בעיית בניין וכדור זכוכית.

יש לנו בניין של N קומות ו- 2 כדורי זכוכית.צריך למצוא את הקומה הכי נמוכה שאם נזרק את הכדור הוא ישבר.

- חיפוש שלם(כאשר יש כדור 1) - אופציה הכי פשוטה לעבור על כל הקומות אינדוקטיבית(לולאת for) אם נשבר מצאנו אם לא נעבור לקומה הבאה.בסיבוכיות - $O(n)$.
- כאשר יש יותר מכדור אחד אפשר לחלק את הבניין לחצי אם הכדור נשבר נעבור עם הכדור השני בחיפוש שלם מ- $1 \to \frac{n}{2}$, אם לא נשבר נזרוק מקומה ה- n, ועם הכדור השני נעבור בחיפוש שלם מ- $\frac{n}{2} \to n$, סה"כ פעולות $O\left(\frac{n}{2}+2\right)$.

כך אפשר לחלק את הבניין ל- $\frac{n}{2}, \frac{n}{3}$ חלקים וכך הלאה...אך זה לא נכון להגיד שנחלק ל- n חלקים את הבניין כי בעצם נחזור לסיבוכיות של $O(n)$.

לכן השאלה היא לכמה חלקים לחלק את הבניין כך שיהיה לנו מס' פעולות מינמ'?

תשובה: כל פעם עושים $f(x)=\frac{n}{x}+x$.

כלומר נחלק את הבניין ל - $\sqrt{n}$ חלקים.

ומס' פעולות $2\sqrt{n}$.

2	$O\ \sqrt{n}$	$2\sqrt{n}$

$$\min_{1\le x\le n}\left(\frac{n}{x}+x\right)$$

$$a^2+b^2\ge 2ab$$

$$a+b\ge 2\sqrt{ab} \longrightarrow \frac{n}{x}+x\ge 2\sqrt{n}$$

$$\min_{\substack{1\le x\le n\\ x\in\mathbb{R}}}\left(\frac{n}{x}+x\right)=2\sqrt{n}$$

יש להתייחס למצב כאשר יש ∞ כדורים אז אפשר להשתמש בחיפוש בינארי, בסיבוכיות $O(\log n)$.

אך מצב זה לא אפשרי בחיים האמיתיים.

PesodCode:

```
int Drop Ball(int X̄ ){// |X̄|=n
      break1 ← false
      koma ← √2n
      i ← 1
      while(koma <n+1){
            if(break1 = true )
                  return (koma -√2n) + BS(koma -√2n, koma)
            else
                  koma ← (koma + √2n)-i
                  i++
}
BS(start,end){
   break2 ← false
   for i←start to end
     if(break2(i) = true)
return i
}
```

אלגוריתם של משחק המספרים.

- אסטרטגיה ראשונה היא לחשב את הסכום של האיברים הזוגיים ואי זוגיים ואת הסכום הכי גבוהה מאינדקס שלו להתחיל,השחקן הראשון מכריח את היריב לקחת כל פעם את הכי נמוך,זה אלגוריתם של מקס' לא להפסיד.
- האלגוריתם השני מבטיח לסיים בתוצאה טובה יותר,יש ליצור עץ אבל רק ליצור אותו זה $O\left(2^n\right)$ לכן,נתכנת דינמי ע"י כך שנמלא מטריצה.קודם נמלא באלכסון את המערך הנתון ואח"כ נמלא עפ"י התנאי: $F[i,j] = \max\left(a_i - F[i+1,j], a_j - F[i,j-1]\right)$ -ואז הרווח נמצא בפינה הימנית העליונה.

נתבונן ב- 3 קודקודים ראשונים של עץ החישובים של חיפוש שלם של המשחק. כדי למצוא פתרון אנו נחשוב בצורה רקורסיבית ונניח שיש פתרון.כדי לדעת רווח של שחקן בקודקוד $[1,n]$,אנחנו מניחים שיש לנו רווח בקודקודים $[2,n]$,ובי- $[1,n-1]$.

את המימוש של האלגוריתם נעשה בצורה אינדוקטיבית.

PesodCode:

אתחול מטריצה(אתחול אינדוקציה)

```
for i←1 to n
    F[i,i] = a_i
```

```
for i←(n-1) to 1
    for j←(i+1) to n
        F[i,j] = max(a_i – F[i+1,j], a_j – F[i,j-1])
```

	1	2	3	4	5	6
1	1	2	4	3	0	6
2		3	3	-2	5	1
3			6	5	4	2
4				1	2	4
5					3	3
6						6

כדי לקבל/למצוא את המסלול נשאל את השאלה הבאה:

```
while(i<j)
    if(a[i]-F[i+1,j]>a[j]-F[i,j-1])
        syso("my choose is:"+a[i])
        i++
    else
        syso("my choose is:"+a[j])
        j--
```

שחקן 1	שחקן 2
$a_6=6$	$a_5=3$
$a_1=1$	$a_2=3$
$a_3=6$	$a_4=1$

מי שנכון לוקחים.בסיבוכיות $O(n-1)$.

אז סה"כ הסיבוכיות $O\left(n^3+n-1\right) \approx O\left(n^2\right)$

דוגמא:

a_1	a_2	a_3	a_4	a_5	a_6
1	3	6	1	3	6

$F(3,6) = \max\left(a_3 - F(4,6), a_6 - F(3,5)\right)$

בעיית תת הריבוע הגדול ביותר של אחדות.

