור, התוכנית mergesort שתיארנו בכיתה מבצעת מספר קריאות ל-split ועלותן הכוללת כ- n*(log n) צעדי חישוב.

את הצעתו של התלמיד נבחן בסעיפים הבאים.

<u>סעיף א'</u> (10 נק')

הטענות הבאות מתייחסות לשיטה multisplit בלבד. שיטה זו אמורה לפצל רשימה X נתונה לרשימה של רשימות ממוינות (מסוג SortedList), כל אחת בעלת איבר יחיד. בסעיף זה יש להתעלם מהמחלקות האחרות. סמנו את כל התשובות הנכונות.

א. השיטה multisplit לא תצלח הידור (לא תעבור קומפילציה). שים לב שהמשתנה `ans` מוגשם כעצם מסוג `LinkedList`, אך השיטה צריכה להחזיר עצם מסוג `List` שהינו ממשק.

ב. השיטה multisplit תצלח הידור ותבצע את המשימה שתיאר הסטודנט בצורה נכונה.

ג. אם נרשום LinkedList במקום List בשתי השורות הראשונות (המסומנות ב-A ו-B) אז השיטה multisplit תצלח הידור ותבצע את המשימה שתיאר הסטודנט בצורה נכונה.

ד. אם נרשום SortableList במקום List בשתי השורות הראשונות (המסומנות ב-A ו-B) אז השיטה multisplit תצלח הידור ותבצע את המשימה שתיאר הסטודנט בצורה נכונה.

ה. השיטה multisplit תצלח הידור ותפצל את איברי הרשימה הנתונה X כנדרש, אך לא נוכל להשתמש בה על מנת למיין עצמים. זאת מכיוון שהרשימה המוחזרת אינה מסוג SortableList, וכתוצאה מכך יתכן שאיבריה אינם ניתנים להשוואה.

ו. אם נחליף את השורה המסומנת ב-D בשורה `List single = new SortedList( );` אז השיטה multisplit תצלח הידור ותבצע את המשימה שתיאר הסטודנט בצורה נכונה.

ז. אם נחליף את השורה המסומנת ב-D בשורה `List single = new LinkedList( );` אז השיטה multisplit תצלח הידור ותתבצע ללא שגיאות של זמן ריצה.

ח. אף אחת מהתשובות הקודמות אינה נכונה.

לנוחיותכם, מצ"ב שוב השיטה:

```
public static List multisplit (SortableList X){          //A
        List ans = new LinkedList( );                    //B
        while (!X.isEmpty( )){                           //C
                SortedList single = new SortedList( ); //D
                single.addFront(X.removeFront( ));       //E
                ans.addFront(single);                    //F
        }
        return ans;                                      //G
}
```

<u>סעיף ב'</u> (10 נק')

הטענות הבאות מתייחסות לשיטה multimerge. כאן הניחו כי שגיאות אפשריות בשיטה multisplit תוקנו, כך שהשיטה multisplit פועלת כמו שהתכוון התלמיד. כלומר, השיטה מפצלת את הרשימה X הנתונה לרשימה של רשימות ממוינות (מסוג SortedList), כל אחת בעלת איבר יחיד. סמנו את כל התשובות הנכונות.

א. השיטה multimerge תצלח הידור ואם הפרמטר שלה, list, פונה לרשימה שכל איבר בה הנו רשימה מסוג SortedList אזי לא יהיו שגיאות של זמן ריצה.

ב. אם נחליף את שתי פעולות ההשלכה (casting) בשורות המסומנות ב-J וב-M בפעולות ההשלכה למחלקה SortedList אז המשפט בסעיף א' הוא נכון.

ג. אם נשנה את טיפוס הפרמטר list של השיטה מ-List ל-SortedList ונשמיט את שתי פעולות ההשלכה (casting) בשורות המסומנות ב-J וב-M אז המשפט בסעיף א' הוא נכון.

ד. אם נחליף את שתי פעולות ההשלכה (casting) בשורות המסומנות ב-J וב-M בפעולות ההשלכה למחלקה SortedList אז השיטה multimerge תצלח הידור, ואם הפרמטר שלה list פונה לרשימה שיש בה לפחות איבר אחד, ושכל איבר בה הנו רשימה מסוג SortedList, אזי לא יהיו שגיאות של זמן ריצה.

