

## סיכום נושאים וחזרה לפגרות

- **רקורסיה**
  - מעקב רקורסיה
  - רקורסיה כפולה
  - רקורסיה ברשימה
  - רקורסיה במחסנית/ תור
- **עצמים ומחלקות**
  - אוספים במערך
  - אוספים ברשימה
  - אוספים במחסנית / בתור
  - פעולות על אוספים.
- **מערך של עצמים**
  - מערך של רשימות
  - מערך של תורים / מחסניות
- **סריקות**
  - סריקת מערך
  - סריקת מערך באוסף
  - סריקת רשימות - עד נקודה ידועה
  - סריקת מחסנית - עד נקודה ידועה, מחסנית עזר
  - סריקת תור - עד הסוף, תור עזר
  - סריקת עץ - סדר תחילי / תוכי / תחילי
- **תבניות אלגוריתמיות :**
  - אורך רשימה, מחסנית, תור
  - בניית רשימה** מההתחלה, מהסוף (דף : [בנייה והדפסה של שרשרת](#))
  - הכנסה בצורה ממוינת** לרשימה, למחסנית ולתור
  - תת-רשימה עולה (דף : [תת סדרה עולה](#))
  - נמצא** ברשימה, במחסנית, בתור, בעץ
  - הוצאה** מרשימה
  - צמצום רשימה**
  - צמצום מחסנית** (שכפול מחסנית)
  - צמצום תור** (שכפול התור באמצעות תור עזר או באמצעות תור מעגלי)
- **עצים**
  - תבניות אלגוריתמיות (דפים : [תבניות אלגוריתמיות](#), [אוסף פעולות](#), [תבניות בעצים](#))
  - סריקת עץ לפי רמות
  - סכום רמות בעץ
  - הורים וצאצאים בעץ

## • רקורסיה

מעקב רקורסיה

רקורסיה כפולה

רקורסיה ברשימה

רקורסיה במחסנית/ תור

$$5! = ?? = 120$$

--- פעולה המקבלת מספר שלם ומחזירה את העצרת שלו ---  
 --- הנחה: המספר אינו שלילי ---

```
public static int factorial (int num)
{
    if (num == 0) return 1;
    return num * factorial (num-1);
}
```

n	n == 0	משפט זימון	ערך מוחזר
5	F	factorial(4)	5 * 24 = 120
4	F	factorial(3)	4 * 6 = 24
3	F	factorial(2)	3 * 2 = 6
2	F	factorial(1)	2 * 1 = 2
1	F	factorial(0)	1 * 1 = 1
0	True		1

ערך מוחזר: 120      5! = 120

## טבלת מעקב לרקורסיה:

עמודה לכל משתנה

עמודה לכל תנאי

עמודה למשפט זימון

עמודה לערך המוחזר

- לא לשכוח לסמן חץ בכיוון הזימון וחץ בכיוון ההחזרה  
 מעקב על מערך - עמודה לכל תא במערך המשתתף בתנאים  
 מעקב על רשימה / מחסנית / תור - עמודה למצב מבנה הנתונים (מוצג ה- toString של המבנה)

## רקורסיה כפולה:

(לדוגמה: פיבונאצ'י)

עץ מעקב

אין צורך להראות חישובים (אלא אם כן הם נוגעים לעצם המעקב ואינם חישובים פשוטים).

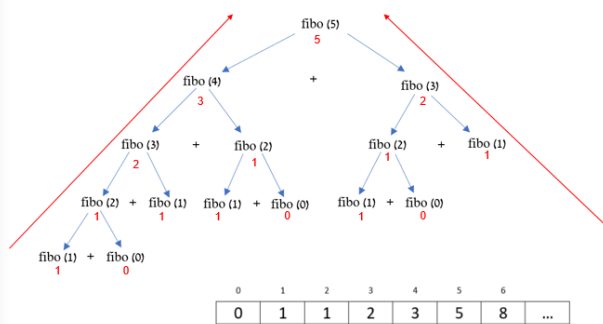
שימוש בשני צבעים: האחד לזימון הרקורסיבי והאחר לחזרה מהרקורסיה.

## סדרת פיבונאצ'י

int num = fibonacci (5);

הגדרה: fibo = fibonacci

עץ מעקב -  
 מעקב אחר רקורסיה כפולה



- **עצמים ומחלקות**  
אוספים במערך  
אוספים ברשימה  
אוספים במחסנית / בתור  
פעולות על אוספים.

**צמ קסיס:**

תכונות, בנאים, Get / Set לתכונות, ToString  
פעולות חישוביות במידת הצורך.