כקלט אנו מקבלים מטריצה וכפלט מחזירים גודל של תת הריבוע הגדול ביותר,ומיקום קודקוד ימני עליון,נפתור בעיה זו ע"י תכנות דינאמי ב- $O\left(n^2\right)$ בעל מקדם 1.

נעתיק את השורה והעמודה הראשונה של המטריצה המקורית ואת כל האפסים שלה ונמלא את המטריצה ע"י החוקיות הבאה:

$f = \min\left(x,y,z\right)+1$.

כמובן יש קודם לבדוק האם $f\left[i,j\right] \neq 0$.

M

4	0	0	1	0	0
3	1	1	1	1	0
2	0	1	1	1	1
1	1	1	1	1	0
0	1	1	0	0	0
	0	**1**	**2**	**3**	**4**

S

4	0	0	1	0	0
3	1	1	2	3	0
2	0	1	2	2	1
1	1	2	1	1	0
0	1	1	0	0	0
	0	**1**	**2**	**3**	**4**

PesodCode:

```
subMatrixOf1(int [][] M){
for i ←0 to n //העתקת הדפנות
      S[0,i] = M[0,i]
      S[i,0] = M[i,0]
for i ←1 to n // מילוי המטריצה
      for j ←1 to n
            if(M[i,j] ≠0)
                  S[i][j] = min(S[i-1][j],S[i-1][j-1],S[i][j-1])+1
            if(S[i,j]>max)
                  max = S[i][j]
                  imax = i,jmax = j
syso("+max+",["+imax+","+jmax+"]"
}
```

<u>בעיית המטוס – המסלול הקצר ביותר.</u>

ישנן שתי שיטות לפתרון:

1. ע"י חיפוש שלם אך בסיבוכיות של $O\left(2^n\right)$ שהיא בסדר למסלול קצר.
2. במקרה זה נבצע חישוב ביניים ונראה מה המסלול העדיף בכול תנועה.סיבוכיות המילוי של המטריצה היא - $O\left(n\times m\right)$,כיוון שאנו כל פעם מתקדמים במסלול אחד, אך עוברים על כל הנקודות של המטריצה.
סיבוכיות החזרה היא - $O\left(n+m\right)$, כי בכול מסלול חזרה שנבחר אנו נעבור ב- n,m צלעות.

נפתור זאת ע"י מטריצה שכל איבר בה הוא מסוג Node,המכיל 3 שדות – entry,x,y.

- נמלא את השורה התחתונה והעמודה השמאלית.
- נמלא את שאר הטבלה ע"י התנאי הנ"ל,התשובה תהיה מס' המסלולים האופטימאליים בפינה הימנית העליונה.
- על מנת למצוא את המסלול,נחזור מהפינה הימנית העליונה ע"י נכונות התנאי.(עפ"י התוצאה הנמוכה ביותר).

PesodCode:

```
class Node {
      int _x,_y,_entry
      public Node(int x,int y)
            _x = x
            _y = y
            _entry= 0
}
priceOfCheepestPaths(){//n = matrix.length
   matrix[0,0]._entry = 0
   for i←1 to n // O(max(n,m))
         M[0,i]._entry ←M[0,i-1]._entry + M[0,i-1]._x;
         M[i,0]._entry←M[i-1,0]._entry + M[i-1,0]._y
  //מילוי המטריצה
   for i←1 to n //O(n)
         for j  1 to m //O(m)
                a←M[i-1,j]._entry + M[i-1,j]._x
                b←M[i,j-1]._entry + M[i,j-1]._y
                M[i,j]._entry←Math.min(a, b)
return matrix[n-1,n-1]._entry
}
String printPath() //מחזיר את המסלול הקצר ביותר
      String S ← "("+n+","+n+")"
      i ← n-1
      j ← n-1
      while(i>0 && j>0){
            a = matrix[i][j-1]._entiry;
            b = matrix[i-1][j]._entiry;
            if(a>b)
                  S = S+"-->["+(i-1)+","+j+"]";
                  i--;
            else
                  S = S+"-->["+i+","+(j-1)+"]";
                  j--;
      return S+"-->[0,0]";
```

אם מבקשים את מס' המסלולים המינמ' או המקס' – בעצם נוסיף ל - Node עוד משתנה `int dis` שהוא בעצם אורך המסלול שהוא דיפולטיבי(שווה ל – 1), ותנאי חזרה לפי בקשת השאלה מינמ' או מקס'.

```
if(a = b)
   matrix[i][j].dis ← matrix[i-1][j].dis + matrix[i][j-1].dis
else if(a < b)
   matrix[i][j].dis ← matrix[i-1][j].dis
else
   matrix[i][j].dis ← matrix[i][j-1].dis
```

סה"כ הסיבוכיות היא - $O\left(n\times m\right)+O\left(n+m\right)\approx O\left(n\times m\right)$.

בעייה – MinMax.

נשווה את שני האברים הראשונים ונחלק אותם ל – min,max.

נחלק את המערך לזוגות וכל פעם נבדוק זוג - זוג $a[i] > a[i+1]$

ואם גדול מהמקס' נחליף למקס' $a[i] > \max \rightarrow \max = a[i]$,

וכך גם ההפך למינמ' - $a[i+1] < \min \rightarrow \min = a[i+1]$.

במקרה שהמערך ממוין אז זה $n-1$, במקרה שלא אז זה $2n-2$ השוואות

לכן לא חשוב מה הסיבוכיות היא - $O(n)$ - $\frac{3}{2}n$ השוואות.