ה. הרעיון הבסיסי בשיטה multimerge הוא נכון, ויכול להוות בסיס למימוש אלגוריתם של מיון. אך הבעיה היא לא רק בשגיאות תחביריות שאותן ניתן לתקן בקלות, כי אם מהותית יותר, ונובעת משימוש לא זהיר ברקורסיה. הקריאה הרקורסיבית (בשורה המסומנת ב-O) זהה לקריאה ההתחלתית לשיטה (בשורה המסומנת ב-H) וגורמת לכך שהביצוע לעולם לא יגיע לסיום.

ו. אף אחת מהתשובות הקודמות אינה נכונה.

לנוחיותכם, מצ"ב שוב השיטה:

```
public static SortedList multimerge (List list){          //H
        if (list.isEmpty( )) return (SortedList) list;    //I
        SortedList s1 =  (List) list.removeFront( );      //J
        if (list.isEmpty( )) return s1;                   //K
        else{                                             //L
                SortedList s2 =  (List) list.removeFront( ); //M
                list.addFront(merge(s1,s2));              //N
                return multimerge (list);                 //O
        }
}
```

<u>סעיף ג'</u> (10 נק')

הטענות הבאות מתייחסות לתוכנית mergesort2 כולה. כאן הניחו כי שגיאות אפשריות בשיטות multisplit ו-multimerge תוקנו, כך שהתוכנית כולה צולחת הידור וממיינת את הרשימה הנתונה.
סמנו את כל התשובות הנכונות.

א. כיון שהשיטה multimerge ממזגת רשימות (ברשימת הרשימות) עד שנשארת רק רשימה אחת, אותה מזהים כ-s1 בשורה המסומנת ב-J, והיא התוצאה המוחזרת בשורה המסומנת ב-K, אזי נוכל להשמיט את השורה המסומנת ב-I וגם אז mergesort2 ממיינת נכון כאשר הפרמטר שלה, list, רשימה שכל איבר בה הנו רשימה מסוג SortedList.

ב. בהינתן רשימה בעלת n איברים, הרי בכל אחת מהתוכניות – mergesort (שתוארה בכיתה) ו-mergesort2 (כפי שהציע התלמיד) – מתבצעות בדיוק n-1 קריאות לשיטה merge.

ג. התוכנית של התלמיד מממשת את האלגוריתם של מיון על-ידי מיזוג והיא יותר יעילה מהתוכנית שתיארנו בכיתה. זאת מכיוון ששתי התוכניות מבצעות אותו מספר קריאות לשיטה merge, אך עבור רשימה באורך n>0, התוכנית של התלמיד מבצעת רק קריאה אחת ל- multisplit, וזו כרוכה במעבר אחד בלבד על n איברי הרשימה. לעומת זאת, התוכנית שתיארנו בכיתה מבצעת מספר פעולות split הכרוכות ב- log n מעברים על n איברי הרשימה.

ד. התוכנית של התלמיד מממשת אלגוריתם של מיון, אך היא פחות יעילה מזו שתיארנו בכיתה ובעצם דומה יותר למיון על-ידי הכנסה.

ה. אם נחליף את השורה המסומנת ב-N בשורה list.addEnd(merge(s1,s2)); אז התוכנית תמשם את האלגוריתם של מיון על-ידי מיזוג. בהינתן מימוש של המחלקה LinkedList המספק גישה ישירה לאיבר האחרון במבנה (last), התוכנית של התלמיד יעילה לפחות כמו זו שתיארנו בכיתה.

ו. אף אחת מהתשובות הקודמות אינה נכונה.

לנוחיותכם, מצ"ב שוב השיטה:

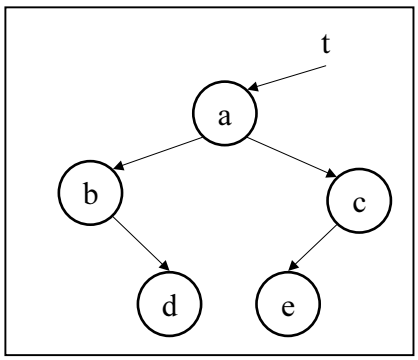
```
public static SortedList mergesort2(SortableList X){
        return multimerge(multisplit(X));
}
```

<u>שאלה 2</u> (25 נקודות)

נתונה התוכנית הבאה:

```
public static void mystery(BinaryNode t){
    if (t !=null){
        System.out.print ( t.get_data()+ " " );
        mystery( t.get_left() );
        System.out.print ( t.get_data()+ " " );
        mystery( t.get_right() );
        System.out.print ( t.get_data()+ " " );
    }
}
```