**אוספ:****תכונות:**

תכונות מזהות (בסיסיות): שם, מס' מזהה וכד'  
אוסף במערך:  
גודל המערך כתכונה סטטית, המערך, lastPosition.  
אוסף ברשימה / שרשרת חוליות, במחסנית, בתור:  
הפנייה למבנה הנתונים

**פעולות:**

פעולה בונה: מקבלת כפרמטר רק ערכים לתכונות הבסיסיות.  
יוצרת אוסף ריק (אינה מקבלת אוסף)  
פעולת הוספה: Add מקבלת עצם מאותחל ורק מוסיפה אותו לאוסף.  
במערך - עדכון lastPosition  
בשרשרת חוליות - אם לא נאמר אחרת - בתחילת הרשימה.  
האם האוסף ריק? IsEmpty  
פעולות איתור - Get - לפי תכונה של העצם במיוצג באוסף  
פעולות מחיקה Remove או Delete - המקבלת תכונה, מאתרת, מוציאה ומחזירה את הערך שהוצא.  
ToString  
פעולות חישוב במידת הצורך.

אין פעולת Get לאוסף (למערך/לרשימה...) - התכנית לא מכירה את מבנה האוסף

**אוסף מורכב:**

אוסף שאחת מתכונותיו מכילה עצם או אוסף  
האוסף אינו מכיר את מבנה התכונה המכילה את העצם או את האוסף.  
המחלקה המגדירה את התכונה חייבת להכיל פעולות המספקות תשובות.  
האוסף פונה לפעולות אלו.

לדוגמה: המחלקה Person שאחת מתכונותיה היא ת.לידה birthday  
תפנה ליום הלידה ב- `this.birthday.GetDay()`

**דוגמאות:**

- בגרות 2007 - מסעדה טעמים 22

מחלקה המגדירה אוסף	מחלקה המגדירה עצם
<b>תכונות:</b> מערך של עצמים קבוע המגדיר את גודל המערך מונה מספר העצמים last	<b>תכונות:</b> תכונות בסיסיות המגדירות את העצם
<b>פעולות:</b> בנאי (היוצר אוסף ריק) isEmpty - האם ריק? add - הוספת פריט לאוסף remove - הסרת פריט לפי תכונה get - איתור והחזרת הפנייה לפריט לפי תכונה toString פעולות נוספות במידת הצורך	<b>פעולות:</b> בנאי פעולת אחזור - get לכל תכונה פעולת עדכון - set לכל תכונה שהגיוני לעדכן הפעולה toString פעולות חישוביות במידת הצורך

- מערך
  - מערך בסיסי
  - מערך של עצמים
  - מערך של רשימות
  - מערך של תורים / מחסניות

הגדרת מערך:

**מצדק של טיפוס בסיסי**, למשל מספרים שלמים (כל תא מכיל 0)

```
int [] arrInt = new int [n];
```

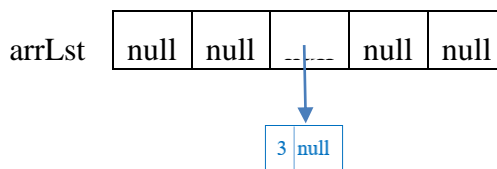
**מצדק של עצמים** (כל תא מכיל null) שאר איברי המערך מכילים null

```
Book [] arrBook = new Book [n]
arrBook [0] = new Book (פרמטרים של ספר);
```

**מצדק של רשימות:**

מערך של הפניות לרשימות, כל תא מכיל ערך null  
אתחול תא 2 עם חוליה שערכה 3

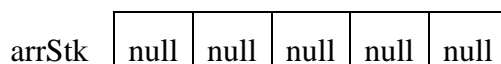
```
Node<int> [] arrLst = new Node<int> [n]
arrLst[2] = new Node<int> (3);
```



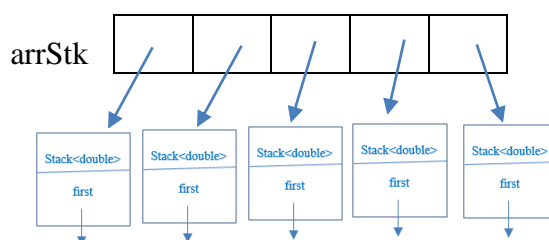
**מצדק של מחסניות:**

יצירת מערך של הפניות למחסניות  
אתחול כל הפניה במערך למחסנית ריקה.

```
Stack<double> [] arrStk = new Stack<double> [5];
for (int i = 0 ; i < arrStk.Length ; i++)
    arrStk [i] = new Stack<double> ();
```