PesodCode:

```
min  ←  Math.min(arr[0],arr[1])
mix  ←  Math.max(arr[0],arr[1])
for i  ←  2 to arr.length-1//i=+2
    if(arr[i]>arr[i+1])
      if(arr[i]>max) max  ←  arr[i]
      if(arr[i+1]<min) min  ←  arr[i+1]
    else
      if(arr[i+1]>max) mix  ←  arr[i+1]
      if(arr[i]<min) min  ←  arr[i]
return {min,max}
```

בעייה – Max1Max2.

קיימים 2 פתרונות אינדוקטיבי והשוואות.

1. אינדוקטיבי – נקבל מערך ונהפוך אותו לרשימה מקושרת שבה נבדוק בין צמודים,כל איבר בה יהיה מסוג Node המורכב מ- 2 שדות: int num,next,כל אובייקט מחזיק מס' שהוא המקס' שלו וגם מחסנית שבה יש אברים פוטנציאלים להיות מקס' 2.
נחלק לזוגות ונבדוק מי יותר גדול,את הקטן מבניהם נכניס למחסנית של הגדול ונמחק אותו מהרשימה ובסופו של דבר יהיה לנו איבר 1 ברשימה שהוא Max1.
נעשה חיפוש שלם על המחסנית שלו(Max1) ונמצא את Max2 .

סיבוכיות – הכנסת אברי המערך לרשימה היא $O(n)$.

השוואת האברים היא - $O(\log n)$.

לכן $O(n) \approx O(n) + O(\log n)$.

PesodCode: Induction

```
List list1;
for i←0 to arr.length
      list1.add(arr[i])
while(list1.size()>1)
      if(list1.get[i]>list.get[i+1])
            list1[i].s1.push[i+1].num
            remove list[i+1]
      else
            list1[i+1].s1.push[i].num
            remove list[i]
      i++
      if(i==list.size()|| i==list.size()-1)
      i = 0;
Max1 ←list[0].num
syso("Max1 is"+Max1)
Max2 ←list[0].s1.pop
while(!list[0].s1.isEmpty){
      if(list[0].s1.pop>Max2)
            Max2 ←list[0].s1.pop
}
syso("Max2 is:"+Max2)
```

PesodCode: Recursion

```
int maxMax2(int low, int high){
   if (low < high)
      index←0
      middle←(low + high)/2
      i ← maxMax2(low, middle)
      j ← maxMax2(middle+1, high)
      if(arr[i].num>arr[j].num)
         arr[i].st.push(arr[j].num)
         index ← i
      else
         arr[j].st.push(arr[i].num)
         index ← j

      return index
   else
      return low

}
```

2. רקורסיבי – גם כאן הופכים את המערך לרשימה מקושרת ולכל איבר בה מסוג Node שיש לו מחסנית ההבדל הוא רק בצורת החילוק.

3. נעביר את הדוגמא של min/max לשלנו ניקח שני אברים ראשוניים a_i, a_{i-1}, ונבדוק את היחס בניהם $a_i > a_{i-1}$.

$$\begin{array}{ccc} \boxed{\max 2} & < & \boxed{\max 1} \\ & \nearrow & \wedge \\ \boxed{a_{i-1}} & < & \boxed{a_i} \end{array}$$

ונבצע 3 השוואות לזוג מספרים.

ולכן יש לנו כאן $\frac{3}{2}n$ השוואות שזו סיבוכיות של $O(n)$.

PesodCode:

```
if(arr[0]>arr[1])
   max1 ← arr[0]
   max2 ← arr[1]
else max1 ← arr[1],max2 ← arr[0]
for i ← 2 to arr.length  //i=+2
    if(arr[i]>arr[i-1])
       if(arr[i]>max1)
          if(arr[i-1]>max1)
             max1 ← arr[i]
             max2 ← arr[i-1]
          else
             max2 ← max1
             max1 ← arr[i]
       else//arr[i]<max1
           if(arr[i]>max2)
              max2 ← arr[i]
   else//arr[i]<arr[i-1]
       if(arr[i]>max1)
          if(arr[i-1]>max1)
             max1 ← arr[i-1]
             max2 ← arr[i]
          else
             max2 ← max1
             max1 ← arr[i-1]
       else//arr[i-1]<max1
           if(arr[i-1]>max2)
              max2 ← arr[i-1]
return max1,max2
```

חישוב חזקה.

השיטה הפרימיטיבית היא ב - $O(n)$, $X^n = x \cdot x \cdot x \cdot \ldots \cdot x$.

העיקרון לשיטה החכמה הוא:

$$X^n = \left(X^{\frac{n}{2}}\right)^2 \text{ or } X \cdot \left(X^{\frac{n}{2}}\right)^2 \ldots$$

for example:

$$X^8 = \left(X^4\right)^2 = \left(\left(X^2\right)^2\right)^2$$

ישנן 2 דרכים רקורסיבי ואינדוקטיבי.

PesodCode:

```
/**
 * O(logn)
 * Recursion
 */
double powerRecursion(double x, int n){
      double result←1;
      if(n == 0){// exit of recursion
            return 1;
      }
      else if(n%2 == 0){// n is even
            return powerRecursion(x*x, n/2);
      }
      else{// n is odd
            return x*powerRecursion(x*x, (n - 1)/2);
      }
}
```

```
/**
 * O(logn)
 * Inductive
 */
double powerInductive(double x , int n ){
      double result←1;
      while(n!=0)
            if(n%2 == 1) result←result*x;
            x←x*x;
            n←n/2;
return result;
}
```

בעיית החציון.

כאשר נבדוק את המערך ה- 64 ההסתברות שהחציון הוא המס' ה –max היא 99.999% $\left[1-\frac{1}{2^{64}}\right]$.

במקרה שלנו בחרנו במספר K=64.

