חומרים שהוכנו על-ידי משתתפי קורס מורים מובילים תשע"ה

ניתן להשתמש בחומרים לצורך הוראה בלבד.

לא ניתן לפרסם את החומרים או לעשות בהם כל שימוש מסחרי

ללא קבלת אישור מראש מצוות הפיתוח

כתיבה ועריכה:

אביטל גרינולד, דורית כהן, משה שטיינר

פעולות בנייה

1. פעולה המקבלת מספר טבעי n ומחזירה רשימה של מספרים טבעיים בסדר עולה מ 1 עד n כולל.

יעילות: O(n)

```
public static Node<Integer> create1_N(int n)
{
        Node<Integer> list=null;
        for (int k=n; k>0;k--)
        {
            list = new Node<Integer>(k,list);
        }
        return list;
}
```

2. פעולה המקבלת הפנייה לחוליה ראשונה ברשימה של מספרים שלמים ומדפיסה אותה כך:

n1 → n2 → … → null . אם הרשימה ריקה – יודפס רק null. יעילות: O(n)

```
public static void print(Node<Integer> list)
{
        while (list!=null)
        {
            System.out.print(list.getValue()+" --> ");
            list = list.getNext();
        }
        System.out.println("null");
}
```

3. פעולה היוצרת רשימה עם n מספרים אקראיים בין low ל up כולל כאשר n מספר טבעי ומחזירה הפנייה לחוליה הראשונה ברשימה. יעילות: O(n)

```
public static Node<Integer> buildRandom(int n,int low,int up)
{
        Random rnd = new Random();
        int num;
        Node<Integer> list = null;
        for (int k=1; k<=n; k++)
        {
            num = rnd.nextInt(up-low+1)+low;
            //System.out.println("num = "+num);
            list = new Node<Integer>(num,list);
        }
        return list;
}
```

4. פעולה הבונה רשימה של מספרים שנקלטים מהמשתמש. סוף קלט 99.
הפעולה מחזירה הפנייה לחוליה ראשונה ברשימה, null אם היא ריקה. יעילות: O(n)

```
public static Node<Integer> create()
{
   Scanner in = new Scanner(System.in);
   int x;
   Node<Integer> first, node, pos;
   System.out.println("Enter integer , 99 to stop");
   x = in.nextInt();
   if (x==99)
   {// an empty linked-list
            return null;
   }
   // list is not empty
   first = new Node<Integer>(x);
   pos = first;            // save reference to first node
   System.out.println("Enter integer value, 99 to stop");
   x = in.nextInt();
   while (x!=99)
   {
      node = new Node<Integer>(x);    // create new node
      pos.setNext(node);              // connect previous node to current node
      pos = node;                     // pos refer to last node added
      System.out.println("Enter integer value, 99 to stop");
      x = in.nextInt();
   }
   return first;
}
```

פעולות החזרת ערך מספרי

5. פעולה המקבלת רשימה של מספרים שלמים ומחזירה את אורכה. יעילות: O(n)

```
public static int count(Node<Integer> list)
{
    int counter = 0;
    Node<Integer> pos = list;
    while (pos!=null)
    {
          counter++;
          pos = pos.getNext();
    }
    return counter;
}
```

5. אותה פעולה במימוש רקורסיבי. יעילות: O(n)

```
public static int countR(Node<Integer> list)
{
   if (list == null)
         return 0;
   return 1 + countR(list.getNext());
}
```

6. פעולה המקבלת רשימה של מספרים שלמים ומחזירה את סכום ערכיה. יעילות: O(n)

```
public static int sum(Node<Integer> list)
{
   int sum = 0;
   Node<Integer> pos = list;
   while (pos!=null)
   {
         sum = sum +  pos.getValue();
         pos = pos.getNext();
   }
   return sum;
}
```

6. אותה פעולה במימוש רקורסיבי. יעילות: O(n)

```
public static int sumR(Node<Integer> list)
{
   if (list == null)
         return 0;
   return list.getValue() + sumR(list.getNext());
}
```

7. פעולה המקבלת רשימה של מספרים שלמים ומחזירה את ממוצע ערכיה. 0 - אם הרשימה ריקה.
יעילות: O(n)

```
public static double average(Node<Integer> list)
{
   if (count(list) != 0)
         return (double)(sum(list))/count(list);
   return 0;
}
```