יצירת מערך של הפניות למחסניות



אתחול כל הפניה למחסנית ריקה

## מצב של תורים:

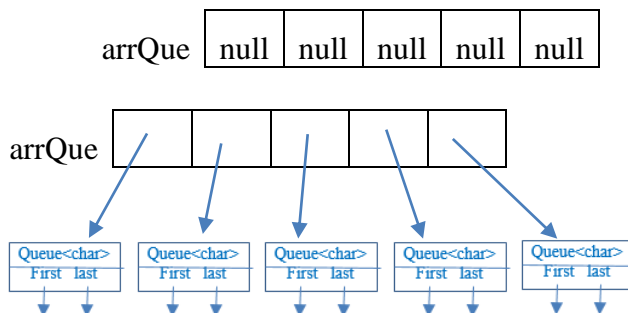
```
Queue<char> [] arrQue = new Queue<char>[5];
for (int i = 0 ; i < arrQue.Length ; i++)
    arrQue [i] = new Queue<char> ();
```

יצירת מערך של הפניות לתורים

אתחול כל הפניה במערך לתור ריק.

רק עכשיו אפשר לבצע פעולות על כל אחד מהתורים

לאחר יצירת מערך התורים:



לאחר אתחול המערך:

## • סריקות

סריקת מערך

סריקת מערך באוסף

סריקת רשימות - עד הסוף

או עד נקודה ידועה

סריקת מחסנית - עד הסוף

או עד נקודה ידועה,

סריקת תור - עד הסוף,

מחסנית עזר

תור עזר

סריקת עץ - סדר תחילי / תוכי / תחילי

## סריקת מערך:

```
for (int i = 0 ; i < arr.Length ; i++) ...
```

## סריקת מערך באוסף:

```
for (int i = 0 ; i < this.lastPosition ; i++) ...
```

## סריקת רשימה / שרשרת חוליות:

pos מקבל הפניה לתחילת הרשימה

```
Node<int> pos = lst;
while (pos != null && pos.GetValue() < x)
{
    טיפול במידע החוליה
    pos = pos.GetNext();
}
```

אם x מטיפוס מחרוזת string,  
 ההשוואה נעשית ב- == או ב- Equals  
 או ב- CompareTo

## סריקת מחסנית:

```
while (! stk.IsEmpty () && stk.Top() < x)
```

אין לשלב stk.Pop() בלולאה (אובדן איבר)

```
{
    int x = stk.Pop()
    טיפול באיבר המחסנית
    שמירת x במחסנית העזר
}
```

לא לשכוח להחזיר את האיברים ממחסנית העזר בחזרה למחסנית

## סריקת תור:

```
while (! que.IsEmpty () )
```

```
{
    int y = que.Remove()
    טיפול באיבר התור
    שמירת x בתור העזר
}
```

לא לשכוח להחזיר את האיברים מתור העזר בחזרה לתור המקורי

## סריקת עץ:

סדר תחילי - PreOrder	סדר תוכי - InOrder	סדר סופי - PostOrder
<pre>static void DoPreOrder     (BinNode&lt;int&gt; bt) {     if (bt != null)     {         טיפול ב- bt.GetValue()         DoPreOrder(bt.GetLeft());         DoPreOrder(bt.GetRight());     } }</pre>	<pre>static void DoInOrder     (BinNode&lt;int&gt; bt) {     if (bt != null)     {         DoInOrder(bt.GetLeft());         טיפול ב- bt.GetValue()         DoInOrder(bt.GetRight());     } }</pre>	<pre>static void DoPostOrder     (BinNode&lt;int&gt; bt) {     if (bt != null)     {         DoPostOrder(bt.GetLeft());         doPostOrder(bt.GetRight());         טיפול ב- bt.GetValue()     } }</pre>

סריקת המחסנית תוך שמירה על איברי המחסנית:

נועד למקרים שבהם ניתן לבצע את הפעולה במעבר אחד על המחסנית

```
Stack<int> sTemp = new Stack<int> ();
while (! stk.IsEmpty())
{
    int x = stk.Pop();
    : טיפול באיבר שנשלף מהמחסנית
    sTemp.Push (x);
}
while (! sTemp.IsEmpty())
    stk.push (sTemp.Pop());
```

שימו לב!

כשסורקים מחסנית, ניתן לעצור את הסריקה באמצע ולהחזיר את האיברים למחסנית (סדר האיברים המקורי נשמר).

סריקת התור תוך שמירה על איברי התור:

נועד למקרים שבהם ניתן לבצע את הפעולה במעבר אחד על התור

```
Queue<int> qTemp = new Queue<int> ();
while (! que.IsEmpty())
{
    int x = que.Remove();
    : טיפול באיבר שנשלף מהתור
    qTemp.Insert (x);
}
while (! sTemp.IsEmpty())
    que. insert (qTemp. Remove ());
```

שימו לב!

כשסורקים תור, אסור לעצור את הסריקה באמצע. חייבים להמשיך לרוקן את התור לתור העזר לפני שמחזירים את האיברים לתור (אחרת סדר האיברים המקורי לא יישמר ואיבר שהיה בראש התור עלול למצוא את עצמו בסופו).



