

רקורסיות

קרוניקליף

ביחידה זו נלמד:

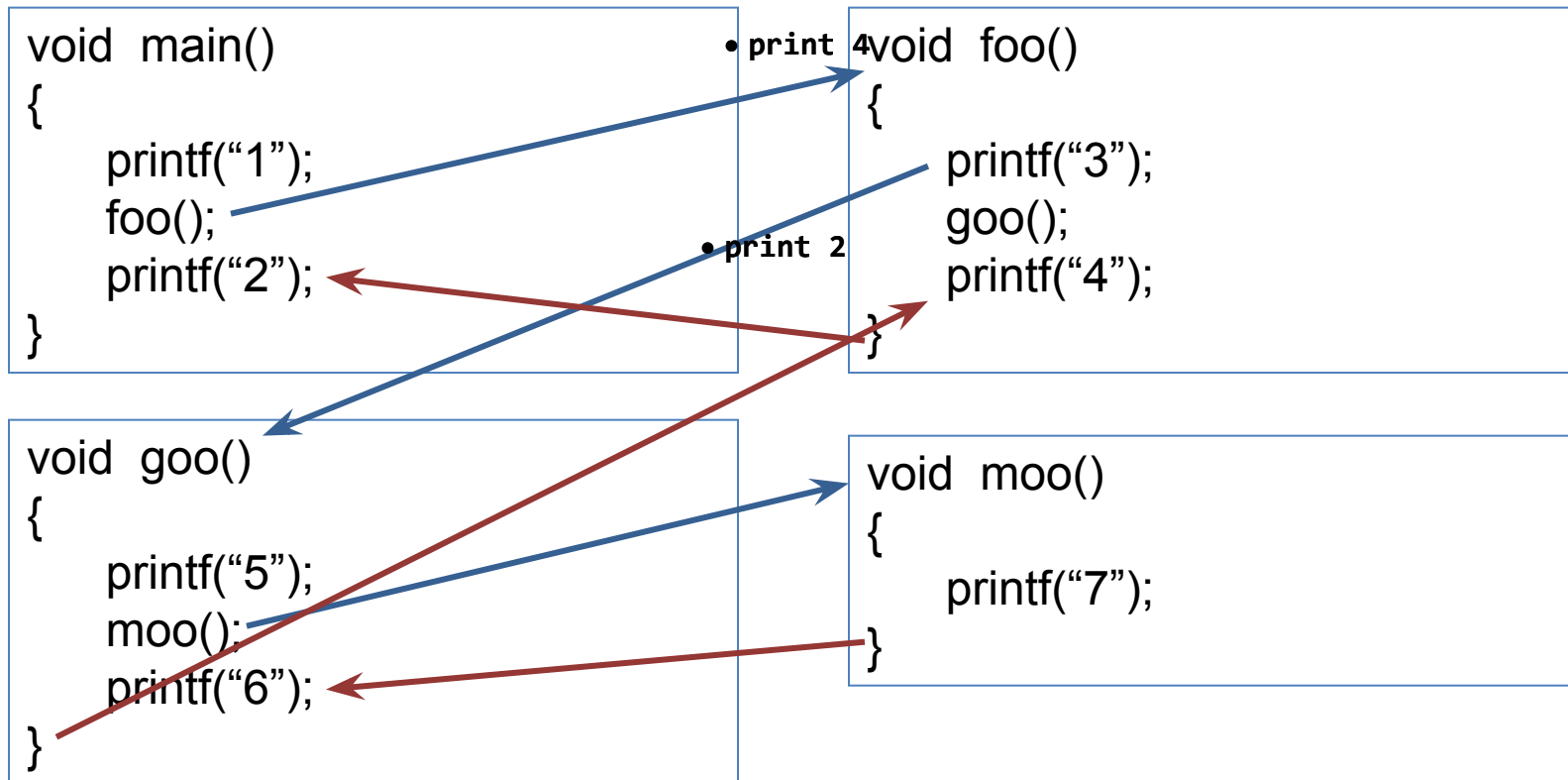
- מהי רקורסיה
- מרכיבי הרקורסיה
- ניתוח זמן ריצה
- רקורסיות ומערכים
- רקורסיות ומצביעים

- goo
- print 5

- moo
- print 7

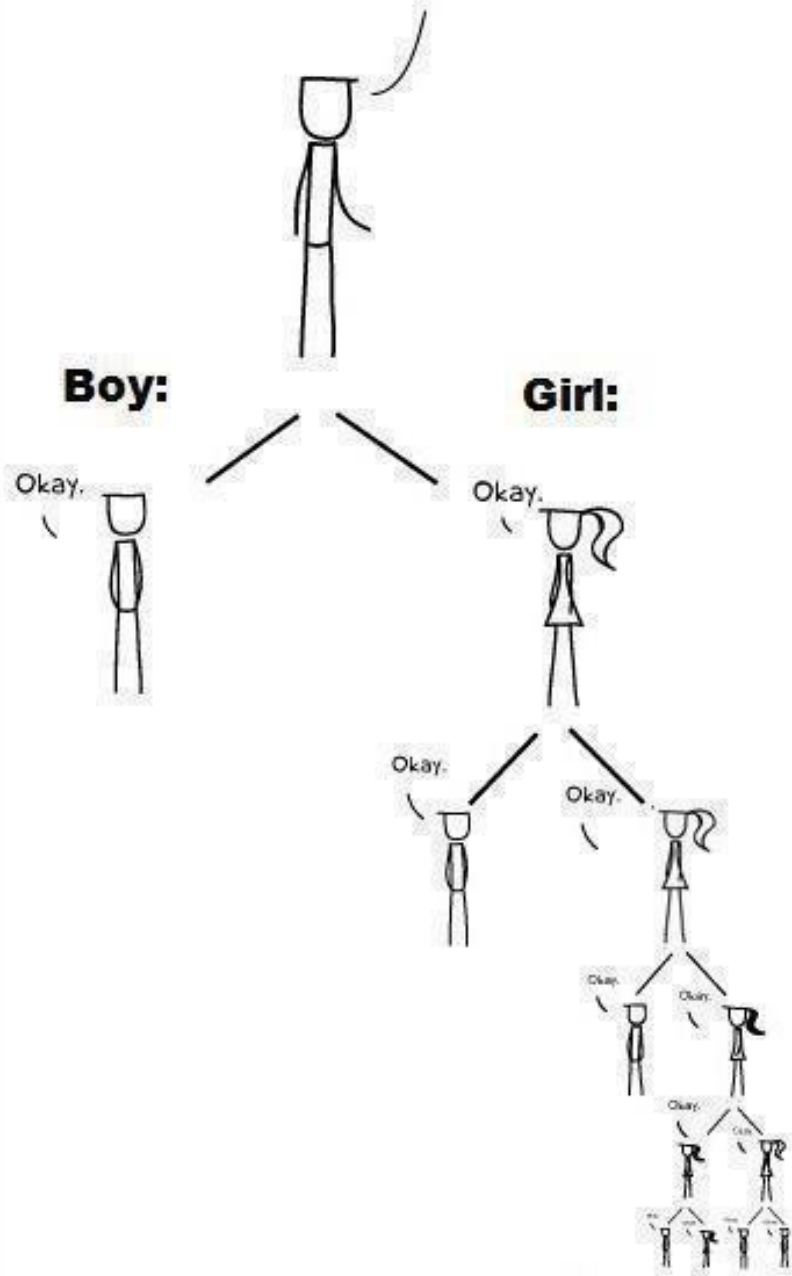
תזכורת: פונקציות

- print 6



Just PROMISE not to tell anyone else.

