

מדעי המחשב

פרק ראשון

שאלה 1

Java

```

--- פעלה המחזירה אמת אם המחרוזת המתקבלת היא מחרוזת כפולה ושקר אחרת ---
--- מחרוזת כפולה היא מחרוזת לא ריקה שהחצי הראשון שלה זהה לחצי השני ---
public static boolean isDouble(String str)
{
    int k = str.length();
    if (k == 0)
        return false;
    String st1 = str.substring(0, k/2);
    String st2 = str.substring(k/2);
    return st1.equals(st2);
}

```

C#

```

--- פעלה המחזירה אמת אם המחרוזת המתקבלת היא מחרוזת כפולה ושקר אחרת ---
--- מחרוזת כפולה היא מחרוזת לא ריקה שהחצי הראשון שלה זהה לחצי השני ---
public static bool IsDouble(string str)
{
    int k = str.Length;
    if (k == 0)
        return false;
    string st1 = str.Substring(0, k / 2);
    string st2 = str.Substring(k / 2);
    return st1.Equals(st2);
}

```

Java

```

//--- פעולה המחזירה מערך של זוגות מספרים אקראיים ---
public static PairOfNums[] generate(int n)
{
    PairOfNums[] arr = new PairOfNums[n];

    for (int i = 0 ; i < arr.length ; i++)
        arr[i] = getPairOfNum();

    return arr;
}

static Random rnd = new Random();

//--- פעולה המחזירה עצם מסוג התחלה-סוף תקין ---
public static PairOfNums getPairOfNum()
{
    int n1, n2;
    PairOfNums pon = new PairOfNums(-1, -1);

    do{
        n1 = rnd.nextInt( bound: 1000)+1;
        n2 = rnd.nextInt( bound: 1000)+1;
        pon = new PairOfNums(n1, n2);
    }while (!pon.endStart());

    return pon;
}

```

C#

```

//--- פעולה המחזירה מערך של זוגות מספרים אקראיים ---
public static PairOfNums[] Generate(int n)
{
    PairOfNums[] arr = new PairOfNums[n];

    for (int i = 0; i < arr.Length; i++)
        arr[i] = GetPairOfNum();

    return arr;
}

static Random rnd = new Random();

//--- פעולה המחזירה עצם מסוג התחלה-סוף תקין ---
public static PairOfNums GetPairOfNum()
{
    int n1, n2;
    PairOfNums pon = new PairOfNums(-1, -1);

    do
    {
        n1 = rnd.Next(1,1001);
        n2 = rnd.Next(1,1001);
        pon = new PairOfNums(n1, n2);
    } while (!pon.EndStart());

    return pon;
}

```

Java

```

//--- פעולה המחזירה אמת אם המטריצה היא מערך פינה ושקר אחרץ ---
//--- מערך פינה הוא מטריצה שבו בכל שורה ובכל עמודה תואמת ---
//--- קיים מספר השווה למס' השורה/עמודה + 1 ---
public static boolean isCorner (int[][]matrix)
{
    int col = matrix[0].length;
    int row = matrix.length;
    if (col != row)
        return false;

    for (int i = 0 ; i < row ; i++)
        if (!chkRow(matrix, i, num: i+1) || !chkCol(matrix, i, num: i+1))
            return false;

    return true;
}

//--- פעולה המקבלת את המטריצה, מספר auxy ומספר שלם ---
//--- ומחזירה אמת אם כל הערכים בשורה שווים למספר ושקר אחרת ---
public static boolean chkRow(int[][]matrix, int r, int num)
{
    int lastCol = matrix[r].length;

    for (int i = r ; i < lastCol ; i++)
        if (matrix[r][i] != num)
            return false;

    return true;
}

//--- פעולה המקבלת את המטריצה, מספר העמודה ומספר שלם ---
//--- ומחזירה אמת אם כל הערכים בעמודה שווים למספר ושקר אחרת ---
public static boolean chkCol(int[][]matrix, int c, int num)
{
    int lastRow = matrix.length;

    for (int i = c ; i < lastRow ; i++)
        if (matrix[i][c] != num)
            return false;

    return true;
}