ישנו מצב מעניין שההסתברות לטעות במידה והמערך רנדומאלי היא $\frac{1}{2^{64}}$.

יותר קטן מטווח הטעות של המחשב.

הסיבוכיות - $O(k)$.

PesodCode:

```
int max(int arr[],int k){
      max ← arr[0]
      for i ← 1 to k
          if(arr[i]>max)
             max ← arr[i]
return max;
}
int HELF(int []arr){
      Arrays.sort(arr);
      HF ← arr[arr.length/2]
      return HF;
}
```

k=	הסתברות להצלחה
1	1 – 1/2 = 1/2
2	1 – (1/2*1/2) = 3/4
k	$1-1/2^k$
64	$1-1/2^{64}$

חישוב עצרת באינדוקציה.

ברקורסיבי חישוב עצרת עולה לנו $O(2^n)$, לכן עדיף להשתמש האינדוקציה שהסיבוכיות של היא $O(n)$.

PesodCode:

```
if(n==0)return 1
ans ← 1
for i ← 2 to n
      ans ← ans*i
}
return ans
```

חישוב מספרי פיבונצ'י.

בעצם כדי להגיע לסיבוכיות זו אנו לוקחים

מטריצה $A=\begin{pmatrix}0 & 1\\ 1 & 1\end{pmatrix}$, וכשנכפיל אותה

ב- n פעמים נקבל $\begin{pmatrix}F_{n-2} & F_{n-1}\\ F_{n-1} & F_n\end{pmatrix}$, שזוהי

בעצם נוסחת פיבונצ'י.

נוכיח באינדוקציה – הנחת האינדוקציה :

$$\begin{pmatrix}0 & 1\\ 1 & 1\end{pmatrix}^n=\begin{pmatrix}F_{n-2} & F_{n-1}\\ F_{n-1} & F_n\end{pmatrix}$$

$$\begin{pmatrix}0 & 1\\ 1 & 1\end{pmatrix}^{n+1}=\begin{pmatrix}F_{n-1} & F_n\\ F_n & F_{n+1}\end{pmatrix}$$

PesodCode:

```
/**
 * Recursive
 * O[log(n)]
 */
public static int FibFibRecursion(int n){
      int[][]M ← {{0,1},{1,1}};
      int[][]ans ← FibRecursion(m,n);
      return ans[0][0];
}
public static int[][] FibRecursion(int M [][], int n){
      int [][] result ← {{1,0},{0,1}};
      if (n == 0){// exit of recursion
            return result;
      }
      else if (n%2 == 0){// n is even
            return FibRecursion(M X M, n/2);
      }
      else{// n is odd
          return M X FibRecursion(M X M,(n-1)/2);
      }
}
```

שלב האינדוקציה:

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n+1} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} F_{n-1} & F_{n-2}+F_{n-1} \\ F_n & F_{n-1}+F_n \end{pmatrix}$$

$$= \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix}$$

מ.ש.ל.

<u>סיבוכיות</u> - $O(\log n)$.

PesodCode:

```
/**
* Inductive
* O[log(n)]
*/
public static int Fib(int n ){
      int[][]M = {{0,1},{1,1}};
      int [][]result = new int[2][2];
      result[0][0] = 1;
      result[0][1] = 0;
      result[1][0] = 0;
      result[1][1] = 1;
      while (n!=0)
            if(n%2 != 0)
                  result = result X M

            M = M X M
            n = n/2;

return result[0][0];
}
```

<u>בעיית החניון.</u>

נתון לנו מערך מעגלי של מכוניות איך נספור את כמות המכוניות במערך?
נתחיל לספור (נשמור את האינדקס שממנו התחלנו),עד שהגענו לאותו ערך(נשמור את מס' הצעדים).
נשנה את אותו ערך לסימון שמוכר לנו.ואז נחזור חזרה כמס' הצעדים שלנו.
אם הגענו לערך שממנו התחלנו נחזור חזרה מאיפה שהפסקנו ונמשיך לספור,אם הגענו לסימון שלנו מכאן שעשינו הקפה שלמה.

הסיבוכיות - $O(n)$.

PesodCode:

```
void robot(int start,int arr[]){
      count←1
      Mark←-1
      for i←start+1 to arr.length
            if(arr[i]==arr[start])
                  arr[start] ←Mark;
            if(arr[i]==arr[i-(count%arr.length)])
                  break
            else
                  count++;
            if(i>arr.length-1)
                  start←Mark
System.out.println(count)
PrintArry(arr,count)
}
```

LCS – המחרוזת המשותפת הארוכה ביותר.

בעצם אנחנו מחפשים את מידת ההתאמה בין מחרוזות.ישנן 3 שיטות חיפוש שלם,אלגוריתם חמדני,ותכנות דינאמי.