7. אותה פעולה במימוש רקורסיבי. יעילות: O(n)

```
public static double averageR(Node<Integer> list)
{
   if (countR(list) != 0)
         return (double)(sumR(list))/countR(list);
   return 0;
}
```

8. פעולה המקבלת רשימה של מספרים שלמים ומחזירה את מכפלת ערכיה. הנחה: הרשימה אינה ריקה. יעילות: O(n)

```
public static int product(Node<Integer> list)
{
   int prod;
   if (list == null)
         prod = 0;
   prod = 1;
   Node<Integer> pos = list;
   while (pos!=null)
   {
         prod = prod * pos.getValue();
         pos = pos.getNext();
   }
   return prod;
}
```

8. אותה פעולה במימוש רקורסיבי. יעילות: O(n)

```
public static int productR(Node<Integer> list)
{
    if (list == null)
        return 0;
    if (!list.hasNext())
        return list.getValue();
    return list.getValue() * productR(list.getNext());
}
```

9. פעולה המקבלת רשימה של מספרים שלמים ומחזירה את הערך המקסימלי . הנחה: הרשימה אינה ריקה. יעילות: O(n)

```
public static int max(Node<Integer> list)
{
   int max = list.getValue();
   Node<Integer> pos = list.getNext();
   while (pos!=null)
   {
         if (pos.getValue() > max)
          max = pos.getValue();
         pos = pos.getNext();
   }
   return max;
}
```

9. מימוש רקורסיבי של אותה פעולה. יעילות: O(n)

```
public static int maxR(Node<Integer> list)
{
    if (!list.hasNext())
        return list.getValue();
    return Math.max(list.getValue(), maxR(list.getNext()));
}
```

10. פעולה המקבלת רשימה של מספרים שלמים ומחזירה את הערך הכי קטן . הנחה: הרשימה אינה ריקה. יעילות: O(n)

```
public static int min(Node<Integer> list)
{
   int min = list.getValue();
   Node<Integer> pos = list.getNext();
   while (pos!=null)
   {
         if (pos.getValue() < min)
             min = pos.getValue();
         pos = pos.getNext();
   }
   return min;
}
```

10. מימוש רקורסיבי של אותה פעולה. יעילות: O(n)

```
public static int minR(Node<Integer> list)
{
   if (!list.hasNext())
         return list.getValue();
   return Math.min(list.getValue(), minR(list.getNext()));
}
```

11. פעולה המקבלת רשימה של מספרים שלמים ומחזירה את סכום הערכים הזוגיים. יעילות: O(n)

```
public static int sumEven(Node<Integer> list)
{
   int sum = 0;
   Node<Integer> pos = list;
   while (pos!=null)
   {
         if (pos.getValue()%2 == 0)
          sum = sum  + pos.getValue();
         pos = pos.getNext();
   }
   return sum;
}
```

11. מימוש רקורסיבי של אותה פעולה. יעילות: O(n)

```
public static int sumEvenR(Node<Integer> list)
{
   if (list == null)
         return 0;
   if (list.getValue()%2 == 0)
         return list.getValue() + sumEvenR(list.getNext());
   return sumEvenR(list.getNext());
}
```

12. פעולה המקבלת רשימה של מספרים שלמים ומחזירה את סכום הערכים במקומות האי-זוגיים.
יעילות: O(n)

```
public static int sumOddPlaces(Node<Integer> list)
{
   int sum = 0;
   Node<Integer> pos = list;
   while (pos!=null)
   {
         sum = sum + pos.getValue();
         pos = pos.getNext();
         if (pos!=null)
          pos = pos.getNext();
   }
   return sum;
}
```

12. מימוש רקורסיבי של אותה פעולה. יעילות: O(n)

```
public static int sumOddPlacesR(Node<Integer> list)
{
    if (list==null)
        return 0;
    return sumOddPlacesR(list, 1);
}

private static int sumOddPlacesR(Node<Integer> list, int place)
{
    if (list==null)
        return 0;
    if (place%2==1)
        return list.getValue()+ sumOddPlacesR(list.getNext(), place+1);
    return sumOddPlacesR(list.getNext(), place+1);
}
```