מה הבעיה בסנריו הבא?



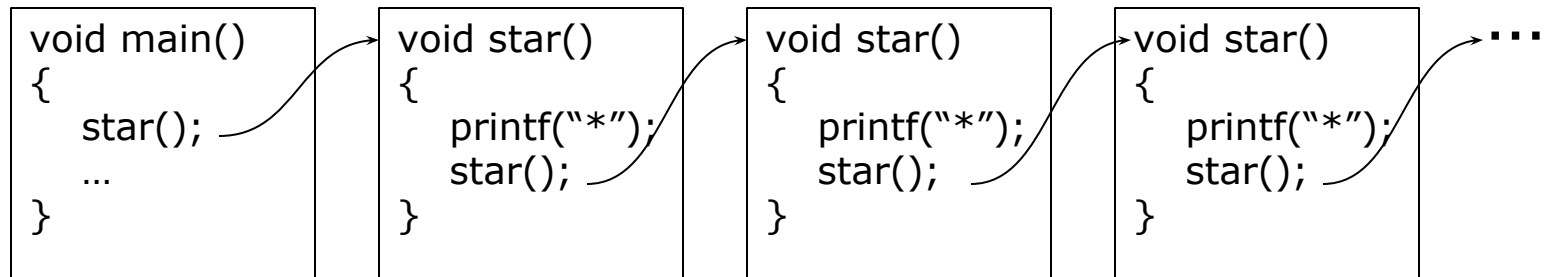
http://www.cutorcopy.com/wp-content/uploads/albums/134075539977097_35844/just-promise-not-to-tell-anyone-else.jpg

מהי רקורסיה?

□ רקורסיה היא פונקציה שקוראת לעצמה

```
void star()
{
    printf("*");
    star();
}
```

בפונקציה זו מודפסת כוכבית
..ואז שוב קוראים לפונקציה



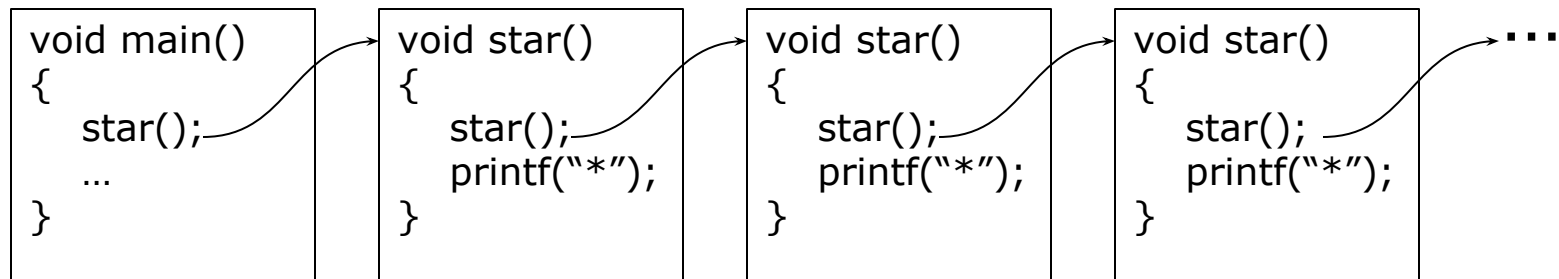
□ מתי התוכנית תסתיים?

■ אף פעם! והפלט יהיה שורה אינסופית של כוכביות

מהי רקורסיה? (2)

מה היה הפלט אם היינו כותבים את הפונקציה כך:

```
void star()
{
    star();
    printf("*");
}
```



הפונקציה רקורסיבית, אך לא מדפיסה דבר משום שהקריאה לרקורסיה מתבצעת לפני ההדפסה, שלעולם אינה מסתיימת..

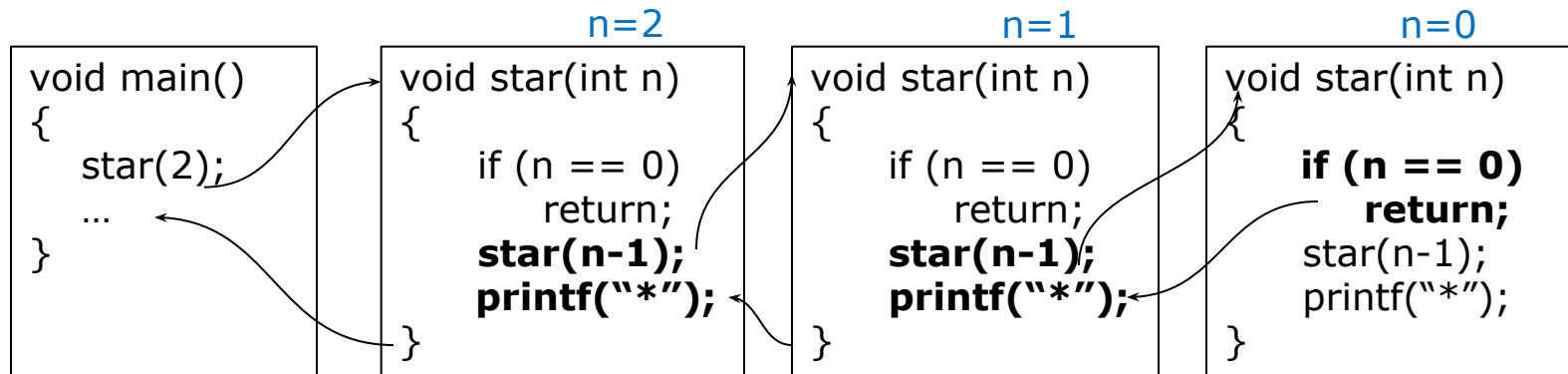
מהי רקורסיה? (3)

□ היינו רוצים שהפונקציה תעצור..

```
void star(int n)
{
    if (n == 0)
        return;
    star(n-1);
    printf("*");
}
```

תנאי העצירה מבטיח שהפונקציה תעצור,
..ותתחיל לחזור משרשרת הקריאות
לכן צריך גם להתקדם לעבר תנאי העצירה
ע"י שקוראים לפונקציה עם ערך קטן יותר

דרך נוספת להסתכל על פתרון הבעיה: קרא פונקציה
המדפיסה n-1 כוכביות, והדפס כוכבית נוספת



לשים לב לא לשכוח תנאי עצירה!

גיא דפני ◀ בדיחות מתכנתים - Programmers Jokes

31 דקות · 🌐



בעשרים וחמישה באוקטובר בשתיים לפנות בוקר מזיזים את השעון שעה לאחור (לאחת).

מי לעזזאל ניסח את זה? הוא שכח תנאי עצירה!

ניסוח נכון: בעשרים וחמישה באוקטובר בשתיים לפנות בוקר אם עדיין לא שינית לשעון חורף, מזיזים את השעון שעה לאחור (לאחת).