```

C#

```

//--- פעולה המחזירה אמת אם המטריצה היא מערך פינה ושקר אחרת ---
//--- מערך פינה הוא מטריצה שבו בכל שורה ובכל עמודה תואמת ---
//--- קיים מספר השווה למס' השורה/עמודה + 1 ---
public static bool IsCorner(int[,] matrix)
{
    int col = matrix.GetLength(1);
    int row = matrix.GetLength(0);
    if (col != row)
        return false;

    for (int i = 0; i < row; i++)
        if (!ChkRow(matrix, i, i + 1) || !ChkCol(matrix, i, i + 1))
            return false;

    return true;
}

//--- ומספר שלם aurv פעולה המקבלת את המטריצה, מספר ---
//--- ומחזירה אמת אם כל הערכים בשורה שווים למספר ושקר אחרת ---
public static bool ChkRow(int[,] matrix, int r, int num)
{
    int lastCol = matrix.GetLength(1);

    for (int i = r; i < lastCol; i++)
        if (matrix[r,i] != num)
            return false;

    return true;
}

//--- פעולה המקבלת את המטריצה, מספר העמודה ומספר שלם ---
//--- ומחזירה אמת אם כל הערכים בעמודה שווים למספר ושקר אחרת ---
public static bool ChkCol(int[,] matrix, int c, int num)
{
    int lastRow = matrix.GetLength(0);

    for (int i = c; i < lastRow; i++)
        if (matrix[i,c] != num)
            return false;

    return true;
}

```

פרק שני

עאלה 4

Java

```
//--- פעולה המחזירה אמת אם הרשימה היא שרשרת מאורגנת ושקר אחרת ---
//--- שרשרת מאורגנת היא רשימה של מספרים שלמים באורך זוגי ---
//--- שכל המספרים בחצי הראשון קטנים מכל המספרים בחצי השני ---
//---
//--- כדי לבדוק את התכונה השנייה מספיק לבדוק שהערך הגדול ביותר ---
//--- בחצי הראשון קטן מהערך הקטן ביותר בחצי השני ---
public static boolean isArranged (Node<Integer>lst)
{
    int k = size(lst);
    if (k % 2 != 0)
        return false;
    int max = maxInSubList(lst, size: k/2);

    Node<Integer> pos = getPosition(lst, size: k/2);
    int min = minInSubList(pos);

    return min >= max;
}
```

```
//--- הפעולה מחזירה את מספר האיברים ברשימה ---
public static int size(Node<Integer>lst)
{
    int count=0;

    while(lst != null)
    {
        count++;
        lst = lst.getNext();
    }

    return count;
}
```

```
//--- פעולה המחזירה הפנייה לחוליה במקום הנתון ברשימה ---
public static Node<Integer> getPosition (Node<Integer> lst, int size)
{
    while (lst != null && size > 0)
    {
        lst = lst.getNext();
        size --;
    }
    return lst;
}
```

```
//--- פעולה המחזירה את המספר הגדול ביותר ברשימה באורך הנתון ---
public static int maxInSubList(Node<Integer> lst, int size)
{
    int max = lst.getValue();
    lst = lst.getNext();
    size --;

    while (lst != null && size > 0)
    {
        if (lst.getValue() > max)
            max = lst.getValue();
        lst = lst.getNext();
        size --;
    }
    return max;
}
```

```
//--- פעולה המחזירה את המספר הקטן ביותר ברשימה החל מהמקום הנתון ---
public static int minInSubList(Node<Integer>pos)
{
    int min = pos.getValue();
    pos = pos.getNext();

    while (pos != null)
    {
        if (pos.getValue() < min)
            min = pos.getValue();
        pos = pos.getNext();
    }
    return min;
}
```