## • תבניות אלגוריתמיות:

אורך רשימה, מחסנית, תור

(דף: [בנייה והדפסה של שרשרת](#)) בניית רשימה מההתחלה, מהסוף

הכנסה בצורה ממוינת לרשימה, למחסנית ולתור

(דף: [תת סדרה עולה](#)) תת-רשימה עולה

נמצא ברשימה, במחסנית, בתור, בעץ

הוצאה מרשימה

//--- פעולה המוסיפה איבר לרשימה ממוינת כך שהמיון נשמר ---

public static Node&lt;int&gt; InsertSorted(Node&lt;int&gt; lst, int x)

{

Node&lt;int&gt; pos = lst;

Node&lt;int&gt; prev = null;

//--- חיפוש המקום להכנסה ---

while (pos != null &amp;&amp; pos.GetValue() &lt; x)

{

prev = pos;

pos = pos.GetNext();

}

//--- הכנסה לשרשרת ---

if (prev == null) // הכנסה בהתחלה

lst = new Node&lt;int&gt;(x, lst);

else // הכנסה אחרי prev

prev.SetNext(new Node&lt;int&gt;(x, pos));

return lst;

}

אם x מטיפוס מחרוזת string,  
השוואה נעשית ב- == או ב- equals  
או ב- CompareTo

//--- פעולה המוסיפה איבר למחסנית ממוינת כך שהמיון נשמר ---

public static void InsertSorted(Stack&lt;int&gt; stk, int x)

{

Stack&lt;int&gt; sTemp = new Stack&lt;int&gt;();

//--- העברת האיברים הקטנים מ-x למחסנית העזר

while (!stk.IsEmpty() &amp;&amp; stk.Top() &lt; x)

sTemp.Push(stk.Pop());

stk.Push(x); // הכנסת x למקומו המתאים

//--- החזרת האיברים למחסנית ---

while (!sTemp.IsEmpty())

stk.Push(sTemp.Pop());

}

```
///--- פעולה המוסיפה איבר לתור ממויין כך שהמיון נשמר ---  
public static void InsertSorted(Queue<int> que, int x)  
{  
    Queue<int> qTemp = new Queue<int>();  
  
    ///--- העברת האיברים הקטנים מ- x לתור העזר  
    while (!que.IsEmpty() && que.Head() < x)  
        qTemp.Insert(que.Remove());  
    qTemp.Insert(x);           /// הכנסת x לתור העזר  
    ///--- העברת שאר האיברים לתור העזר ---  
    while (!que.IsEmpty())  
        qTemp.Insert(que.Remove());  
  
    ///--- החזרת האיברים לתור ---  
    while (!qTemp.IsEmpty())  
        que.Insert(qTemp.Remove());  
}
```

למעשה אין הבדל גדול בין חיפוש ברשימה ממוינת וחיפוש ברשימה לא ממוינת. שתי הפעולות דומות הן מבחינת פונקציית זמן ריצה והן מבחינת יעילות. היתרון נמצא ברשימה ממוינת, אם האיבר לא נמצא ברשימה ייפסק החיפוש כשיימצא איבר ראשון גדול ממנו בשעה שבחיפוש ברשימה לא ממוינת יימשך החיפוש עד סוף הרשימה.

**נמצא ברשימה?**

```
//===== חיפושים - האם נמצא =====
//--- חיפוש האם נמצא ברשימה לא ממוינת ---
public static bool IsExist(Node<int> lst, int x)
{
    while (lst != null)
    {
        if (lst.GetValue() == x)
            return true;
        lst = lst.GetNext();
    }
    return false;
}
```

אם x מטיפוס מחרוזת string,  
השוואה נעשית ב- == או ב- equals  
או ב- CompareTo

```
//--- חיפוש האם נמצא ברשימה ממוינת ---
public static bool DoesExist(Node<int> lst, int x)
{
    while (lst != null && lst.GetValue() < x)
        lst = lst.GetNext();

    if (lst != null && lst.GetValue() == x)
        return true;
    return false;
}
```

גם כאן ההבדל הוא כאשר המחסנית לא ממוינת והאיבר לא נמצא.