מהי רקורסיה? (4)

- ראינו כי עבור פונקציה הקוראת לפונקציה אחרת יש לנו כניסה נוספת במחסנית הקריאות
- ראינו כי כאשר יוצאים מפונקציה מסוימת, חוזרים לפונקציה שקראה לה, לפקודה אחרי פקודת הקריאה
- אותו הדבר עם פונקציה רקורסיבית. אין שוני בהתנהגות בגלל שהפונקציה קוראת לעצמה
- כל קריאה לפונקציה רקורסיבית מקבלת שטח זיכרון (כמו כל פונקציה) ובה מוגדרים משתני הפונקציה
 - אין משמעות לכך שיש כמה מחסניות בו זמנית עם אותם משתנים, שכן כל מחסנית שייכת לקריאה שונה של הפונקציה

פתרון בעיות באמצעות רקורסיה

- ישנן בעיות שהפתרון שלהן מתבסס על פתרון אותה בעיה, רק פשוטה יותר (קלט קטן יותר)
- הבעיה הפשוטה יותר מתבססת על פתרון של בעיה פשוטה עוד יותר
- בסופו של התהליך נגיע למקרה פשוט עד מאוד, שאת הפתרון עבורו אנו יודעים
- רקורסיה היא הדרך לבנות פתרון זה..
- ברוב במקרים ניתן לפתור את הבעיה בקלות ע"י לולאה (אבל לא תמיד..)

דוגמא: חישוב עצרת

עצרת היא הפונקציה (עבור n חיובי) : $n! = 1*2*3*...*n$

דרך נוספת לראות חישוב זה: $n! = (n-1)!*n$

יש לזהות מהו הפתרון לבעיה הקטנה ביותר:

■ עבור $n \leq 1$ הפתרון הוא 1

ומכאן הדרך לקוד מאוד פשוטה:

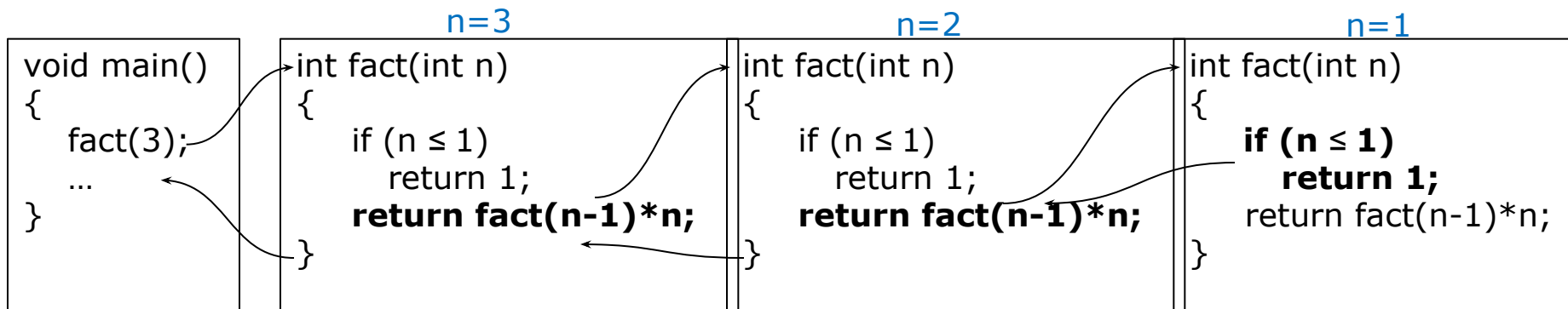
```
int factorial(int n)
{
    if (n ≤ 1)
        return 1;
    return factorial(n-1)*n;
}
```

דוגמא: חישוב עצרת (מחסנית הקריאות)

```
int factorial(int n)
{
    if (n ≤ 1)
        return 1;
    return factorial(n-1)*n;
}
```

- fact(3)
- fact(2)
- fact(1)
- 1

•*3



מרכיבי הרקורסיה

□ כדי לכתוב פונקציה רקורסיבית שעוצרת צריך 3 חלקים:

- תנאי עצירה (המקרה הקטן ביותר של הבעיה)
- קריאה רקורסיבית (לבעיה בגודל אחד יותר קטן)
- או: האמונה שהפונקציה יודעת לעשות את מה שמצפים ממנה
- קישור הפתרון של הבעיה בגודל אחד יותר קטן עם הגודל הנוכחי

□ דוגמא:

```
int factorial(int n)
```

```
{
```

```
if (n ≤ 1)
```

```
return 1;
```

```
return factorial(n-1) * n;
```