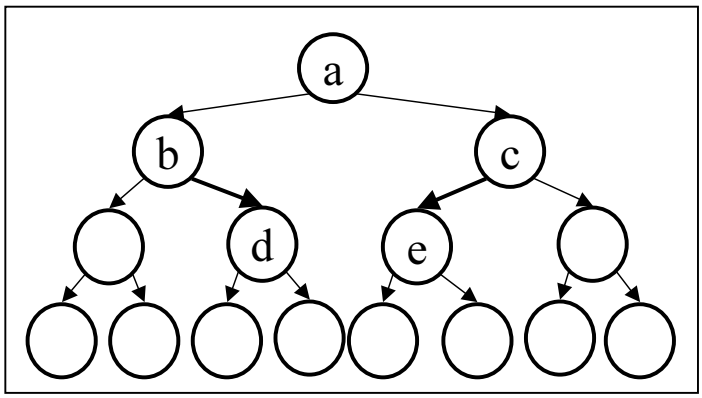
<u>סעיף א'</u> (5 נק') מה תדפיס התוכנית כאשר הפרמטר t פונה לקדקוד בינארי כמתואר בציור הבא?
(הכוונה היא ש- t.data = "a", t.left פונה לתת-העץ ששורשו מסומן ב-b, t.right פונה לתת-העץ ששורשו מסומן ב-c, וכדומה).



<u>סעיף ב'</u> (5 נק') ציירו עץ שאם t פונה לשורש שלו אז קריאה ל- mystery(t) תדפיס את השורה הבאה:

b a c c d d d c a e e f f f e a b b

בדף התשובות השתמשו בתבנית הנתונה. לדוגמא - את העץ מסעיף א' ניתן לרשום בתבנית כך:



<u>סעיף ג'</u> (15 נק')

ניתן להגדיר את מיקום הקדקודים בעץ בינארי על-ידי תיאור המסלול מהשורש לכל קודקוד, כאשר מסלול מאופיין כסדרת "פניות", *ימינה* או *שמאלה*. למשל, המסלול אל הקדקוד שמסומן באות "d" בעץ מסעיף א' הוא "*שמאל, ימין*"; המסלול אל הקדקוד שמסומן באות "c" הוא: "*ימין*". המסלול אל השורש הוא הסדרה הריקה.

השיטה paths שאותה תוסיפו במחלקה BinaryNode, תאפשר לקדקוד להדפיס תאור של כל הקדקודים בעץ המושרש בו (כל הקדקודים שמתחתיו). מסלול יתואר על-ידי מחרוזת של האותיות L (שמאל) ו-R (ימין). על השיטה להדפיס את המסלולים, אחד לשורה, כך שאם קדקוד x מופיע בדרך מהשורש לקדקוד אחר y, אז המסלול של x יודפס לפני המסלול של y. למשל, הקדקוד ש- t פונה אליו בציור של סעיף א' יוכל להדפיס את המחרוזות הבאות:

“”
“L”
“R”
“LR”
“RL”

רמז: מומלץ לתת הגדרה רקורסיבית שמשלימה את המבנה הבא:

```
public  void  paths(String st){

        השלם

}
```

כאשר הפרמטר st מציין את המסלול עד לקדקוד שמפעיל את השיטה (בהתחלה st = ""). השיטה הבאה במחלקה BinaryTree תדפיס תאור של כל קדקודי העץ:

```
public void paths( ){
    if ( !isEmpty( ))
        root.paths(“”);
}
```

שאלה 3 (15 נקודות)

ביטוי סוגריים הוא מחרוזת המורכבת ממספר כלשהו של מופעים של כל אחד מן התווים הבאים:
סוגר עגול שמאלי - "(", סוגר עגול ימני - ")", סוגר מרובע שמאלי - "[", סוגר מרובע ימני - "]"
ורווח - " ". בין כל שני תווים שאינם רווח מופיע לפחות רווח אחד.

נאמר שביטוי סוגריים הוא **תקין** אם הוא מאחת הצורות הבאות:

- מחרוזת ריקה או מחרוזת המכילה רווח בלבד;
- $st_1 + " " + st_2$, כאשר st_1 ו-st_2 ביטויי סוגריים תקינים (לדוגמה "( ) [ ]");
- "(" + st + ")" או "[" + st + "]" כאשר st ביטוי סוגריים תקין.