נמצא במחסנית?

```
//--- חיפוש האם נמצא במחסנית לא ממוינת ---
public static bool IsExist(Stack<int> stk, int x)
{
    Stack<int> sTemp = new Stack<int>();
    bool found = false;

    //--- חיפוש האיבר במחסנית תוך שמירת האיברים במחסנית עזר ---
    while (!stk.IsEmpty() && !found)
        if (stk.Top() == x)
            found = true;
        else
            sTemp.Push(stk.Pop());

    //--- החזרת האיברים למחסנית ---
    while (!sTemp.IsEmpty())
        stk.Push(sTemp.Pop());

    return found;
}
```

```
//--- חיפוש האם נמצא במחסנית ממויינת ---  
public static bool DoseExist(Stack<int> stk, int x)  
{  
    Stack<int> sTemp = new Stack<int>();  
    bool found = false;  
  
    //--- חיפוש האיבר במחסנית תוך שמירת האיברים במחסנית עזר ---  
    while (!stk.IsEmpty() && stk.Top() < x)  
        sTemp.Push(stk.Pop());  
  
    //--- האם הלולאה נעצרה כי נמצא האיבר? ---  
    if (!stk.IsEmpty() && stk.Top() == x)  
        found = true;  
  
    //--- החזרת האיברים למחסנית ---  
    while (!sTemp.IsEmpty())  
        stk.Push(sTemp.Pop());  
  
    return found;  
}
```

שימוש בתור עזר

נמצא בתור?

```
//--- חיפוש האם נמצא בתור ----
public static bool IsExist(Queue<int> que, int x)
{
    Queue<int> qTemp = new Queue<int>();
    bool found = false;

    //--- חיפוש האיבר בתור תוך שמירת האיברים בתור עזר ---
    while (!que.IsEmpty())
    {
        if (que.Head() == x)
            found = true;
        qTemp.Insert(que.Remove());
    }

    //--- החזרת האיברים לתור ---
    while (!qTemp.IsEmpty())
        que.Insert(qTemp.Remove());

    return found;
}
```

שים לב שגם אם נמצא x, ממשיכים להעביר את איברי התור לתור העזר

חיפוש בתור מעגלי

```
//--- חיפוש האם נמצא בתור - תור מעגלי ----
public static bool DoseExist(Queue<int> que, int x)
{
    int deme = int.MinValue;    // הנחה איבר זה לא קיים בתור
    bool found = false;

    que.Insert(deme);
    while (que.Head() != deme)
    {
        if (que.Head() == x)
            found = true;
        que.Insert(que.Remove());
    }

    que.Remove();    // הוצאת איבר הדמה

    return found;
}
```

## נמצא בעץ?

```

//--- פעולה המחזירה אמת אם ערך קיים בעץ, אחרת שקר ---
public static bool IsExist(BinNode<int> bt, int x)
{
    if (bt == null)
        return false;
    if (bt.GetValue() == x)
        return true;
    return IsExist(bt.GetLeft(), x) || IsExist(bt.GetRight(), x);
}

```

יש להחזיר את הרשימה כי ייתכן ו- x הוא האיבר הראשון.

## מחיקה מרשימה

```

//--- פעולה המקבלת כפרמטר רשימה ומספר ומוחקת את המספר מהרשימה ---
//--- אם המספר לא קיים ברשימה, לא יתבצע דבר ---
public static Node<int> Remove(Node<int> lst, int x)
{
    //--- איתור המקום של המספר למחיקה ---
    Node<int> pos = lst, prev = null;
    while (pos != null && pos.GetValue() != x)
    {
        prev = pos;
        pos = pos.GetNext();
    }
    if (pos != null) // יצאנו מהלולאה כי נמצא x
    {
        if (prev == null) //--- מחיקת האיבר הראשון ברשימה ---
            lst = lst.GetNext();
        else //--- מחיקת איבר שאינו הראשון ---
            prev.SetNext(pos.GetNext());
    }
    return lst;
}

```

## שים לב!

- ברשימה סטטית (לא חלק ממחלקה המגדירה אוסף), הרשימה (המזמנת) lst והפרמטר lst הם שתי הפניות שונות. שינוי ב- lst של הפעולה אינו משפיע על lst של התכנית, ולכן:
- מותר לרוץ על הרשימה עם lst (אין צורך בהפניה מיוחדת)
  - אם מוחקים בתחילת הרשימה, יש להחזיר את ההפניה החדשה לתחילת הרשימה ( lst של התכנית לא יודע שנמחק האיבר הראשון)

ברשימה שהיא תכונה של המחלקה (רשימת הספרים בחנות ספרים, רשימת הקלפים בחפיסת קלפים וכד'), ההפניה היא הקישור היחיד לתחילת הרשימה, ולכן:

- הריצה על הרשימה תיעשה באמצעות הפנית עזר.

- אין צורך להחזיר רשימה כשמוחקים את האיבר הראשון או כשמוסיפים איבר בהתחלה, העדכון נעשה ממילא על `this.lst`.
- צמצום רשימה  
צמצום מחסנית (שכפול מחסנית)  
צמצום תור (שכפול התור באמצעות תור עזר או באמצעות תור מעגלי)

להלן ממשק חלקי של המחלקה `Item`:

<b>private int num</b> <b>private int amount</b>	מספר שלם המציין מספר ברשימה מספר מופעיו של המספר ברשימה
<b>public Item (int num, int amount)</b>	פעולה בונה היוצרת עצם מסוג <code>Item</code> שתכונותיו הן מספר שלם המופיע ברשימה ומספר מופעיו ברשימה.