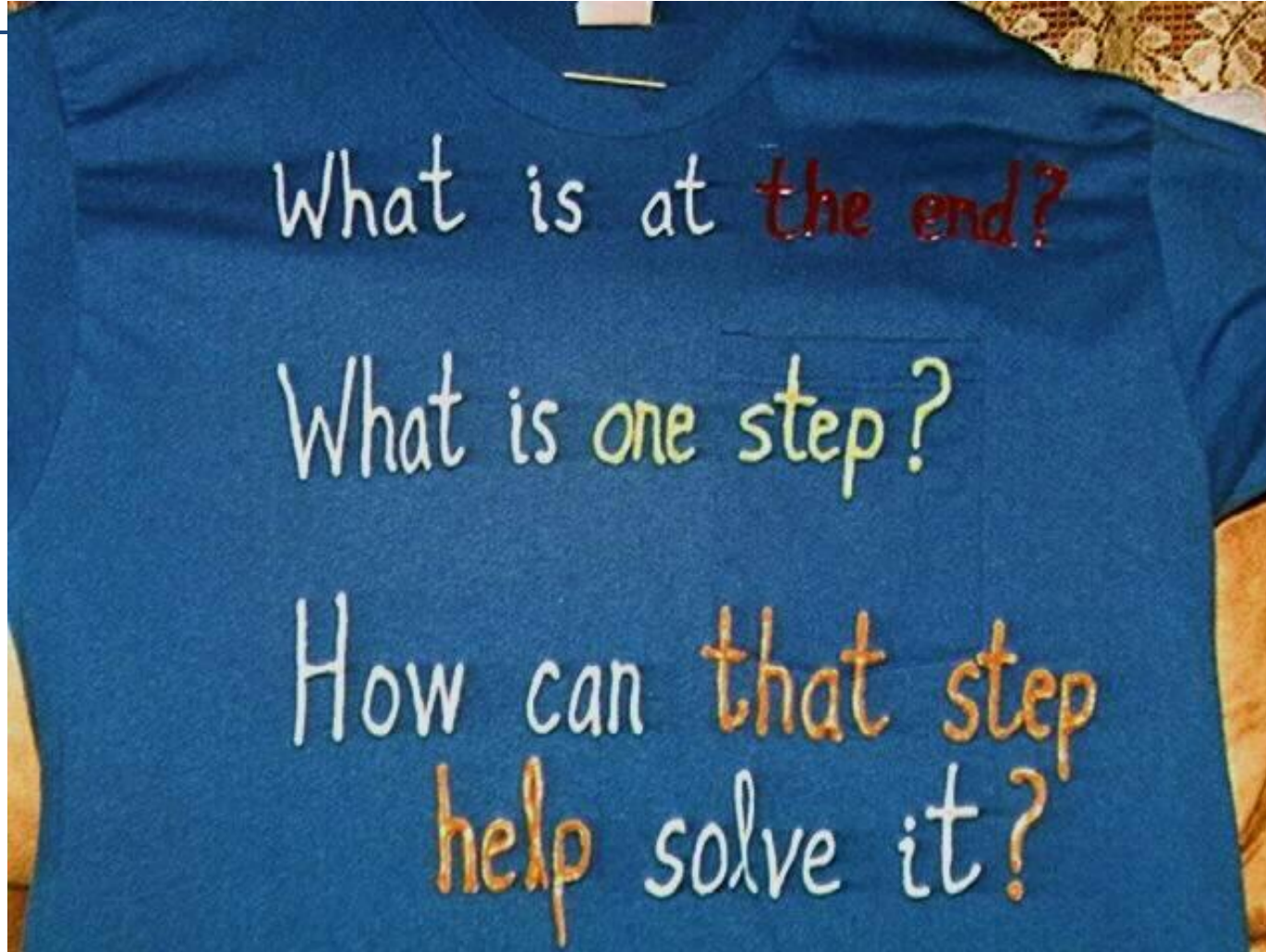
```
}
```

תנאי עצירה

קריאה רקורסיבית

קישור

ומישהו כבר הגדיר את זה טוב ממני 😊



<http://www.eecs.ucf.edu/~leavens/T-Shirts/227-342-recursion-front.JPG>

רקורסיות – ניתוח זמן ריצה

□ נרצה לדעת לחשב את זמן הריצה של פונקציה רקורסיבית

□ זמן הריצה של פונקציה רקורסיבית הוא:

■ זמן הריצה של הפונקציה (מלבד הקריאה הרקורסיבית) *
מספר הקריאות הרקורסיביות בסה"כ

□ נראה בהמשך דוגמא בה לגירסא הרקורסיבית זמן פחות
טוב מאשר לגירסא האיטרטיבית ולהפך

דוגמא: חישוב עצרת - ניתוח יעילות

□ הפונקציה factorial עושה פעולות בסיסיות (בדיקת שוויון וכפל) שלוקחות $O(1)$ וכן קוראת לפונקציה factorial

□ בסה"כ נבצע n קריאות לפונקציה factorial, שבכל קריאה מבצעים $O(1)$ פעולות

□ לכן זמן הריצה של factorial הוא $O(n)$

□ זמן הריצה של הגרסה הרקורסיבית זהה לזמן הריצה של הגירסא האיטרטיבית: $O(n)$

הדפסת משולש

```
C:\WINDOWS\system32\cmd.exe
*
**
***
****
Press any key to continue . . .
```

```
void printTriangle(int n)
```

```
{
```

```
    int i;
```

```
    if (n == 0)
```

```
        return;
```

```
    printTriangle(n-1);
```

```
    for (i=0 ; i < n ; i++)
```

```
        printf("*");
```

```
    printf("\n");
```

```
}
```

- printTri(4)

- printTri(3)

- printTri(2)

- printTri(1)

זוהי "רקורסיית ראש":
הקריאה הרקורסיבית
נקראית בהתחלה

• וכביות:

• כוכביות:

• 4 כוכביות: — — — — —

ניתוח זמן הריצה: בכל קריאה

לפונקציה מבצעים $O(n)$ פעולות,

ובסה"כ יש n קריאות לפונקציה ←

$O(n^2)$, כמו הגירסא האיטרטיבית

מה היה קורה אם היינו מחליפים בסדר בין הקריאה הרקורסיבית

C:\WINDOWS\system32\cmd.exe

```
****
***
**
*
Press any key to continue .
```

הדפסת משולש

```
void printTriangle(int n)
{
    int i;
    if (n == 0)
        return;

    for (i=0 ; i < n ; i++)
        printf("*");
    printf("\n");

    printTriangle(n-1);
}
```

• printTri(4)
• printTri(3)
• printTri(2)
• printTri(1)
• ק
• ת
• יות
• נביות
• ונביות

זוהי "רקורסיית זנב":
הקריאה הרקורסיבית
נקראת לאחר הפעולות

דוגמא: חישוב חזקה

□ חזקה היא פונקציה המקבלת בסיס n וחזקה k :

□ דרך נוספת לראות חישוב זה: $n^k = n^{k-1} * n$

□ יש לזהות מהו הפתרון לבעיה הקטנה ביותר:

■ עבור $k=0$ הפתרון הוא 1 ($n^0 = 1$)

□ ומכאן הדרך לקוד מאוד פשוטה:

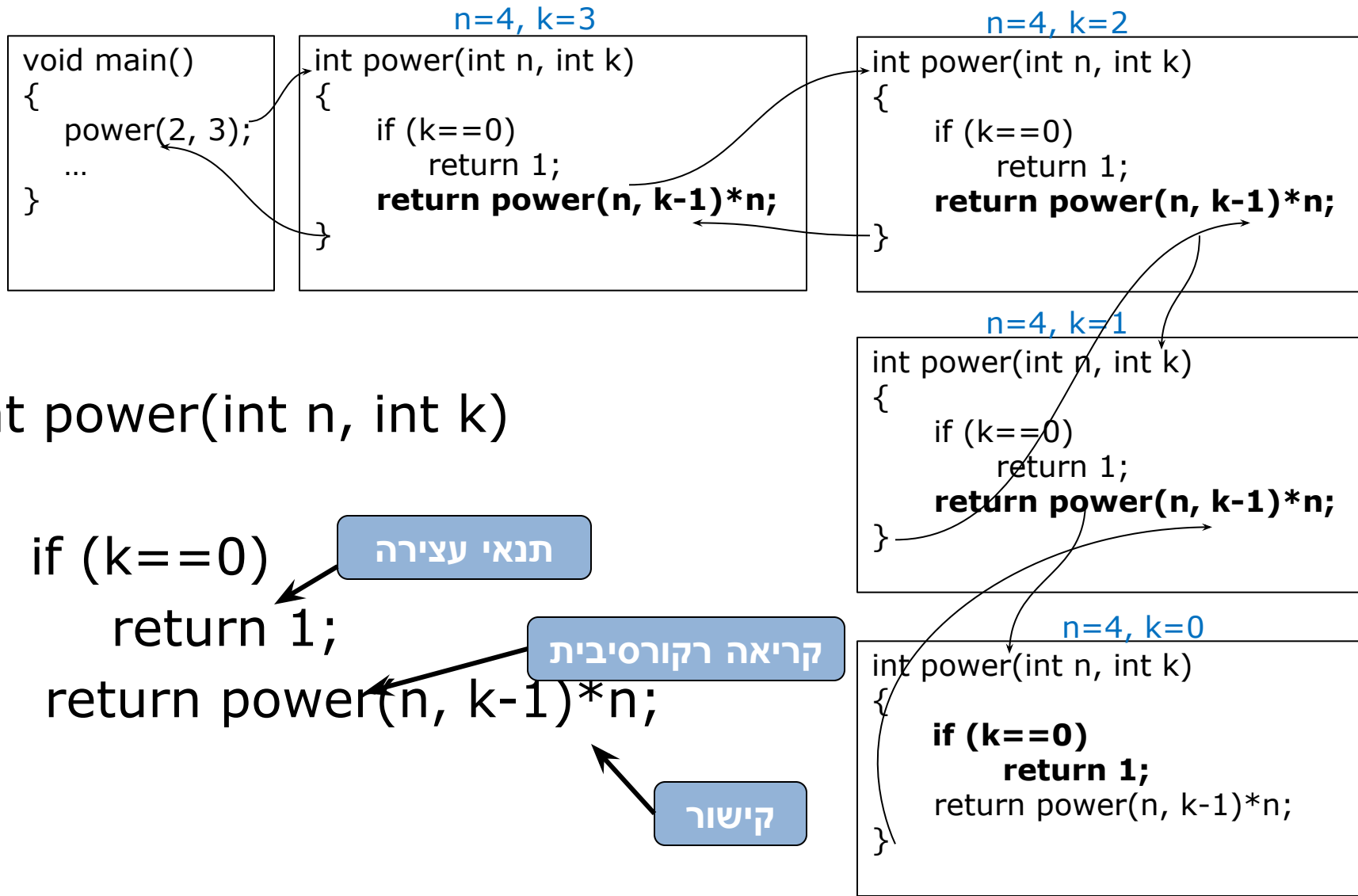
```
int power(int n, int k)
{
    if (k==0)
        return 1;
    return power(n, k-1)*n;
}
```