1. חיפוש שלם – כאן נסרוק את כל תתי המחרוזות בראשונה ובשנייה ומשווה כל פעם שיש התאמה.אבל יש מחרוזות בעלות n אברים ולהן יש 2^n תתי קבוצות,(כי לאות אחת יש 2 תתי קבוצות האות עצמה וקבוצה ריקה ולכן לכל אות שנוסיף תיצור פי 2 תתי קבוצות),ולכן נגיע לסיבוכיות של - $O\left(2^{n+m}\cdot\min(n,m)\right)$,אלגוריתם זה טוב למחרוזות באורך 32,לא יותר.
2. אלגוריתם חמדני – כאן נעבור על המחרוזת הקצרה ביותר וניצור מערך שבו כל אינדקס מסמל את מה – ABC ובכול תא כזה יופיע מס' הפעמים שאותה אות מופיעה במחרוזות.סיבוכיות של - $O(n\cdot m)+O(n)+O(n+m)$,במקרה הגרוע ביותר.

```
String greedy(String X, String Y){
      String res←"";
      int ind←0, index←0, beg←0;
      while (ind < X.length()){
            index←findElement(beg,Y,X.charAt(ind));
            if(index != -1){
                  res←res + X.charAt(ind);
                  beg←index+1;
            }
            ind←ind + 1;
      }
      return res;
}
```

דוגמא להרצת האלגוריתם: $\bar{X}$

0	0	0	0
a	b	c	d

a	b	c	b	d	a	b
1	2	3	4	5	6	7

$help[x(1)] \leftarrow help[x(1)]+1$

$help[x(2)] \leftarrow help[x(2)]+1$

$help[x(3)] \leftarrow help[x(3)]+1$

$help[x(4)] \leftarrow help[x(2)]+1$

$help[x(5)] \leftarrow help[x(5)]+1$

$help[x(6)] \leftarrow help[x(1)]+1$

$help[x(7)] \leftarrow help[x(4)]+1$

בממוצע החמדני לא יקטין את היעילות לכן נבחר שיטה אחרת.

3. תכנות דינאמי – נבנה מטריצה נמלא את השורה והעמודה הראשונה באפסים ונבצע השוואה בין האותיות עפ"י התנאים הבאים:
   - אם $a=b \rightarrow F[i,j]=F[i-1,j-1]+1$.
   - אם $a\neq b \rightarrow F[i,j]=\max\left(F[i-1,j],F[i,j-1]\right)$.

PesodCode:

בניית המטריצה:

```
// build matrix of numbers
int[][] buildMatrix(char[]X,char[]Y){// |X|=n,|Y|=m
// כי מוסיפים שורות אפסים n+1,m+1
      row← n+1
      col← m+1
      M[][] ←new int[row][col]
// מילוי שורה ראשונה באפסים
      for i←0 to row
         M[i][0]=0
// מילוי עמודה ראשונה באפסים
      for j←0 to col
         M[0][j]=0
// מילוי המטריצה עפ"י החוקיות
      for i←1 to row //O(n)
         for j←1 to col //O(m)
            if(X[i-1]==Y[j-1])
               M[i][j] ←M[i-1][j-1] + 1

            else
               M[i][j] ←Math.max(M[i-1][j],M[i][j-1])
      return M
}
```

הסיבוכיות היא - $O(n\cdot m)$.

PesodCode:

מחזיר את המחרוזת המשותפת הארוכה ביותר:

```
returns max common sequence length
int maxSeqLength(char[]X,char[]Y){// |X|=n,|Y|=m
      M[][]←buildMatrix(X,Y)
return M[n][m]
}
returns max common sequence
char[]maxSequence(char[]X,char[]Y)
      M[][]←buildMatrix(X, Y);
      seqLength←M[n-1][m-1]
      result[]←new char[seqLength]
      i←n-1
      j←m-1
      count←seqLength-1
      while(i>0&&j>0)
            if(X[i-1]==Y[j-1])
               result[count--]←X[i-1]
               i--
               j--
            else if(M[i][j]== M[i-1][j])
               i--
            else
               j--
return result
}
```

```
max seq len = 4
b, d, a, b
```

דוגמא למימוש האלגוריתם:

$$F\left("a","b"\right)=\begin{cases} a=b,1 \\ a\neq b,0 \end{cases}$$

		Y j	b j+1
X	i		
a	i+1		

			1	2	3	4	5	6
	F	**Y**	**b**	**d**	**c**	**a**	**b**	**a**
	X	0	0	0	0	0	0	0
1	**a**	0	0	0	0	1	1	1
2	**b**	0	1	1	1	1	2	2
3	**c**	0	1	1	2	2	2	2
4	**b**	0	1	1	2	2	3	3
5	**d**	0	1	2	2	2	3	3
6	**a**	0	1	2	2	3	3	4
7	**b**	0	1	2	2	3	4	4

when x(i)=y(j)		when x(i)!=y(j)	
2	2	2	3
2	F(2)+1=3	2	max(3,2)=3

אלגוריתם רקורסיבי למציאת כל תתי קבוצות בגודל K.

PesodCode:

```
Vector<String> combinations(String s, int k) {
    Vector<String> vec = new Vector<String>()
    combinations(s, "", k, vec)
    return vec
}
void combinations(String s,String prefix,int k,Vector<String>vec)
    if (s.length()<k) return;
    else if(k == 0)vec.add(prefix)
    else
        combinations(s.substring(1),prefix + s.charAt(0),k-1,vec)
        combinations(s.substring(1),prefix,k,vec)

}
```

LIS - חיפוש של תת המערך הממוין הארוך ביותר.

נפתור זאת ע"י תכנות דינאמי.

נבנה מטריצה בגודל $M\left[a.length\right]\left[a.length\right]$, ומערך d שהוא האלכסון של המטריצה.

נעתיק את האיבר הראשון מהמערך למטריצה ול - d, עוברים על כל איבר ב - a, ומחפשים מה אפשר לשפר באלכסון d, אחרי שמוצאים שיפור מכניסים אותו לאלכסון ומשמאלו מעתיקים את השורה שמתחתיו.