13. פעולה המקבלת רשימה של מספרים שלמים ומספר שלם נוסף x , ומחזירה את מספר הפעמים שהמספר x מופיע ברשימה. יעילות: O(n)

```
public static int countX(Node<Integer> list, int x)
{
   int counter = 0;
   Node<Integer> pos = list;
   while (pos!=null)
   {
         if (pos.getValue() == x)
          counter++;
         pos = pos.getNext();
   }
   return counter;
}
```

13. מימוש רקורסיבי של אותה פעולה. יעילות: O(n)

```
public static int countXR(Node<Integer> list, int x)
{
   if (list==null)
         return 0;
   if (list.getValue() == x)
         return 1 + countXR(list.getNext(), x);
   return countXR(list.getNext(), x);
}
```

פעולות המחזירות ערך בוליאני

14. הפעולה מקבלת רשימה של מספרים שלמים ומחזירה 'אמת' אם היא ממוינת מסודרת בסדר עולה, 'שקר' – אחרת. עבור רשימה ריקה תחזיר 'אמת'. יעילות: O(n)

```
public static boolean isUp(Node<Integer> list)
{
      if (list==null)
          return true;
      if (!list.hasNext())
          return true;
      while (list.hasNext())
      {
          if (list.getValue()>=list.getNext().getValue())
              return false;
          list = list.getNext();
      }
      return true;
}
```

14. מימוש רקורסיבי של אותה פעולה. יעילות: O(n)

```
public static boolean isUpR(Node<Integer> list)
{
   if (list==null)
       return true;
   if (!list.hasNext())
       return true;
   return (list.getValue() < list.getNext().getValue()) &&
           isUpR(list.getNext());
}
```

15. הפעולה מקבלת רשימה של מספרים שלמים ומספר שלם נוסף x, ומחזירה 'אמת' אם המספר x נמצא ברשימה, 'שקר' – אחרת. יעילות: O(n)

```
public static boolean isExist(Node<Integer> list, int x)
{
    while (list!=null)
    {
        if (list.getValue()==x)
            return true;
        list = list.getNext();
    }
    return false;
}
```

15. מימוש רקורסיבי של אותה פעולה. יעילות: O(n)

```
public static boolean isExistR(Node<Integer> list, int x)
{
    if (list==null)
        return false;
    if (list.getValue() == x)
        return true;
    return isExistR(list.getNext(), x);
}
```

16. הפעולה מקבלת רשימה של מספרים שלמים ומחזירה 'אמת' אם היא מהווה סדרה חשבונית, 'שקר' – אחרת. עבור רשימה ריקה תחזיר 'אמת'. יעילות: O(n)

סדרה חשבונית היא סדרת מספרים שההפרש בין כל שני ערכים סמוכים זהה.

```
public static boolean isArithmetic(Node<Integer> list)
{
     if (list==null)
         return true;
     if (!list.hasNext())
         return true;
     int first, second, diff;
     // at least 2 values
     first = list.getValue();
     list= list.getNext();
     second = list.getValue();
     diff = second - first;
     while (list.hasNext())
     {
         first = second;
         list = list.getNext();
         second = list.getValue();
         if (second - first != diff)
             return false;
     }
     return true;
}
```

16. מימוש רקורסיבי של אותה פעולה. יעילות: O(n)

```
public static boolean isArithmeticR(Node<Integer> list)
{
    if (list==null)
        return true;
    if (!list.hasNext())
        return true;
    int diff = list.getNext().getValue()-list.getValue();
    return isArithmeticR(list.getNext(), diff);
}

private static boolean isArithmeticR(Node<Integer> list, int d)
{
    if (!list.hasNext())
        return true;
    if ( list.getNext().getValue() - list.getValue() != d)
        return false;
    return isArithmeticR(list.getNext(), d);
}
```

פעולות המחזירות הפנייה למקום ברשימה

17. פעולה המקבלת רשימה של מספרים שלמים לא ריקה ומספר צעדים step ומחזירה את ההפניה לחוליה במקום step.