•pow(4,3)
•pow(4,2)
•pow(4,1)
•pow(4,0)
•1

•*4

•*4

דוגמא: חישוב חזקה (מחסנית הקריאות)



דוגמא: חישוב חזקה - ניתוח יעילות

- הפונקציה `power` עושה פעולות בסיסיות (בדיקת שוויון וכפל) שלוקחות $O(1)$ וכן קוראת לפונקציה `power`
- בסה"כ נבצע k קריאות לפונקציה `power`, שבכל קריאה מבצעים $O(1)$ פעולות
- לכן זמן הריצה של `factorial` הוא $O(k)$
- זמן הריצה של הגירסא הרקורסיבית זהה לזמן הריצה של הגירסא האיטרטיבית: $O(k)$

```
int power(int n, int k)
{
    int result = 1, i;

    for (i=0 ; i < k ; i++)
        result *= n;

    return result;
}
```

דוגמא: חישוב חזקה (גירסא 2)

- חזקה היא פונקציה המקבלת בסיס n וחזקה k :
- דרך נוספת לראות חישוב זה:

■ עבור k זוגי: $n^k = n^{k/2+k/2} = n^{k/2} * n^{k/2}$

■ עבור k אי-זוגי: $n^k = n^{k/2+k/2+1} = n^{k/2} * n^{k/2} * n$

■ דוגמא:

$$2^8 = 2^{4+4} = 2^4 * 2^4 = 2^{8/2} * 2^{8/2}$$

$$2^9 = 2^{4+4+1} = 2^4 * 2^4 * 2 = 2^{8/2} * 2^{8/2} * 2$$

□ יש לזהות מהו הפתרון לבעיה הקטנה ביותר:

■ $k=0$ הפתרון הוא 1 ($n^0 = 1$)

דוגמא: חישוב חזקה (גירסא 2) – הקוד במימוש איטרטיבי

$n^k = n^{k/2+k/2} = n^{k/2} * n^{k/2}$ עבור k זוגי:

$n^k = n^{k/2+k/2+1} = n^{k/2} * n^{k/2} * n$ עבור k אי-זוגי:

```
int power(int n, int k)
{
    int result = 1, i;

    for (i=0 ; i < k/2 ; i++)
        result *= n;

    if( k%2 == 0)
        return result*result
    else
        return result*result*n;
}
```

היעילות היא $O(k)$

דוגמא: חישוב חזקה (גירסא 2) – הקוד

הרצת פונקציה
 • pow(2,8)
 • t=pow(2,4)
 • t=pow(2,2)
 • t=pow(2,1)
 • t=pow(2,0)
 עבור k זוגי:

$$n^k = n^{k/2+k/2} = n^{k/2} * n^{k/2}$$

עבור k אי-זוגי: $n^k = n^{k/2+k/2+1} = n^{k/2} * n^{k/2} * n$

```
int power(int n, int k)
```

```
{
```

```
    int t;
```

```
    if( k == 0)
        return 1;
```

```
    t = power(n, k/2);
```

```
    if( k%2 == 0)
        return (t*t);
```

```
    else
        return (t*t*n);
```

```
}
```

• pow(2,9)
 • t=pow(2,4)
 • t=pow(2,2)
 • t=pow(2,1)
 • t=pow(2,0)
 • 1

• t*t*2

• t*t

• t*t*2

• t*t

• t*t

• t*t

דוגמא: חישוב חזקה (גירסא 2) – עץ הקריאות

עבור בעיה בגודל 16

- $\text{power}(2, 16)$
- $\text{power}(2, 8)$
- $\text{power}(2, 4)$
- $\text{power}(2, 2)$
- $\text{power}(2, 1)$
- $\text{power}(2, 0)$
- פעולות קבועות

פעולות קבועות

פעולות קבועות

פעולות קבועות

פעולות קבועות

פעולות קבועות

ניתן לראות שבכל רמה בעץ יש קריאה רקורסיבית אחת ופעולות קבועות, ובעץ יש $\log(k)$ רמות זמן ריצה הוא $\leftarrow O(\log(k))$. בגירסא הקודמת זמן הריצה היא $O(k)$. ירדנו בסדר גודל שלם!

אינטואיציה לגבי חישוב זמן הריצה: בכל איטרציה אנו מחשבים קלט שקטן פי $1/2$ מהקלט הקודם, וזו בדיוק הפונקציה \log .

ויש מקרים בהם הפתרון הרקורסיבי יותר גרוע..

□ למשל חישוב סדרת פיבונצ'י, המוגדרת כך:

$$a_0 = 0, a_1 = 1$$

$$a_n = a_{n-1} + a_{n-2} \quad \text{לכל } n \geq 2$$

■ למשל עבור $n=7$ הסדרה נראית כך: 0,1,1,2,3,5,8,13

□ מאופן הצגת הבעיה הפתרון ברור:

```
int fib(int n)
{
    if (n ≤ 1)
        return n;
    return fib(n-1) + fib(n-2);
}
```

•Fib(0)
•0

סדרת פיבונצ'י – עץ הקריאות

•Fib(1)
•1

•Fib(2)
•Fib(1)
•1

•Fib(0)
•0

ניתן לראות שבכל רמה בעץ יש 2 קריאות רקורסיביות,
ובעץ יש n רמות ← זמן ריצה הוא $O(2^n)$

•Fib(3)
•Fib(2)
•Fib(1)
•Fib(0)

•Fib(0)

סדרת פיבונצ'י – גירסא איטרטיבית

נסתכל על מימוש הפתרון בשימוש לולאות: □

```
int fibonacci(int n)
{
    int x0 = 0, x1 = 1, temp, i;

    if (n <= 1)
        return n;

    for (i=2 ; i <= n ; i++) {
        temp = x0 + x1;
        x0 = x1;
        x1 = temp;
    }
    return x1;
}
```

איברי הסדרה הראשונים:

13 8 5 3 2 1 1 0

int: n	5
int: x0	3
int: x1	5
int: temp	5
int: i	6

זמן הריצה של הפונקציה הוא $O(n)$, משמעותית טוב
מזמן הריצה של הגירסא הרקורסיבית!

אז למה בעצם צריך רקורסיות?