דוגמא לרצת האלגוריתם:

a = {1,100,101,2,3,4,5,6,7,8}

1	100
1	

1	100	101
1	100	
1		

1	100	101
1	2	
1		

1	2	3
1	2	
1		

1	2	3	4
1	2	3	
1	2		
1			

1	2	3	4	5
1	2	3	4	
1	2	3		
1	2			
1				

PesodCode:

מחזיר LIS – בסיבוכיות $O\left(n^2\right)$.

```
int[] LIS(int [] A){// |A|=n
      M[][] ←new int[n][n]
      d[] ←new int[n]
      M[0][0] ←A[0]
      d[0] ←A[0]
      end←0
      for i←1 to n //O(n)
      // d - מחזיר את האינדקס של האיבר מהמערך למערך
          index←binarySearchBetween(d,end,A(i))// O(log n)
          M[index][index] ←arr[i]
          for j←0 to index//O(n)
              M[index][j]←M[index-1][j]
          d[index] ←A(i);
          if(index>end) end++
      }
//copy O(n)
      int ans[]←new int[end+1];
      for j←0 to end
         ans[j]←M[end][j]
return ans
```

אם נרצה רק את הגודל נעשה חיפוש על המערך אבל המערך ממוין לכן נעשה חיפוש בינארי שזה $\log n$.

PesodCode:

מחזיר את האורך של LIS – בסיבוכיות $O(n\log n)$

```
int LISLength(int[]arr){
    size←arr.length
    d[]←new int[size]
    d[0] ←arr[0]
    end←0
    for i←1 to size//O(n)
       index←binarySearchBetween(d, end, arr[i])// O(log n)
       if(index<=end)
          d[index] ←arr[i];
       else{
          end++;
          d[end] ←arr[i];
return end+1;
}
```

פונקציית עזר בבעיית LIS.

PesodCode:

```
/**    ממיין מערך בתוך המספר מיקום של בינארי חיפוש*/
int binarySearchBetween(int []arr, int end, int value){
      int low = 0, high = end;
      if (value<arr[0]) return 0;
      if (value>arr[end])  return end+1;
      while (low <=high){
            int middle = (low + high)/2;
            if (low == high) {
                  if (arr[low] == value) return low;
                  else return low;
            }
            else {
                  if (arr[middle] == value){//value was found
                        return middle;
                  }
                  // value suppose to be left
                  if (value < arr[middle]){
                        high = middle;
                  }
                  // value suppose to be right
                  else{
                        low = middle+1;
                  }
            }
      }
      return -1;
}
```

בעיית LCS בעזרת LIS.

המטרה – קיימות 2 מחרוזות ואנו צריכים לייצור איזו סדרה C במטרה לשפר את הסיבוכיות, $LCS(A,B)=LIS(C)$.

$$\bar{X}:\{a,b,a,c,u\},|X|=m$$
$$\bar{Y}:\{b,a,a,b,c,a\},|Y|=n$$

האלגוריתם מחולק לכמה חלקים:

1. ראשית ניצור מערך עזר שהוא מערך של מחסניות(לפי ...a,b,c סה"כ 26 אברים), שמייצג אינדקסים של כל אות במערך $\bar{Y}$. (כל אות ב – ...a,b,c).
2. שנית ניצור מערך A. עוברים על כל אות במערך $\bar{X}$ ומוציאים את כל האינדקסים שהאות שלהם מופיעה במערך עזר ומוסיפים אותם ל – A.
3. אנו מפעילים LIS על מערך A.

Z:

a	b	c	d	e	f	……….	u	……….	z
2	1	5							
3	4								
6									

$A:\{6,3,2,4,1,6,3,2,5\}$

הסבר של הקוד:

- עוברים על כל איבר במערך Y,ומכניסים את האינדקסים של האות למחסנית המתאימה במערך עזר Z.

PesodCode: $O(n)$

$$\begin{aligned}&For\ i\leftarrow 1 \text{ to } n\\&\quad Z[\bar{Y}(i)]\leftarrow push(i)\end{aligned}$$

- עוברים על כל איבר במערך X ניגשים למקום הנכון במערך מחסניות Z ומוסיפים את כל האברים של אותה מחסנית למערך A.

PesodCode: $O(r)$

$$\begin{aligned}&A[\,]\\&For\ i\leftarrow 1 \text{ to } m\\&\quad if\ (!Z[\bar{X}(i)].empty(\,))\\&\qquad A=A+Z[\bar{X}(i)]\end{aligned}$$

- מפעילים LIS על Aומקבלים תוצאה(המספרים שיוצאים הם האינדקסים שיוצרים LCS בין 2 המחרוזות).

PesodCode: $O(r\log r)$

$$ANS\leftarrow LISLength(A)$$

הסיבוכיות היא - $\min(n,m)+(r)+(r\log r)\approx O(r\log r)$.

PesodCode:

```
buildHelpZ(){בניית מערך עזר←--O(n)
        // O(Y.length)
        int index
        for i←n to 0   //i--
                index←Y[i]-'a'
                Z[index].st.push(i)
}
buildClusterA (){// O(|A|*log|A|) בניית מערך אשכולות←--O(r)
        int index, count←0
        for i←0 to m
                index←X[i]-'a'
                if (!Z[index].st.empty())
                        for j←0 to Z[index].st.size()
                                A[count] ←Z[index].st.get(j)
                                System.out.print(A[count]+",")
                                count++
}
Int LongestCommSeq(A){ חישוב המחרוזת המשותפת הארוכה ביותר←--O(r log r)
         buildHelpY();
         buildClusterArr();
         S ←LIS(A)
return  S.length
}
```

שברים עשרוניים אינסופיים מחזוריים.

נכתוב פונקציה שמקבלת שבר פשוט כשני מספרים שלמים, כך שהראשון m הוא מונה והשני n הוא מכנה.