אם step גדול ממספר החוליות, תחזיר null . יעילות: O(n)

```
public static Node<Integer> goTo(Node<Integer> list, int step)
{
     while (list!=null && step>1)
     {
         list = list.getNext();
         step--;
     }
     return list;
}
```

17. מימוש רקורסיבי של אותה פעולה. יעילות: O(n)

```
public static Node<Integer> gotoR(Node<Integer> list, int step)
{
     if (step==1 || list==null)
         return list;
     return gotoR(list.getNext(), step-1);
}
```

18. פעולה המקבלת רשימה של מספרים שלמים לא ריקה ומחזירה את ההפניה לחוליה אחרונה, או null אם הרשימה ריקה. יעילות: O(n)

```
public static Node<Integer> gotoLast(Node<Integer> list)
{
     if (list==null)
         return null;
     while (list.hasNext())
         list = list.getNext();
     return list;
}
```

18. מימוש רקורסיבי של אותה פעולה. יעילות: O(n)

```
public static Node<Integer> gotoLastR(Node<Integer> list)
{
     if (list==null)
         return null;
     if (!list.hasNext())
         return list;
     return gotoLastR(list.getNext());
}
```

19. פעולה המקבלת רשימה של מספרים שלמים ומספר שלם נוסף x ומחזירה הפניה למקום ההופעה הראשון של x מתחילת הרשימה, או null – אם לא נמצא. יעילות: O(n)

```
public static Node<Integer> firstPosX(Node<Integer> list, int x)
{
    while (list!=null && list.getValue()!=x)
        list= list.getNext();
    return list;
}
```

19. מימוש רקורסיבי של אותה פעולה

```
public static Node<Integer> firstPosX_R(Node<Integer> list, int x)
{
    if (list==null || list.getValue()==x)
        return list;
    return firstPosX_R(list.getNext(), x);
}
```

20. פעולה המקבלת רשימה של מספרים שלמים ומחזירה הפנייה למיקום הערך הקטן ביותר ברשימה.

הנחות: הרשימה אינה ריקה. כל הערכים שונים זה מזה. יעילות: O(n)

```
public static Node<Integer> posMin(Node<Integer> list)
{
    Node<Integer> minPos = list;
    int val, minVal;
    if (list == null)
        return null;
    minVal = list.getValue();
    list = list.getNext();
    while (list!=null)
    {
        val = list.getValue();
        if (val<minVal)
        {
            minVal = val;
            minPos = list;
        }
        list = list.getNext();
    }
    return minPos;
}
```

שינוי רשימה קיימת, בפועל מחזיר רשימה חדשה

21. פעולה המקבלת רשימה, מקום ברשימה pos וערך x ומחזירה רשימה חדשה כאשר הערך x נוסף מקום אחד אחרי pos . אם pos=null , תוחזר רשימה בה x התווסף לפני חוליה ראשונה.

יעילות הפעולה היא O(1)

```
public static Node<Integer> insert(Node<Integer> list, Node<Integer> pos, int x)
{
    Node<Integer> node = new Node<Integer>(x);
    if (list == null)
    {// list is empty, return list with one value x
        return node;
    }
    if (pos==null)
    {// insert before first
        node.setNext(list);
        return node;
    }
    // insert after pos
    node.setNext(pos.getNext());
    pos.setNext(node);
    return list;
}
```

22. פעולה המקבלת רשימה, מספר צעדים step וערך x ומחזירה רשימה חדשה בה הערך x נוסף אחרי step צעדים מתחילת הרשימה. אם step=0 , x יתווסף לפני חוליה ראשונה. אם step גדול מאורך הרשימה, x יתווסף בסוף הרשימה. יעילות: O(n)

```
public static Node<Integer> insert(Node<Integer> list, int step, int x)
{
    Node<Integer> node = new Node<Integer>(x);
    if (list==null)
        return node;
    if (step==0)
    {
        node.setNext(list);
        return node;
    }
    Node<Integer> pos = list;
    while (step>1 && pos!=null)
    {
        pos = pos.getNext();
        step--;
    }
    // insert after pos
    node.setNext(pos.getNext());
    pos.setNext(node);
    return list;
 }
```

23. פעולה המקבלת רשימה ומקום ברשימה prev ומחזירה רשימה חדשה ללא החוליה העוקבת ל prev.
אם prev=null , תוחזר רשימה ללא החוליה הראשונה.