□ עד כה ראינו דוגמאות שאנו כבר יודעים לפתור בצורה איטרטיבית

■ אבל הפתרון הרקורסיבי מאוד אינטואיטיבי

□ יעילות זמן הריצה של הגירסא האיטרטיבית זהה

■ נראה דוגמא בה דווקא זמן הריצה הרקורסיבי טוב מזמן הריצה של הפתרון האיטרטיבי

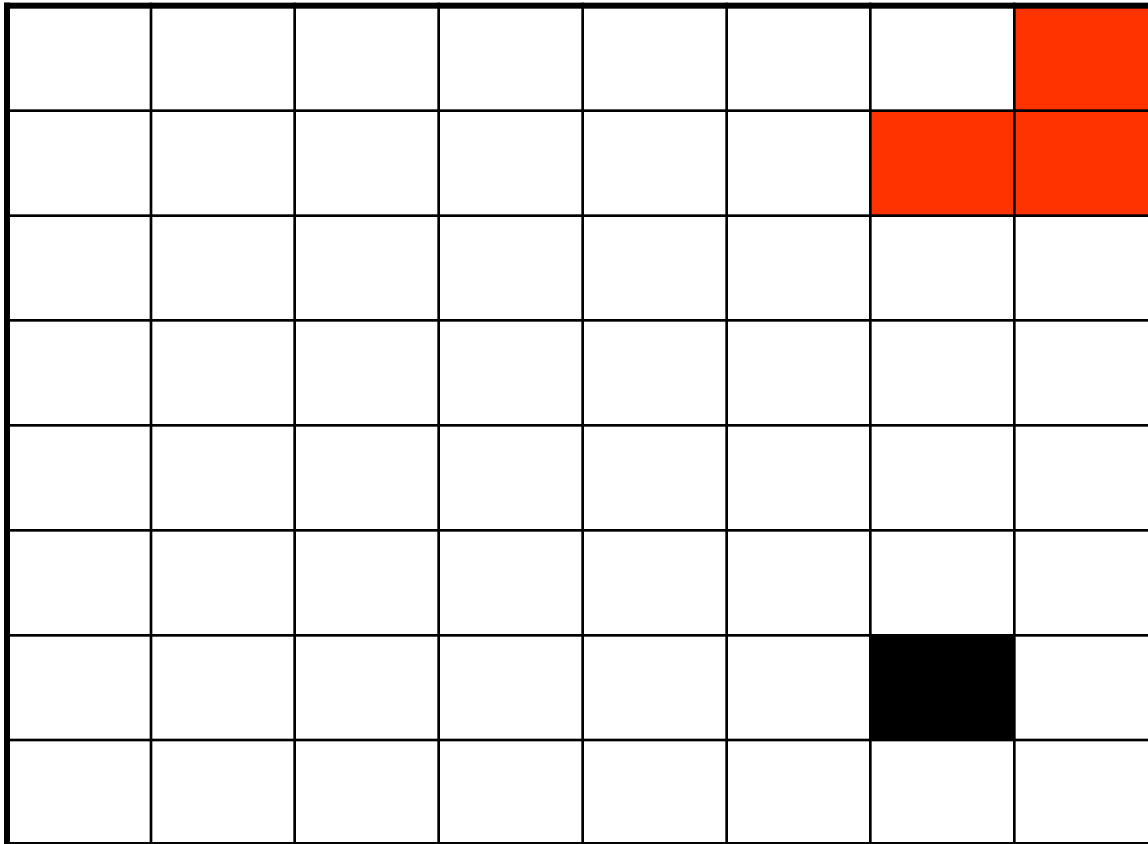
□ אפילו יש חיסרון של זיכרון בגלל פתיחת מחסנית הקריאות, וכן לפעמים הקוד פחות קריא

□ אבל...

■ יש בעיות שמאוד קשה למצוא להן פתרון לא רקורסיבי!

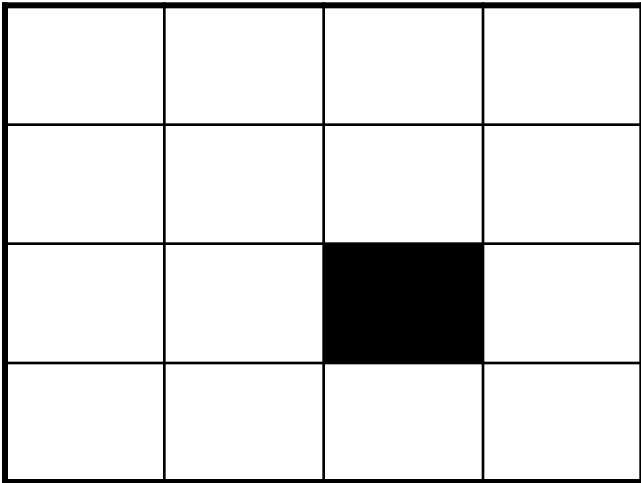
חידה - מילוי לוח משבצות

לפנינו לוח בגודל 8×8 (או במילים אחרות: $2^3 \times 2^3$). בלוח יש משבצת שחורה אחת. ברשותנו חלקים כמו החלק האדום. עלינו למלא את הלוח. כיצד?



מילוי לוח משבצות (המשך)

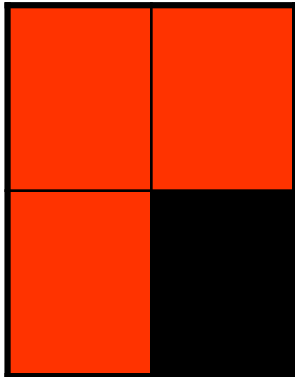
נסתכל על מקרה יותר קטן של הבעיה, עבור לוח בגודל $2^2 * 2^2$



האם עכשיו הפתרון ברור?

מילוי לוח משבצות (המשך)

נסתכל על מקרה יותר קטן של הבעיה, עבור לוח בגודל $2^1 * 2^1$



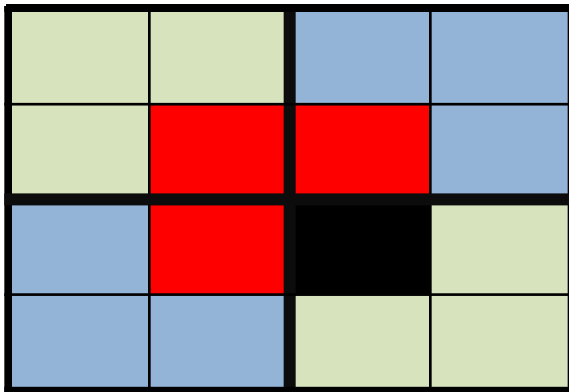
האם עכשיו הפתרון ברור?

כן! עבור לוח בגודל $2^1 * 2^1$ פשוט נשים את החלק אדום על השטח הנותר.

כלומר, עבור לוח בגודל $2^1 * 2^1$ שכבר יש בו ריבוע צבוע אחד אנחנו יודעים לפתור את הבעיה.