הפונקציה מחזירה מחרוזת של שבר עשרוני עם N=20 ספרות אחרי נקודה עשרונית.

הנחה: המונה קטן ממכנה.

PesodCode:

```
String Fraction(String ans1,int m,int n){
        index  ←  0
        ans1   ←  ""
        mone  ←  m*10
        N  ←  20
        sharit  ←  m
        ans ←  "0."
        for i←1 to N
                if(mone>=n)
                        sharit  ←  mone%n
                        ans1  ←  ans1+sharit
                        index++
                        mone  ←  mone/n
                        ans  ←  ans+mone
                        mone  ←  sharit
                        mone  ←  mone*10

                else
                        sharit  ←  mone % n
                        ans1  ←  ans1+sharit
                        index++
                        mone  ←  sharit
                        mone  ←  mone*10
                        ans  ←  ans+"0"
        return ans
}
```

הסיבוכיות - $O(n)$.

בעיית בדיקת מיון המערך ע"י חוקיות מסוימת.

נתונה לי סדרה מסוימת ממוינת $A=\{a_1,a_2,......,a_n\}$ ועפ"י חוקיות נתונה $\sum_1^n(-1)^n a_n$ יש לבנות סדרה חדשה ולבדוק האם היא ממוינת.

PesodCode:

```
Check(A){
  B ← A[0]
  temp ← B
  ans ← true;
  for i ← 2 to n
        B ← B + (−1)^n a_n
        if(temp<B)
            temp ← B
        else
            ans ← false
            break
 end for
return ans
}
```

הסבר: temp - תמיד משתנה שבעל ערך קודם ל – B.

B – משתנה שמקבל כל פעם את האיבר הבא בחישוב.

Ans – משתנה המראה אם הסדרה ממויינת.

אנחנו סוכמים כל פעם את המשתנה B ובודקים אם הוא יותר גדול מ – temp .

הוצאת min/max ממערך מעגלי ב סיבוכיות של $O(\log n)$.

PesodCode:

```
function int smallest/bigest(int[]x){// |x| = n
      int start ← x[0]
      int finish ← n
      int middle ← n/2
      boolean flag ← false
      while(flag==false)
            if(x[midle-1]>x[midle]<x[midle+1])
                  flag ← true
                  ans ← x[midle] for min
                  ans ← x[midle-1] for max
            if(x[midle]<x[0])
                  finish ← midle-1
            else if(x[midle]>x[0])
                  start ← midle+1
            midle ← (start+finish)/2
      end while
return ans
}end function
```

הסיבוביות $O(\log n)$ באה כי אנו משתמשים באלגוריתם חיפוש בינארי.

הרעיון של השימוש הוא כי הסדרה ממוינת מעגלית שזה אומר של איבר שבא אחרי X_i הוא יותר גדול וההפך.

העוגן שלנו הוא תמיד לכיוון של מהקטן ביותר לגדול ביותר או להפך.

בעיית המטבעות.

בעיה זו דומה מאוד לבעיית המטוס,אנו נבנה מטריצה ונמלא רק חצי,כל צעד יכול להיות למעלה או שמאלה(לא ימינה).
ולאחר שנמלא אותה נחזיר את המסלול המקס' במקום המינמ'.
עבור n = 4 נבנה מטריצה 4X4 ונמלא רק חצי שלכל איבר בה יש את סכום המטבע ואת ה – entry של הסכום שצבר עד כה.

PesodCode:

```
for i ← 1 to n
      for j  ← 1 to n
             a←a[i-1][j].entry + a[i-1][j]
             b←a[i-1][j-1].entry + a[i][j]
             if(a>b) then a[i][j] = a
             else a[i][j] = b
return a[i][j].entry
```

הסיבוכיות היא - $O\left(n^2\right)$, כי ממלאים את ה – entry בכול איבר במטריצה .

בשביל לקבל את במסלול הולכים לקודקוד העליון ובודקים עפ"י התנאי הבא:

			5
		5	4
	2	1	8
1	7	4	3

PesodCode:

```
p  ←  p + "["+n+","+n+"]"
i  ←  n
j  ←  n
if((a[i][j].entry - a[i][j])== a[i][j-1])
      p  ←  p + "->["+(i)+","+(j-1)+"["

      j--
else
      p  ←  p + "->["+(i-1)+","+(j-1)+"]"
      j--,i--
syso("the max path:"+p)
```

משחק המספרים – במערך מעגלי.

בעצם לשחק הראשון יש אפשר לבחור בין כל אחד מהמספרים מהמערך – ולכן לכל מס' שייקח השחקן הראשון יש מערך משחק שונה, לכן יש n מערכי משחקים.