ניתן להניח את קיומן של פעולות `Get` ו-`Set` לכל תכונה.

נתון מבנה נתונים (מסוג רשימה, מחסנית או תור) של מספרים שלמים `lst` שיש בהם מספרים החוזרים על עצמם יותר מפעם אחת.

כתוב פעולה המקבלת כפרמטר מבנה נתונים כזה של מספרים שלמים ומחזירה רשימה מצומצמת שבה עבור כל מספר המופיע במבנה הנתונים המקורי יופיע איבר אחד מסוג `Item` שערכיו הם המספר ומספריו מופעיו במבנה הנתונים. אף מספר לא יופיע יותר מפעם אחת ברשימה החדשה. סדר האיברים ברשימה החדשה יהיה כסדר הופעת המופע הראשון במבנה הנתונים המקורי.

אם אתה משתמש בפעולות עזר, כתוב את הפעולה.

שים לב! אין להשתמש במבנה נתונים אחר מעבר לרשימה ול-`Item`.

לדוגמה, עבור מבנה הנתונים הבא :  
`lst: [3, 2, 1, 5, 3, 3, 8, 9, 7, 1, 7, 5, 5, 3]`  
 תוחזר הרשימה הבאה :  
`lst1: [(3,4), (2,1), (1,2), (5,3), (8,1), (9,1), (7,2)]`

#### צמצום רשימה (`lst`)

1. צור רשימה מסוג `Item` ← `lst1`

2. עבור כל איבר `x` ברשימה `lst`

2.1 אם הוא עדיין לא מופיע ברשימה `lst1` // `lst1.GetValue().GetNum() == x`

2.1.1 ספור כמה פעמים הוא מופיע ב-`lst` ← `count`

2.1.2 צור איבר מסוג `Item` עם `x` ו-`count` והכנס אותו לסופה של הרשימה `lst1`

2.2 קדם את `lst` לאיבר הבא

4. חזר את הרשימה `lst1`

#### יעילות הפעולה : $O(n^2)$

מעבר על  $n$  האיברים ברשימת המספרים `lst`, ועבור כל איבר מבצעים :

א. חיפוש ברשימה החדשה שנבנית  $O(n)$

ב. ספירת מספר מופעיו ברשימה `lst`  $O(n)$

ג. הוספת האיבר לרשימת העזר  $O(1)$

ד. יצירת איבר והכנסתו לרשימה החדשה  $O(1)$

סה"כ :  $f(n) = n(2n + 2) \Rightarrow 2n^2 + 2n \Rightarrow O(n^2)$

**צמצום מחסנית (stk)**

1.  $lst \leftarrow \text{Item מסוג}$
2.  $sCopy \leftarrow$  שכפל את המחסנית
3. כל עוד המחסנית sCopy לא ריקה (עבור כל איבר x במחסנית sCopy)
  - 3.1.  $x \leftarrow$  שלוף איבר מראש המחסנית
  - 3.2.  $count \leftarrow 1$
  - 3.3. עבור כל איבר y במחסנית
    - 3.3.1. אם הוא שווה ל-x  $count++$
    - 3.3.2. אחרת - הוסף את y ל-sTemp
  - 3.4. צור איבר מסוג Item עם x ו-count והכנס אותו לסופה של הרשימה lst
  - 3.5. החזר (את האיברים שנותרו) ממחסנית העזר ל-sCopy
4. החזר את הרשימה lst

**שכפול מחסנית (stk)**

1. צור 2 מחסניות sTemp ו-sCopy
2. העבר את כל איברי stk ל-sTemp
3. על עוד sTemp אינה ריקה
  - 3.1. הוצא מהמחסנית  $x \leftarrow$
  - 3.2. הכנס אותו ל-2 המחסניות stk ו-sCopy
  4. החזר את sCopy

**יעילות הפעולה:  $O(n^2)$** 

- שיכפול המחסנית ( $2n$  צעדים)  $O(n)$
- מעבר על n האיברים במחסנית המספרים stk, ועבור כל איבר מבצעים:
- א. ספירת מספר מופעיו במחסנית stk  $2n$
  - ג. הוספת האיבר למחסנית העזר 1
  - ד. יצירת איבר והכנסתו לרשימה החדשה  $O(1)$
- סה"כ:  $f(n) = O(n) + n(2n + 2) \Rightarrow 2n^2 + 3n \Rightarrow O(n^2)$

**צמצום תור (que)**

אותו אלגוריתם כמו צמצום מחסנית תוך שימוש בתורי עזר. שכפול תור - באמצעות תור עזר או באמצעות תור מעגלי.