יעילות הפעולה היא O(1)

```
public static Node<Integer> delete(Node<Integer> list, Node<Integer> prev)
{
    if (list==null)
        return list;
    if (prev==null)
        return list.getNext();
    if (prev.hasNext())
    {
        Node<Integer> pos = prev.getNext();
        prev.setNext(pos.getNext());
        pos.setNext(null);
    }
    return list;
}
```

24. פעולה המקבלת רשימה ומספר צעדים step מתחילת הרשימה ומחזירה רשימה חדשה ללא הערך הנמצא במיקום step . יעילות: O(n)

```
public static Node<Integer> delete(Node<Integer> list, int step)
{
    if (list==null)
        return list;
    if (step<=1)
        return list.getNext();
    Node<Integer> prev = list;
    while (step >2 && prev.hasNext())
    {
        prev = prev.getNext();
        step--;
    }
    if (prev.hasNext())
    {
        Node<Integer> pos = prev.getNext();
        prev.setNext(pos.getNext());
        pos.setNext(null);
    }
    return list;
}
```

פעולות על רשימות ממוינות

25. פעולה המקבלת רשימה ממוינת בסדר עולה ומספר שלם x ומחזירה רשימה חדשה הכוללת את הערך x כך שהיא שומרת על המיון שלה. יעילות: O(n)

```
public static Node<Integer> insertSort(Node<Integer> list, int x)
{
    Node<Integer> node = new Node<Integer>(x);
    if (list==null)
        return node;
    Node<Integer>pos = list;
    if (x<pos.getValue())
    {// add before first node
        node.setNext(pos);
        return node;
    }
    while (pos.hasNext() && x > pos.getNext().getValue())
        pos = pos.getNext();
    node.setNext(pos.getNext());
    pos.setNext(node);
    return list;
}
```

26. פעולה המקבלת שתי רשימות ממוינות בסדר עולה. בכל רשימה הערכים שונים זה מזה. הפעולה מחזירה רשימה חדשה שהיא מיזוג של שתי הרשימות. כל ערך מופיע רק פעם אחת לכל היותר. הרשימה החדשה מכילה את הערכים משתי הרשימות בסדר עולה.

מותר לשנות את הרשימות המקוריות. המיזוג יתבצע ביעילות ליניארית. O(n+m)

26. מימוש ראשון, **ללא שינוי הרשימות המקוריות**.
מוחזרת רשימה חדשה שהיא רשימת המיזוג, תוך יצירת חוליות חדשות.

```
public static Node<Integer> merge (Node<Integer> list1, Node<Integer> list2)
{
        if (list1==null)
            return list2;
        if (list2==null)
            return list1;
        Node<Integer> mergedList = null;
      Node<Integer> pos1, pos2, pos3;

      pos1 = list1;
      pos2 = list2;
      pos3 = mergedList;

      while (pos1 != null && pos2 != null)
      {
            if (pos1.getValue() < pos2.getValue())
            {
                mergedList = insert(mergedList, pos3, pos1.getValue());
             pos1 = pos1.getNext();
            }
            else
            {
             if (pos1.getValue() > pos2.getValue())
             {
                    mergedList = insert(mergedList, pos3, pos2.getValue());
                    pos2 = pos2.getNext();
             }
             else
             {
                    mergedList = insert(mergedList, pos3, pos2.getValue());
                    pos1 = pos1.getNext();
                    pos2 = pos2.getNext();
             }
            }
            if (pos3 == null)  // first node in merged list
                pos3 = mergedList;
            else
             pos3 = pos3.getNext();
      }
      while (pos1 != null)
      {//  tail of list1
            mergedList = insert(mergedList, pos3, pos1.getValue());
            pos1 = pos1.getNext();
            pos3 = pos3.getNext();
      }
      while (pos2 != null)
      {// tail of list2
            mergedList = insert(mergedList, pos3, pos2.getValue());
            pos2 = pos2.getNext();
            pos3 = pos3.getNext();
      }
      return mergedList;
   }
```