יעילות הפעולה: $O(n)$
 n מייצג את מספר האיברים ברשימה
 הפעולות:
 אורך רשימה - $size$
 מקום ברשימה - $getPosition$
 מקסימום ברשימה - $maxInSubList$
 מינימום ברשימה - $minInSubList$
 עוברות כל אחת פעם אחת על הרשימה ולכן יעילותן $O(n)$
 הפעולה $isArranged$ מפעילה את כל אחת מהפעולות פעם אחת בלבד, ולכן:
 $f(n) = 4n$
 וסיבוכיות הפעולה: $O(n)$

C#

```
//--- פעולה המחזירה אמת אם הרשימה היא שרשרת מאורגנת ושקר אחרת ---
//--- שרשרת מאורגנת היא רשימה של מספרים שלמים באורך זוגי ---
//--- שכל המספרים בחצי הראשון קטנים מכל המספרים בחצי השני ---
//---
//--- כדי לבדוק את התכונה השנייה מספיק לבדוק שהערך הגדול ביותר ---
//--- בחצי הראשון קטן מהערך הקטן ביותר בחצי השני ---
public static bool IsArranged(Node<int> lst)
{
    int k = Size(lst);
    if (k % 2 != 0)
        return false;
    int max = MaxInSubList(lst, k / 2);

    Node<int> pos = GetPosition(lst, k / 2);
    int min = MinInSubList(pos);

    return min >= max;
}

//--- הפעולה מחזירה את מספר האיברים ברשימה ---
public static int Size(Node<int> lst)
{
    int count = 0;

    while (lst != null)
    {
        count++;
        lst = lst.GetNext();
    }

    return count;
}

//--- פעולה המחזירה הפנייה לחוליה במקום הנתון ברשימה ---
public static Node<int> GetPosition(Node<int> lst, int size)
{
    while (lst != null && size > 0)
    {
        lst = lst.GetNext();
        size--;
    }
    return lst;
}

//--- פעולה המחזירה את המספר הגדול ביותר ברשימה באורך הנתון ---
public static int MaxInSubList(Node<int> lst, int size)
{
    int max = lst.GetValue();
    lst = lst.GetNext();
    size--;

    while (lst != null && size > 0)
    {
        if (lst.GetValue() > max)
            max = lst.GetValue();
        lst = lst.GetNext();
        size--;
    }
    return max;
}

//--- פעולה המחזירה את המספר הקטן ביותר ברשימה החל מהמקום הנתון ---
public static int MinInSubList(Node<int> pos)
{
    int min = pos.GetValue();
    pos = pos.GetNext();

    while (pos != null)
    {
        if (pos.GetValue() < min)
            min = pos.GetValue();
        pos = pos.GetNext();
    }
    return min;
}
```

יעילות הפעולה: $O(n)$
 n מייצג את מספר האיברים ברשימה
 הפעולות:
 אורך רשימה - Size
 מקום ברשימה - GetPosition
 מקסימום ברשימה - MaxInSubList
 מינימום ברשימה - MinInSubList
 עוברות כל אחת פעם אחת על הרשימה ולכן יעילותן $O(n)$
 הפעולה IsArrange מפעילה את כל אחת מהפעולות פעם
 אחת בלבד, ולכן: $f(n) = 4n$
 וסיבוכיות הפעולה: $O(n)$

פתרון אחר (פחות יעיל) *isArranged* 4

Java

```

//--- פעולה המחזירה אמת אם הרשימה היא שרשרת מאורגנת ושקר אחרת ---
//--- שרשרת מאורגנת היא רשימה של מספרים שלמים באורך דוגי ---
//--- שכל המספרים בחצי הראשון קטנים מכל המספרים בחצי השני ---
public static boolean isArranged(Node<Integer> lst)
{
    int k = Size(lst);
    if (k % 2 != 0)
        return false;

    k = k / 2;

    //--- העתקת החצי השמאלי של הרשימה המקורית לרשימה חדשה ---
    Node<Integer> lst1 = new Node<Integer> (lst.getValue());
    Node<Integer> pos1 = lst1;
    Node<Integer> pos = lst.getNext();
    for (int i = 1; i < k; i++)
    {
        pos1.setNext(new Node<Integer>(pos.getValue()));
        pos1 = pos1.getNext();
        pos = pos.getNext();
    }

    //--- העתקת החצי הימני של הרשימה המקורית לרשימה חדשה ---
    Node<Integer> lst2 = new Node<Integer>(pos.getValue());
    Node<Integer> pos2 = lst2;
    pos = pos.getNext();
    while(pos != null)
    {
        pos2.setNext(new Node<Integer>(pos.getValue()));
        pos2 = pos2.getNext();
        pos = pos.getNext();
    }

    //--- האם כל איברי תת-רשימה 1 קטנים מכל איברי תת-רשימה 2 ---
    pos1 = lst1;
    while (pos1 != null)
    {
        pos2 = lst2;
        while (pos2 != null)
        {
            if (pos1.getValue() >= pos2.getValue())
                return false;
            pos2 = pos2.getNext();
        }
        pos1 = pos1.getNext();
    }

    return true;
}