מילוי לוח משבצות (המשך)

כעת נחזור למקרה היותר גדול של הבעיה, עבור לוח בגודל $2^2 * 2^2$

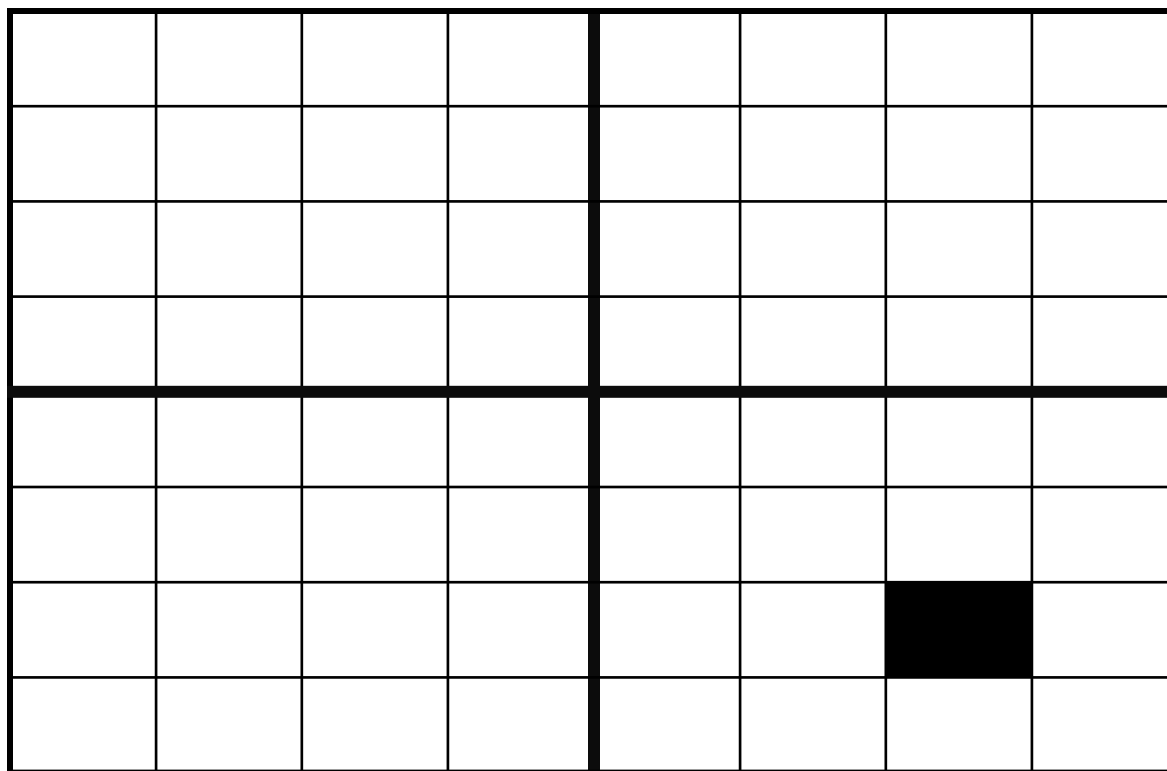


אנחנו יודעים שעבור לוח בגודל $2^1 * 2^1$ שיש בו ריבוע צבוע אחד אנחנו יודעים לפתור את הבעיה, לכן החלק הראשון שלנו ימוקם בדיוק במרכז, כך שכל ריבוע שלו יהיה חלק מרבע אחר של הלוח.

המצב שנוצר הוא שיש לנו 4 רבעים בגודל $2^1 * 2^1$ שיש בהם ריבוע צבוע אחד, ואת המקרה הזה אנחנו כבר יודעים לפתור!

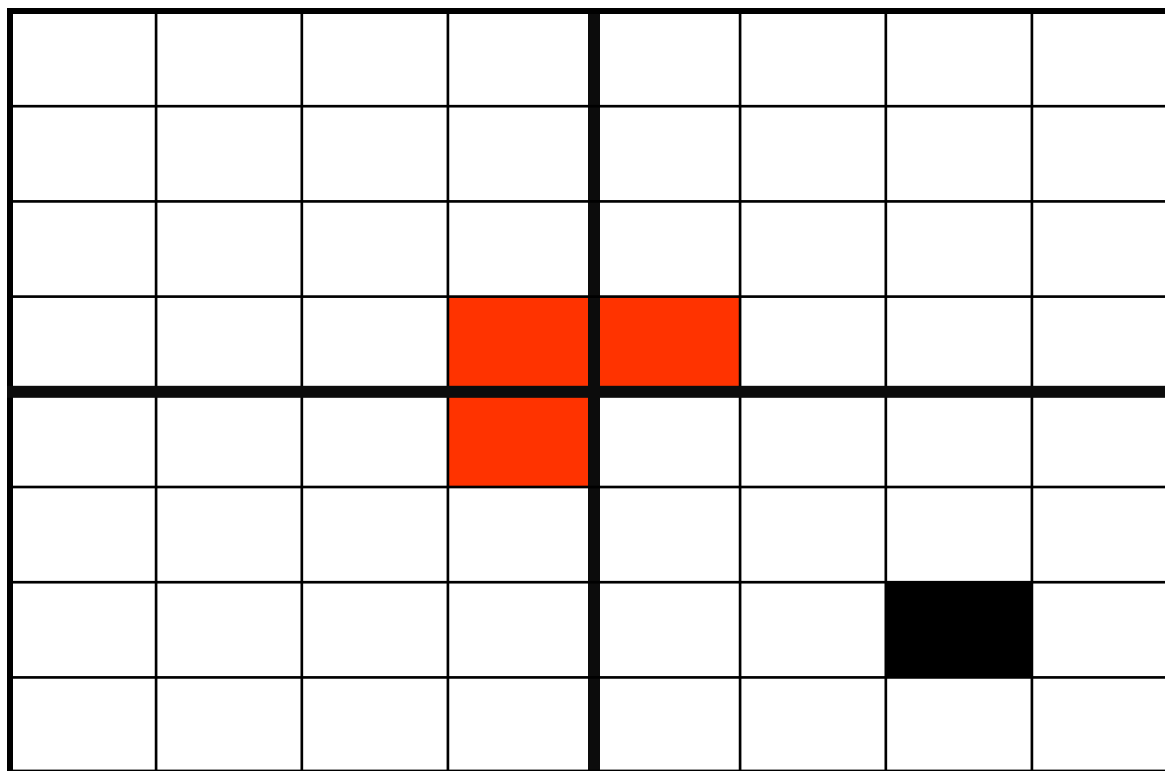
מילוי לוח משבצות (המשך)