26. מימוש שני , בה הרשימה הראשונה מוחזרת כרשימת המיזוג. **יש שימוש בחוליות קיימות**.

```
public static Node<Integer> merge1(Node<Integer> list1, Node<Integer> list2)
{
    if (list1==null)
        return list2;
    if (list2==null)
        return list1;
    // list1!=null && list2!=null
    int x,y;
    Node<Integer> mrgList;    // refer to merged sortedlist
    // choose reference to mrgList
    if (list1.getValue()<list2.getValue())
    {
        mrgList = list1;
        list1 = list1.getNext();
    }
    else
    {// list1.getValue()>=list2.getValue()
        mrgList = list2;
        list2 = list2.getNext();
    }
    Node<Integer> posMrg = mrgList;
    while (list1!=null && list2!=null)
    {
        if (list1.getValue()<list2.getValue())
        {
            posMrg.setNext(list1);
            list1 = list1.getNext();
        }
        else
        {// list1.getValue()>=list2.getValue()
            posMrg.setNext(list2);
            list2 = list2.getNext();
        }
        posMrg = posMrg.getNext();
    }// end while
    while (list1!=null)
    {// tail of list1
        posMrg.setNext(list1);
        list1 = list1.getNext();
        posMrg = posMrg.getNext();
    }
    while (list2!=null)
    {// tail of list2
        posMrg.setNext(list2);
        list2 = list2.getNext();
        posMrg = posMrg.getNext();
    }
    return mrgList;
}
```

27. פעולה המקבלת שתי רשימות ממוינות בסדר עולה. בכל רשימה הערכים שונים זה מזה. הפעולה מחזירה רשימה חדשה שהיא **חיתוך** של שתי הרשימות. כלומר: הרשימה החדשה תכיל את הערכים הנמצאים בשתי הרשימות והיא בסדר עולה. מותר לשנות את הרשימות המקוריות. החיתוך יתבצע ביעילות ליניארית. $O(n+m)$

מימוש ראשון, **ללא שינוי הרשימות המקוריות** ומוחזרת רשימה חדשה תוך יצירת חוליות חדשות.

```
public static Node<Integer> cut(Node<Integer> list1, Node<Integer> list2)
{
        Node<Integer> listCut=null;
        Node<Integer> posCut=listCut;
        while (list1!=null && list2!=null)
        {
            if (list1.getValue()<list2.getValue())
                list1 = list1.getNext();
            else
                if (list1.getValue()>list2.getValue())
                    list2 = list2.getNext();
                else
                {// list1.getValue()==list2.getValue()
                    listCut = insert(listCut, posCut, list1.getValue());
                    list1 = list1.getNext();
                    list2 = list2.getNext();
                    if (posCut==null)   // first node
                        posCut = listCut;
                    else
                        posCut = posCut.getNext();
                }
        }// end while
        return listCut;
}
```

27. מימוש שני , בה הרשימה הראשונה מוחזרת כרשימת החיתוך. **יש שימוש בחוליות קיימות**.

```
public static Node<Integer> cut1(Node<Integer> list1, Node<Integer> list2)
{
    Node<Integer> listCut;
    Node<Integer> posCut;
    Node<Integer> tempPos;
    // search first common value
    while (list1!=null && list2!=null &&
            list1.getValue()!=list2.getValue())
    {
        if (list1.getValue()<list2.getValue())
            list1 = list1.getNext();
        else    // list1.getValue()>list2.getValue()
            list2 = list2.getNext();
    }
    if (list1==null || list2==null)
        return null;
    // list1.getValue()==list2.getValue()
    listCut = list1;    // for example
    posCut = listCut;
    int val1,val2;
    while (posCut.hasNext())
    {
        val1 = posCut.getNext().getValue();
        val2=list2.getValue();
        if (val1==val2)
        {// leave this value in list1
            posCut = posCut.getNext();
            list2 = list2.getNext();
        }
        else
        {
            if (val1<val2)
            {// removes value from list1
                tempPos = posCut.getNext();
                posCut.setNext(tempPos.getNext());
                tempPos.setNext(null);
            }
            else    // go on with list2
                list2 = list2.getNext();
        }
    }
    return listCut;
}
```

פעולות על שתי רשימות

28. פעולה המקבלת שתי רשימות list1, list2 , ומחזירה 'אמת' אם כל איברי list2 נמצאים בסדר כלשהו ב-list1, 'שקר'-אחרת. יעילות: O(n*m)

```
public static boolean isAllExist(Node<Integer> list1, Node<Integer> list2)
{
      if (list2==null)
          return true;
      if (list1==null)
          return false;
      int x = list2.getValue();
      return isExistR(list1, x) && isAllExist(list1, list2.getNext());
}
```