```

```

//--- הפעולה מחזירה את מספר האיברים ברשימה ---
public static int Size(Node<Integer> lst)
{
    int count = 0;

    while (lst != null)
    {
        count++;
        lst = lst.getNext();
    }

    return count;
}

```

יעילות הפעולה: $O(n^2)$
 n מייצג את מספר האיברים ברשימה
 סיבוכיות הפעולה אורך רשימה - size הוא $O(n)$
 הפעולה `isArranged` מחשבת:
 העתקת חצי הרשימה הימני והשמאלי לרשימות חדשות,
 כל לולאה עוברת על מחצית הרשימה, ולכן בסה"כ $O(n)$
 לאחר מכן משווה כל אחד מאיברי תת-רשימה-1 ($\frac{1}{2}n$)
 איברים) מול כל אחד מאיברי תת-רשימה-2 ($\frac{1}{2}n$ איברים),
 סה"כ $\frac{1}{2}n^2$ צעדים

$$f(n) = \frac{1}{2}n^2 + 2n \Rightarrow O(n^2)$$

C#

```

//--- פעולה המחזירה אמת אם הרשימה היא שרשרת מאורגנת ושקר אחרת ---
//--- שרשרת מאורגנת היא רשימה של מספרים שלמים באורך זוגי ---
//--- שכל המספרים בחצי הראשון קטנים מכל המספרים בחצי השני ---
public static bool IsArranged(Node<int> lst)
{
    int k = Size(lst);
    if (k % 2 != 0)
        return false;

    k = k / 2;

    //--- העתקת החצי השמאלי של הרשימה המקורית לרשימה חדשה ---
    Node<int> lst1 = new Node<int> (lst.GetValue());
    Node<int> pos1 = lst1;
    Node<int> pos = lst.GetNext();
    for (int i = 1; i < k; i++)
    {
        pos1.SetNext(new Node<int>(pos.GetValue()));
        pos1 = pos1.GetNext();
        pos = pos.GetNext();
    }

    // --- העתקת החצי הימני של הרשימה המקורית לרשימה חדשה ---
    Node<int> lst2 = new Node<int>(pos.GetValue());
    Node<int> pos2 = lst2;
    pos = pos.GetNext();
    while(pos != null)
    {
        pos2.SetNext(new Node<int>(pos.GetValue()));
        pos2 = pos2.GetNext();
        pos = pos.GetNext();
    }

    //--- האם כל איברי תת-רשימה 1 קטנים מכל איברי תת-רשימה 2 ---
    pos1 = lst1;
    while (pos1 != null)
    {
        pos2 = lst2;
        while (pos2 != null)
        {
            if (pos1.GetValue() >= pos2.GetValue())
                return false;
            pos2 = pos2.GetNext();
        }
        pos1 = pos1.GetNext();
    }

    return true;
}

//--- הפעולה מחזירה את מספר האיברים ברשימה ---
public static int Size(Node<int> lst)
{
    int count = 0;

    while (lst != null)
    {
        count++;
        lst = lst.GetNext();
    }

    return count;
}
    
```