**מתי משכפלים מחסנית / תור?**

רק במקרים בהם סריקת המחסנית / תור אינם שומרים על הסדר או מחייבים מחיקה. למשל: צמצום איברי המחסנית / התור הכפולים לאיברים המכילים שתי תכונות: ערך המחסנית / התור ומספר מופעיו.



- **עצים**  
 תבניות אלגוריתמיות  
 סריקת עץ לפי רמות  
 סכום רמות בעץ  
 הורים וצאצאים בעץ

(דפים: [תבניות אלגוריתמיות](#), [אוסף פעולות](#), [תבניות בעצים](#))

### כאלים לטיפול בעץ בינארי:

- אין לפנות למידע או לבניו של צומת לפני שווידאנו שהצומת לא null
- אין לפנות למידע בבניו של הצומת לפני שווידאנו שיש לו בנים.
- כאשר ערך הצומת הינו מבנה נתונים (מערך, רשימה, מחסנית, תור או אפילו עץ אחר), טפל תחילה הערך הצומת לפי מבנה הנתונים שלו ורק אחר כך המשך בסריקה תחילה על צד שמאל ואחר כך על צד ימין, אלא אם כן נאמר אחרת.
- כאשר בודקים צומת בעץ, יש לבדוק את כל 5 המקרים:
  - א. `bt == null` הוא עץ ריק
  - ב. `bt` הוא עלה
  - ג. יש ל-`bt` בן יחיד והוא בן שמאלי
  - ד. יש ל-`bt` בן יחיד והוא בן ימני
  - ה. יש ל-`bt` שני בנים
- שים לב! חובה להתחיל בסעיף א' - עץ ריק. סדר שאר הסעיפים תלוי בשאלה.
  - למרות חשיבותה, הפעולה **עלה?** (`bt`) אינה קיימת ויש לממש אותה

### תבניות אלגוריתמיות:

- האם כל הצמתים מקיימים תנאי ? (`bt`)**
1. אם (`bt == null`) החזר אמת
  2. אם הצומת הנוכחי אינו מקיים את התנאי החזר שקר
  3. החזר צד שמאל מקיים את התנאי וגם צד ימין מקיים את התנאי

#### לדוגמה:

עץ ממויין הוא עץ שבו ערך כל צומת קטן מהערך של כל אחד מבניו וגם בן שמאלי קטן מערך בן ימני.

#### עץ ממויין ? (`bt`)

1. אם (`bt == null` או **עלה?** (`bt`)) החזר אמת
2. אם (יש ל-`bt` בן שמאלי, וגם ערך הבן השמאלי קטן מערך השורש) החזר שקר
3. אם (יש ל-`bt` בן ימני, וגם ערך הבן הימני קטן מערך השורש) החזר שקר
4. אם (יש ל-`bt` בן שמאלי וגם יש ל-`bt` בן ימני וגם ערך הבן השמאלי גדול מערך הבן הימני) החזר שקר
5. החזר עץ ממויין (בן שמאלי) וגם עץ ממויין (בן ימני) שים לב! אם אחד הצדדים יחזיר שקר, יהיה הערך הביטוי שקר.

האם לפחות צומת אחד מקיים תנאי (bt) (או האם חלק מהעץ מקיים את התנאי)

1. אם (bt == null) החזר שקר
2. אם (הצומת הנוכחי מקיים את התנאי) החזר אמת
3. החזר צד שמאל מקיים את התנאי או צד ימין מקיים את התנאי

דוגמאות:

- \* האם נמצא (bt, x)
- \* עץ בעל שרשרת x הוא עץ שקיים בו מסלול המתחיל בשורש ומסתיים באחד העלים וערך כל צמתי המסלול הוא x

עץ-בעל-שרשרת-x (bt, x)

1. אם (bt == null) החזר שקר
2. אם (עלה? (bt) וגם ערך העלה הוא x) החזר אמת
3. אם (ערך הצומת שונה מ-x) החזר שקר
4. החזר עץ-בעל-שרשרת-x (בן שמאלי) או עץ-בעל-שרשרת-x (בן ימני)

סריקה לפי רמות (bt)

1. צור תור של עצים ← que תור מסוג Queue<BinNode<int>>
2. הכנס את bt לתור
3. כל עוד התור לא ריק,
  - 3.1 הוצא מהתור ← bt
  - 3.2 הצג את ערך השורש של bt
  - 3.3 אם יש ל-bt בן שמאלי, הכנס אותו לתור
  - 3.4 אם יש ל-bt בן ימני, הכנס אותו לתור

סכום צמתים ברמה (bt, level)

1. אם (bt == null) החזר 0
2. אם (level == 0) החזר את ערך הצומת
3. החזר סכום-צמתים-ברמה (בן שמאלי של bt, level-1) + סכום-צמתים-ברמה (בן ימני של bt, level-1)