29. פעולה המקבלת שתי רשימות list1, list2 ומחזירה 'אמת' אם שתי הרשימות מכילות אותן ערכים ללא חשיבות לסדר, 'שקר'-אחרת. יעילות: O(n*m)

```
public static boolean isSameValues(Node<Integer> list1, Node<Integer> list2)
    {
        return isAllExist(list1, list2) &&
               isAllExist(list2, list1) ;
    }
```

30. פעולה המקבלת שתי רשימות list1, list2 ומחזירה 'אמת' אם לשתי הרשימות אין ערכים משותפים, 'שקר'-אחרת. יעילות: O(n*m)

```
public static boolean isAllDifferent(Node<Integer> list1, Node<Integer> list2)
  {
      int x;
      while (list1!=null)
      {
          x = list1.getValue();
          if (isExist(list2, x))
              return false;
          list1 = list1.getNext();
      }
      return true;
  }
```

30a. מימוש בעזרת הפעולה cut (שנכתבה בסעיף 27) יעילות: O(n*m)

```
public static boolean isAllDifferentSort
                      (Node<Integer> list1, Node<Integer> list2)
{
    return ( cut(list1,list2)==null);
}
```

31. פעולה המקבלת שתי רשימות list1, list2 ומחזירה 'אמת' אם הרשימה list2 מוכלת בשלמותה ברשימה list1 החל מראשית הרשימה list1, 'שקר'-אחרת. יעילות: O(n), כאשר n הוא אורך הרשימה list2.

```
public static boolean isIncludeStart(Node<Integer> list1,Node<Integer> list2)
{
    int val1,val2;
    while (list1!=null && list2!=null)
    {
        if (list1.getValue()!=list2.getValue())
            return false;
        list1 = list1.getNext();
        list2 = list2.getNext();
    }
    return list2==null;
}
```

32. פעולה המקבלת שתי רשימות list1, list2 ומחזירה 'אמת' אם רשימה list2 מוכלת בשלמותה ברשימה list1, 'שקר'-אחרת. יעילות: O(n*m)

```
public static boolean isInclude (Node<Integer> list1, Node<Integer> list2)
{
    while (list1!=null)
    {
        if (isIncludeStart(list1, list2))
            return true;
        list1 = list1.getNext();
    }
    return false;
}
```

פעולות מיון רשימות

33. פעולה הבונה רשימה ממוינת בסדר עולה ממספרים שנקלטו מהמשתמש. סוף קלט 99.
הפעולה מחזירה הפנייה לחוליה הראשונה ברשימה, null אם היא ריקה. יעילות: $O(n^2)$

```
public static Node<Integer> createSorted()
{
    Scanner in = new Scanner(System.in);
    Node<Integer> list=null;
    int x;
    System.out.println("Enter integer , 99 to stop");
    x = in.nextInt();
    while (x!=99)
    {
        list = insertSort(list, x);
        System.out.println("Enter integer , 99 to stop");
        x = in.nextInt();
    }
    return list;
}
```

34. פעולה המקבלת רשימה של מספרים בסדר כלשהו ומחזירה רשימה ממוינת בסדר עולה בשיטת **מיון-הכנסה insertion-sort**. יעילות: $O(n^2)$

```
public static Node<Integer> insertionSort(Node<Integer> list)
{
     int x;
     Node<Integer> sortList = null;
     while (list!=null)
     {
         x = list.getValue();
         sortList = insertSort(sortList, x);
         list = list.getNext();
     }
     return sortList;
}
```

35. פעולה המקבלת רשימה של מספרים בסדר כלשהו ומחזירה רשימה ממוינת בסדר עולה בשיטת **מיון-בחירה selection-sort** . יעילות: $O(n^2)$

```
public static void selectionSort(Node<Integer> list)
{
     int temp;
     Node<Integer> pos, minPos;
     pos = list;
     while (pos!=null)
     {
         minPos = posMin(pos);
         if (pos!=minPos)
         {
             temp = pos.getValue();
             pos.setValue(minPos.getValue());
             minPos.setValue(temp);
         }
         pos = pos.getNext();
     }
}
```