כתבו שיטה סטטית boolean evaluate(String str) אשר מקבלת ביטוי סוגריים (אין צורך לבדוק), ומחזירה ערך אמת (true) אם הביטוי תקין או שקר (false) אחרת.

דוגמאות:

evaluate(" ") = = true	evaluate(" ( ) ") = = true
evaluate("( ) [ ] ") = = true	evaluate("(( ) [ ]) ") = = true
evaluate("( ] [ ] ") = = false	evaluate(") (") = = false
evaluate(" ((") = = false	evaluate("))") = = false

השיטה evaluate תשתמש לבצוע המשימה במחסנית, בממשק StackOperator:

```
public interface StackOperator{
        boolean operate(Stack s);
}
```

ובמחלקות הבאות המממשות את הממשק StackOperator:

- LeftRoundOperator - מייצגת סוגר שמאלי עגול.
- LeftSquareOperator - מייצגת סוגר שמאלי מרובע.
- RightRoundOperator - מייצגת סוגר ימני עגול.
- RightSquareOperator - מייצגת סוגר ימני מרובע.

המחלקה BracketTokenizer מרחיבה את המחלקה StringTokenizer ומחזירה בעזרת השיטה nextStackOperator() עצמים המיצגים את הסוגרים השונים. לנוחיותכם, תזכורת לגבי המחלקה StringTokenizer מצורפת כנספח למבחן.

השלימו את החסר בשיטות הבאות.

```
public static boolean evaluate(String term) {
        BracketTokenizer operators =
                new BracketTokenizer (term);
        Stack stk = new SomeImplementationOfStack();
        while (operators.hasMoreStackOperators())
                if (! operators.nextStackOperator().operate(stk)) return false;
        return stk.isEmpty();
}

public class BracketTokenizer extends StringTokenizer {
        public BracketTokenizer (String expression) { super(expression);}
        public boolean hasMoreStackOperators() {return hasMoreTokens();}
        public StackOperator nextStackOperator() {
                String token = nextToken();
                if (token.equals("("))
                        return new   LeftRoundOperator();
                 if (token.equals(")"))
                        {       השלם א' (1 נק')     }
                 if (token.equals("["))
                        return new   LeftSquareOperator();
                {       השלם ב' (1 נק')       }
        }
}

public class LeftRoundOperator implements StackOperator
        {       השלם ג' (6 נק')         }

public class LeftSquareOperator implements StackOperator
        {       אין צורך להשלים        }


public class RightRoundOperator implements StackOperator
        {       השלם ד' (7 נק')         }

public class RightSquareOperator implements StackOperator
        {       אין צורך להשלים        }
```

שאלה 4 (10 נקודות)

נתון ממשק עבור קבוצות

```
public interface Set{
        public void add(Object o);
        public boolean contains( Object o);
        public Iterator iterator( );
}
```

ונתונה גם המחלקה

```
public class Relation implements Set{ … }
```

אשר מממשת יחס בינארי כקבוצה של זוגות סדורים.

זוג סדור מוגדר על-ידי המחלקה

```
public class Pair{
        private Object first, second;
        public Pair( Object f, Object s) {
                first = f; second = s;
        }
        public Object getFirst( ) { return first;}
        public Object getSecond( ) { return second; }
        public boolean equals( Object other) { /*    מוגדר ועובד נכון   */ }
}
```

עליכם להשלים במחלקה Relation את הגדרת השיטה isAntiSymmetric אשר בודקת האם היחס הממומש הוא אנטי-סימטרי ומחזירה ערך בוליאני בהתאם.

```
public boolean isAntiSymmetric( ){
        boolean answer = true;
        Iterator it = iterator( );
        Pair p;

        /*   השלם   */

        return answer;
}
```

להזכירכם, יחס R על קבוצה X הוא אנטי-סימטרי אם לכל a,b ב-X, אם $(a,b)\in R$ אז $(b,a)\notin R$.
תוכלו להניח שהמימוש של המחלקה Relation דואג לכך שכל האיברים בו הם זוגות סדורים.

אין צורך להעתיק את שורות הקוד הנתונות לדף התשובות.