רמת-הצומת (bt, t) /\* הפעולה מחזירה את מספר הרמה של צומת t אם הוא בעץ bt אחרת יוחזר -1 \*/

1. אם (bt == null) החזר -1
2. אם (bt == t) החזר 0
3. רמת-הצומת (t, בן שמאלי של bt) ← left
4. אם (left >= 0) החזר left+1 // נמצא הצומת בצד שמאל
5. רמת-הצומת (t, בן ימני של bt) ← right
6. אם (right >= 0) החזר right+1 // נמצא הצומת בצד ימין
7. החזר -1 // הצומת לא נמצא בצד זה

הורה של צומת: פעולה המקבלת הפנייה לשורש העץ והפנייה לצומת בעץ שאינו השורש ומחזירה הפניה להורה של הצומת.

הורה (bt, t)

1. אם  $bt == \text{null}$  או בן שמאלי של bt שווה ל-t או בן ימני של bt שווה ל-t החזר את bt

2. הורה (t, בן-שמאלי של bt)  $\leftarrow \text{left}$

3. אם  $(\text{left} != \text{null})$  // ההורה נמצא בצד שמאל החזר את left

4. החזר הורה (t, בן-ימני של bt)

### חיפוש באי-סדר:

- האם צאצא? (bt, x, y) מוחזר אמת אם קיימים בעץ bt שני צמתים שערכיהם x ו-y והאחד צאצא של האחר, ושקר אחרת.
- צאצא? (bt, t1, t2) - t1 הוא צאצא של t2 אם קיים מסלול מצומת t2 לצומת t1. הנחה: t1 ו-t2 הם הפניות לצמתים בעץ bt.
- סבא-רבא (bt, t1, t2) חיפוש הצומת הקרוב ביותר לעלים t1 ו-t2 המהווה אב קדמון של שניהם

האם-צאצא? (bt, x, y) // x ו-y הם ערכי הצמתים

1. אם  $(bt == \text{null})$  החזר שקר

2. אם (ערך הצומת הוא x)

החזר נמצא-בעץ? (y, בן-שמאלי של bt) או נמצא-בעץ? (y, בן-ימני של bt)

3. אם (ערך הצומת הוא y)

החזר נמצא-בעץ? (x, בן-שמאלי של bt) או נמצא-בעץ? (x, בן-ימני של bt)

4. החזר האם-צאצא? (bt.GetLeft(), x, y) או האם-צאצא? (bt.GetRight(), x, y)

צאצא? (bt, t1, t2) // t1 ו-t2 הם הפניות לשני צמתים. מוחזר אמת אם t1 צאצא של t2

1. אם  $(t2 == \text{null})$

החזר שקר // t1 לא נמצא במסלול זה

2. אם  $(t2 == t1)$

החזר אמת

3. החזר צאצא? (t2, בן-שמאלי של bt, t1) או צאצא? (t2, בן-ימני של bt, t1)

(\*) הפעולה בושקת האם t2 הוא צאצא של t1.

כדי לדעת האם קיים יחס של צאצאים בין t1 ו-t2

נפעיל את הפעולה: החזר צאצא? (bt, t1, t2) או צאצא? (bt, t2, t1)

בגרות תש"ס, 2000 :

נגדיר **סבא-רבא** של שני עלים שונים בעץ בינארי הוא **האב הקדמון** המשותף לשני העלים כלשהם בעץ.  
 לפיכך הפעולה: **סבא-רבא** (עלה1, עלה2, עץ) : הפעולה מחזירה את הצומת שהוא **סבא-רבא** של שני העלים. הנחה, שני העלים נמצאים בעץ.  
 יש לממש את הפעולה תוך שימוש בפעולות: **האם-צאצא?** (צומת1, צומת2, עץ)  
**הורה** (צומת, עץ)

**רעיון הפתרון:**  $t1$  ו- $t2$  הם הפניות לשני עלים בעץ  $bt$ .  
 נבחר באחד העלים, נניח  $t1$  ונבדוק אם  $t2$  הוא צאצא של אבא של  $t1$   
 אם כן, נחזיר את ההורה.  
 אחרת, נבדוק באופן רקורסיבי את ההורה של האבא.  
 הרקורסיה תעצור כאשר נמצא צומת שהוא אב-קדמון של שני העלים או כאשר נגיד לשורש  $bt$

**סבא-רבא** ( $bt, t1, t2$ )  
 1. **צאצא?** ( $bt, t1, t2$ ) // אם  $t2$  הוא צאצא של  $t2$ , נמצא האב הקדמון של שני העלים  
 החזר את  $t1$   
 2. החזר **סבא-רבא** ( $t2$ , **הורה** ( $bt, t1$ ),  $bt$ )