יעילות הפעולה: $O(n^2)$
 n מייצג את מספר האיברים ברשימה
 סיבוכיות הפעולה אורך רשימה - Size הוא $O(n)$
 הפעולה IsArrange מחשבת:
 העתקת חצי הרשימה הימני והשמאלי לרשימות חדשות,
 כל לולאה עוברת על מחצית הרשימה, ולכן בסה"כ $O(n)$
 לאחר מכן משווה כל אחד מאיברי תת-רשימה-1 ($\frac{1}{2}n$)
 איברים) מול כל אחד מאיברי תת-רשימה-2 ($\frac{1}{2}n$ איברים),
 סה"כ $\frac{1}{2}n^2$ צעדים

$$f(n) = \frac{1}{2}n^2 + 2n \Rightarrow O(n^2)$$

שאלה 5

Java

```
//---
//--- פעולה המחזירה אמת אם הרשימה הראשונה מהווה את תחילתה של הרשימה השנייה ושקר אחרת
//---
public static boolean isPrefix(Node<Integer> lst1, Node<Integer> lst2)
{
    Node<Integer>pos1 = lst1;
    Node<Integer>pos2 = lst2;
    while (pos1 != null && pos2 != null)
    {
        if (pos1.getValue() != pos2.getValue())
            return false;
        pos1 = pos1.getNext();
        pos2 = pos2.getNext();
    }

    return pos1 == null;
}

---
---
//--- פעולה המחזירה אמת אם הרשימה הראשונה מוכלת בתוך הרשימה השנייה ושקר אחרת
//---
public static boolean isSubChain(Node<Integer> lst1, Node<Integer> lst2)
{
    Node<Integer>pos1 = lst1;
    Node<Integer>pos2 = lst2;
    while (pos1 != null && pos2 != null)
    {
        if (pos1.getValue() == pos2.getValue())
        {
            pos1 = pos1.getNext();
            pos2 = pos2.getNext();
        }
        else
        {
            pos1 = lst1;
            if (pos2.getValue() != pos1.getValue())
                pos2 = pos2.getNext();
        }
    }

    return pos1 == null;
}
}
```

C#

```
//--- סעיף א ---
//--- פעולה הממזירה אמת אם הרשימה הראשונה מהווה את תחילתה של הרשימה השנייה ושקר אחרת ---
public static bool IsPrefix(Node<int> lst1, Node<int> lst2)
{
    Node<int> pos1 = lst1;
    Node<int> pos2 = lst2;
    while (pos1 != null && pos2 != null)
    {
        if (pos1.GetValue() != pos2.GetValue())
            return false;
        pos1 = pos1.GetNext();
        pos2 = pos2.GetNext();
    }

    return pos1 == null;
}

// --- סעיף ב ---
//--- פעולה הממזירה אמת אם הרשימה הראשונה מוכלת בתוך הרשימה השנייה ושקר אחרת ---
public static bool IsSubChain(Node<int> lst1, Node<int> lst2)
{
    Node<int> pos1 = lst1;
    Node<int> pos2 = lst2;
    while (pos1 != null && pos2 != null)
    {
        if (pos1.GetValue() == pos2.GetValue())
        {
            pos1 = pos1.GetNext();
            pos2 = pos2.GetNext();
        }
        else
        {
            pos1 = lst1;
            if (pos2.GetValue() != pos1.GetValue())
                pos2 = pos2.GetNext();
        }
    }

    return pos1 == null;
}
```

Java

```

//--- פעולה המקבלת תור של נבדקים ומחזירה את קוד העיר שבה הכי הרבה חולים ---
public static int mostSick(Queue<CovidTest>q)
{
    int max = 0;
    int cityCode = 0;

    int num = 0;
    while (!q.isEmpty())
    {
        int code = q.head().getCityCode();
        num = numOfSickInCity(q, code);
        if (num > max)
        {
            max = num;
            cityCode = code;
        }
    }
    return cityCode;
}

//--- פעולה המקבלת תור של חולים וקוד של עיר ומחזירה את מספר החולים מעיר זו בתור ---
//--- בסיום הפעולה לא יהיו יותר נבדקים מעיר זו בתור ---
public static int numOfSickInCity (Queue<CovidTest> q, int code)
{
    Queue<CovidTest> qTemp = new Queue<>();
    int count = 0;
    CovidTest ct;

    while (! q.isEmpty())
    {
        ct = q.remove();
        if (ct.getCityCode() == code)
            count ++;
        else
            qTemp.insert(ct);
    }

    while (!qTemp.isEmpty())
        q.insert(qTemp.remove());

    return count;
}