שאלה 5 (10 נקודות)

המספרים n, . . . ,1 נשמרים (כעצמים) בסדר כלשהו בתור. נגשים עץ חיפוש בינארי בעזרת הבונה הבא אותו נוסיף למחלקה BST (BinarySearchTree):

```
BST(Queue q){
   super( );
   while (!q.isEmpty( ))
        this.insert(q.dequeue( ));
}
```

להזכירכם, הבונה במחלקת האם (BinaryTree) מגשים עץ בינארי ריק והשיטה insert במתלקה BST מוסיפה קדקוד חדש בעץ חיפוש בינארי במקום המתאים.

סמנו את כל התשובות הנכונות:

א. כל אחד מ- n! הסדרים השונים של האיברים בתור ייצור עץ בינארי שונה (בצורתו). בפרט, מספר העצים הבינאריים (השונים בצורתם) בעלי n קדקודים הוא לפחות n!.

ב. לכל עץ בינארי בעל n קדקודים קיים סידור התחלתי של המספרים בתור, כך שהבונה הנ"ל יגשים עץ חיפוש בעל אותה צורה. בפרט, מספר העצים הבינאריים (השונים בצורתם) בעלי n קדקודים הוא לכל היותר n!.

ג. לכל עץ בינארי בגובה n-1 ובעל n קדקודים קיים סידור יחיד של המספרים בתור שייצור עץ זהה לו בצורתו.

ד. אם נחליף את שני האיברים האחרונים בתור, עץ החיפוש הבינארי המתקבל לא ישתנה.

ה. אם f היא פונקציה מונוטונית עולה ואם לפני ההעברה מהתור לעץ החיפוש הבינארי נחליף כל איבר i ב-f(i) , צורת העץ לא תשתנה.

ו. קיימים מספרים n וסידורים התחלתיים של המספרים בתור אשר עבורם מעבר בשיטת inorder על קדקודי העץ שיוגשם נותן את האיברים בסדרם המקורי בתור. קיימים מספרים n וסידורים התחלתיים של המספרים בתור אשר עבורם מעבר בשיטת inorder על קדקודי העץ שיוגשם נותן את האיברים שלא בסדרם המקורי בתור.

ז. אף אחת מהתשובות הנ"ל אינה נכונה.

שאלה 6 (10 נקודות)

התבוננו בשיטות הבאות:

```
static void evenSubsets(int n) {
     evenSubsets(n," ");
}
static void evenSubsets(int n,String suffix) {
      if (n == 0 && ((suffix.length() %2) == 0)) // LineA
             System.out.println(suffix);
      else {
          oddSubsets(n-1," " + n + suffix); // Line B
          evenSubsets(n-1,suffix);
     }
}
static void oddSubsets(int n) {
     oddSubsets(n," ");
}
static void oddSubsets(int n,String suffix) {
      if (n >= 1)  {
            evenSubsets(n-1," " + n + suffix);
            oddSubsets(n-1,suffix);
        }
}
```

השיטות evenSubsets(int n) ו- oddSubsets(int n) נבנו במטרה שידפיסו את כל תת-הקבוצות בגדלים זוגיים ואי-זוגיים בהתאמה של הקבוצה {1,...,n}.
לדוגמה, הקריאה oddSubsets(3) צריכה להדפיס את תת-הקבוצות (אם כי לא בהכרח לפי סדר זה):

```
1
2
3
1 2 3
```

סמן את כל התשובות הנכונות.

א. שתי השיטות תעבודנה היטב לכל ערך של n. כמובן, עבור ערכים גדולים, מגבלות המחשב יהפכו את השיטה ללא ישימה.

ב. שתי השיטות תעבודנה היטב לכל ערך של n עד 9 ועד בכלל.

ג. אם נחליף את השורה המסומנת Line A בשורה

```
if(n= =0)  // Line A
```

אז שתי השיטות תעבודנה היטב לכל ערך של n. כמובן, עבור ערכים גדולים, מגבלות המחשב יהפכו את השיטה ללא ישימה.

ד. השיטה תעבוד היטב לכל מספר n בעל מספר ספרות אי-זוגי. כמובן, עבור ערכים גדולים מגבלות המחשב יהפכו את השיטה ללא ישימה.

ה. כידוע a + b תמיד זהה ל b + a. לכן החלפת השורה המסומנת Line B בשורה

```
oddSubsets(n-1 , n  +" "  +  suffix); // Line B
```

לא תשפיע בשום צורה על הפלט.

ו. אף אחת מהתשובות א-ה אינה נכונה.