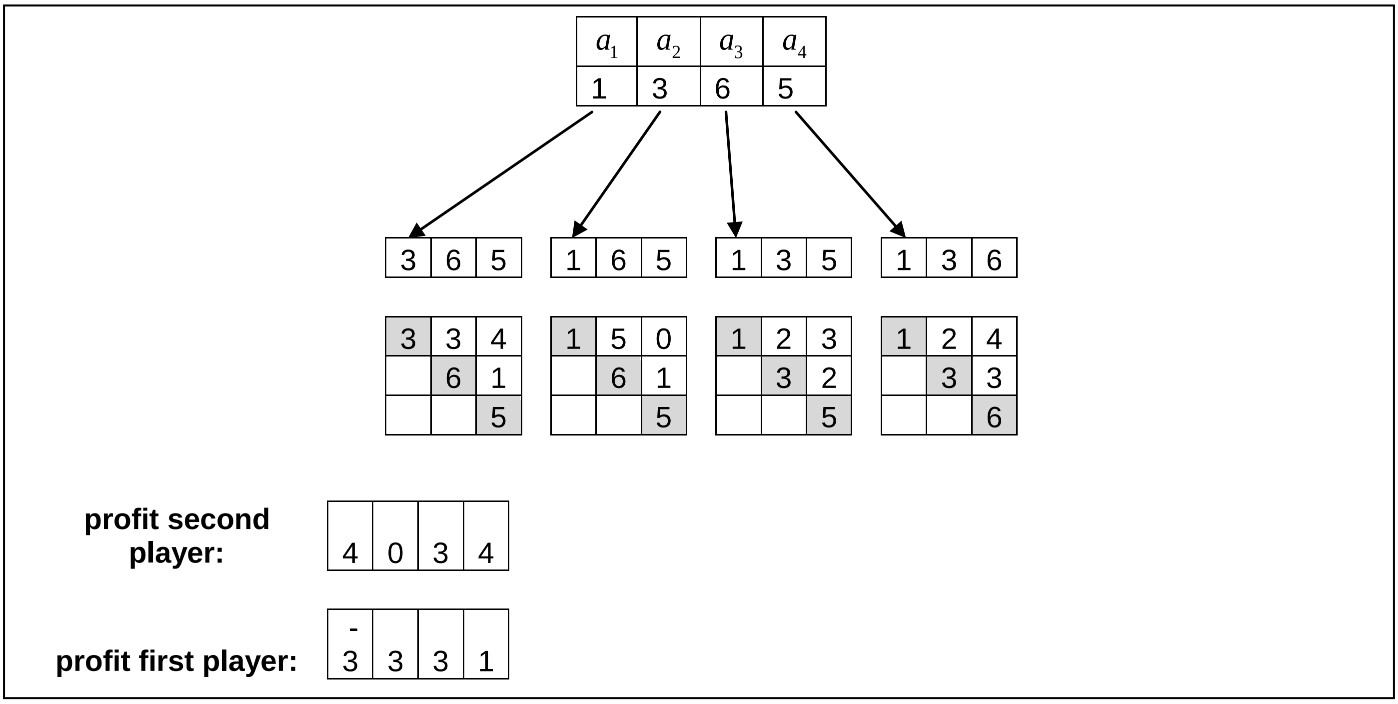
לכן נכניס את כל ה n – מערכי משחקים למטריצה: `Matrix[][]SubGameArray←GenerateSubGames(A)`

כל שורה במטריצה מציינת מערך של משחק.

ולכל מערך משחק כזה יש מטריצת משחק משלו,לכן יש n מטריצות בגודל [n-1][n-1], נכניס למערך תלת מימדי.

`Aarray[][][]GameMatrix←new Array[n][n-1][n-1]`

יש n מערכי משחקים שלכל משחק יש מטריצת המשחק בגודל [n-1][n-1].



לאחר מכן נבנה את מטריצות המשחק לכל תתי המשחקים,ונבנה מערך של רווחים ונכניס אליו את כל הרווחים מכל המשחקים. הרווחים הם של השחקן השני.

```
for i ← 1 to n
     GameMatrix [i][][] ← BuildMatrix(SubGameArray[i][])
     profitArray[i] ← GameMatrix[i][1][n-1]
end for
```

אבל אנחנו רוצים את הרווח של השחקן הראשון לכן נמצא את הרווחים של השחקן הראשון.

```
for i ← 1 to n
    profitArray[i] ← A[i] - profitArray[i]
end for
```

לאחר מכן נמצא ממערך הרווחים של השחקן הראשון את הרווח המקס' .

```
index ← GetMax(profitArray)
Print(A[Index]) // print 1 player choose
printGameStrategy(GameMatrix[index][][])
```

ונדפיס את הדרך האופטימאלית לניצחון של השחקן הראשון.

```
Numbers Game in Circle Array :

Function CircleNumbersGame(int[]A)
      int n ← A.length
      // create arrays of all subgames
      Matrix  [][] SubGameArray ← GenerateSubGames (A)
      //create 3D array. Means array of game matrices.
      Aarray [][][] GameMatrix ← new Array [n][n-1][n-1]
      Array [] ProfitArray ← new Array[n]
fill the game matrices. For max difference in "profit" and stack all profits in profitArray
      for i ← 1 to n
           GameMatrix [i][][] ← BuildMatrix(SubGameArray[i][])
           profitArray[i] ← GameMatrix[i][1][n-1]
      end for
      // evaluate total profit for all subgames
      for i ← 1 to n
          profitArray[i] ← a_i - profitArray[i]
      end for
      //find the max profit index. It would be our strategy.
      index ← GetMax(profitArray)
      Print(A[Index])       // print 1 player choose
      printGameStrategy(GameMatrix[index][][])print the whole game strategy after first move
end function
//בניית המטריצה מתבצעת עפ"י הכללים שלמדנו
Function BuildMatrix(Array[]A)
      Matrix [][] M ← new Matrix[][]
      for i ← 1 to A.length
             M [i][i] ← a_i
      for i ← (n-1) to 1
            for j ← (i+1) to n
                 M[i][j] ← Max(a_i -M[i+1][j], a_j - M[i][j-1] )
      return M
End BuildMatrix

Function printGameStrategy( Matrix M [][] )
      i ← 1 ,  j ← M[1].length
      for k ← 0 to n
              X ← a[i]-F[i+1,j]
              Y ← a[i]-F[i,j-1]
              if(X>Y)
                   print A[i] i←i+1
              else print A[j] j←j-1
      end for
End printGameStrategy
```

20 פברואר, 2010