```

C#

```

//--- פעולה המקבלת תור של נבדקים ומחזירה את קוד העיר שבה הכי הרבה חולים ---
public static int MostSick(Queue<CovidTest> q)
{
    int max = 0;
    int cityCode = 0;

    int num = 0;
    while (!q.IsEmpty())
    {
        int code = q.Head().GetCityCode();
        num = NumOfSickInCity(q, code);
        if (num > max)
        {
            max = num;
            cityCode = code;
        }
    }
    return cityCode;
}

//--- פעולה המקבלת תור של חולים וקוד של עיר ומחזירה את מספר החולים מעיר זו בתור ---
//--- בסיום הפעולה לא יהיו יותר נבדקים מעיר זו בתור ---
public static int NumOfSickInCity(Queue<CovidTest> q, int code)
{
    Queue<CovidTest> qTemp = new Queue<CovidTest>();
    int count = 0;
    CovidTest ct;

    while (!q.IsEmpty())
    {
        ct = q.Remove();
        if (ct.GetCityCode() == code)
            count++;
        else
            qTemp.Insert(ct);
    }

    while (!qTemp.IsEmpty())
        q.Insert(qTemp.Remove());

    return count;
}

```

שאלה 7

א. sod1 (123)

x	x < 10	משפט זימון	ערך מוחזר
123	לא	sod1 (12)	
12	לא	sod1 (1)	
1	כן		1
הפעולה מחזירה: 1			

מטרת הפעולה: הפעולה מחזירה את הספרה המשמעותית ביותר במספר (הספרה השמאלית ביותר)

ב. sod2 (123)

x	x < 10	משפט זימון	ערך מוחזר
123	לא	sod2 (12)	$\underline{2} * 10 + 3 = 23$
12	לא	sod2 (1)	$\underline{0} * 10 + 2 = 2$
1	כן		0
הפעולה מחזירה: 23			

מטרת הפעולה: הפעולה מחזירה את המספר ללא הספרה המשמעותית ביותר שבו (ללא הספרה השמאלית ביותר)

ג. sod3 (123, 68)

x	y	y == 10	tmp1	tmp2	משפט זימון	ערך מוחזר
123	68	לא	1236	8	sod3 (1236, 8)	
12	8	לא	12368	0	sod3 (12368, 0)	
1	0	כן				12368
(1) הפעולה מחזירה: 12368						
(2) עבור sod3(35, 792) יוחזר 35792						
(3) מטרת הפעולה: להחזיר מספר שלם חדש שתחילתו המספר הראשון וסופו המספר השני ("לשרשר" את המספר השני לסופו של המספר הראשון)						

פרק פיסי

לפניך שאלות מ-4 מסלולים שונים: מערכות מחשב ואסמבלי (שאלות 8-9), מבוא לחקר ביצועים (שאלות 10-11), מודלים חישוביים (שאלות 12-13), תכנות מונחה עצמים בשפת Java (שאלות 14-15), תכנות מונחה עצמים בשפת C# (שאלות 16-17).

מערכות מחשב ואסמבלי

פתרון לפרק זה נכתב ע"י: רונית (מרציאנו) גל-אור

עאלה 8

עאלה 9

מבוא לחקר ביצועים

עלף 10

עלף 11

מודלים חישוביים

נכתב ע"י: רחל לודמר

עאלה 12

עאלה 13

תכנות מונחה עצמים בשפת Java

נכתב ע"י: אביטל Evi גרינוואלד

עאלה 14

עאלה 15

תכנות מונחה עצמים בשפת C#

נכתב ע"י: דיתה אוהב ציון

אלף 16

אלף 17

בהצלחה !