ובאותו אופן נמלא לוח בגדול $2^3 * 2^3$



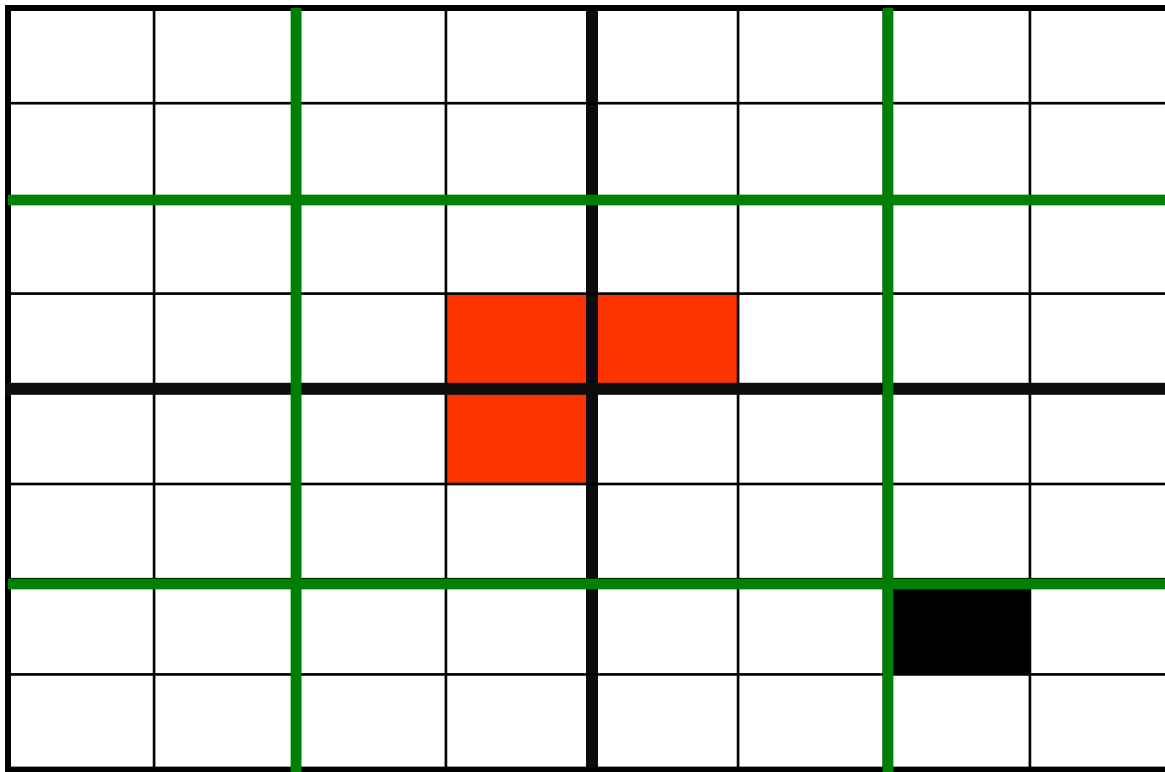
מילוי לוח משבצות (המשך)

ובאותו אופן נמלא לוח בגדול $2^3 * 2^3$



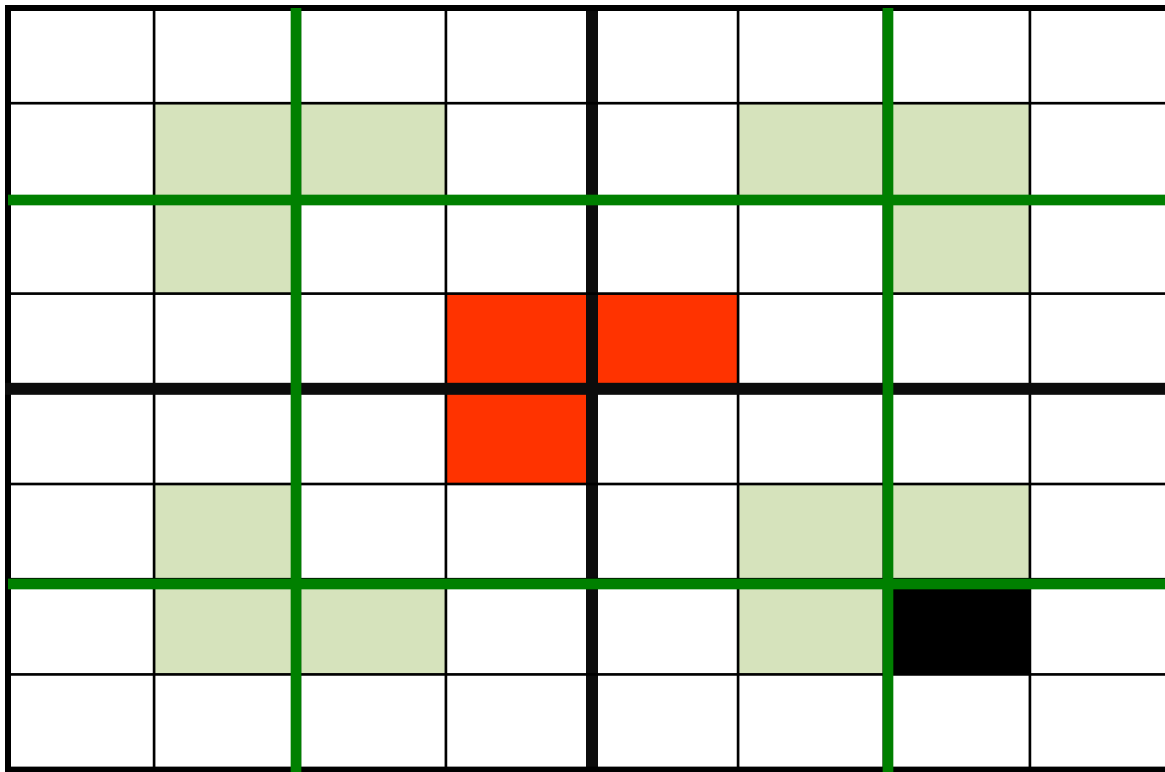
מילוי לוח משבצות (המשך)

ובאותו אופן נמלא לוח בגדול $2^3 * 2^3$



מילוי לוח משבצות (המשך)

ובאותו אופן נמלא לוח בגדול $2^3 * 2^3$



מילוי לוח משבצות - סיכום

□ הבעיה הגדולה הייתה קשה, ולכן בדקנו האם אנחנו יודעים לפתור מקרה יותר קטן של הבעיה. לאחר שפתרנו מקרה קטן יותר, בדקנו כיצד אנחנו מקשרים את גודל הבעיה הרצוי עם גודל הבעיה הקטנה יותר

□ כלומר, אם הייתה לנו השיטה:

```
void fillBoard(Color board[][], int size 8)
```

■ היא הייתה שמה את החלק הראשון במרכז, וקוראת לעצמה 4 פעמים, פעם עבור כל רבע

□ בכל קריאה אנחנו מניחים שהשיטה יודעת למלא את הלוח

□ פתרון כזה נקרא **רקורסיבי**, והוא יותר פשוט מפתרון איטרטיבי

רקורסיות ומערכים

□ רקורסיות יכולות לשמש אותנו גם בפתרון בעיות עם מערכים

□ הרעיון:

■ תנאי העצירה הוא מערך בגודל 1 או 0, ויש לדעת מה הפתרון עבורו

■ הרקורסיה יודעת להחזיר את הפתרון עבור מערך הקטן באורכו באיבר אחד (לרוב)

■ לנו נותר לקשר את הפתרון על המערך הקטן יותר עם הפתרון לגודל הנוכחי

□ כדי שהרקורסיה תפתור את הבעיה עבור מערך קטן יותר, עליה לקבל את גודל המערך המבוקש (אין לנו דרך יעילה להקטין את המערך)

דוגמא: חישוב סכום איבריו של מערך

□ חישוב סכום איבריו של מערך עם n איברים היא

$$\sum_{i=0}^{n-1} arr[i]$$

□ דרך נוספת לראות חישוב זה:

$$\sum_{i=0}^{n-1} arr[i] = (\sum_{i=0}^{n-2} arr[i]) + arr[n-1]$$

□ יש לזהות מהו הפתרון לבעיה הקטנה ביותר:

■ $n=0$ הפתרון הוא 0

□ ומכאן הדרך לקוד מאוד פשוטה:

```
int sumArr(int arr[], int n)
{
    if (n==0) return 0;
    return sumArr(arr, n-1) + arr[n-1];
}
```


דוגמא: חישוב סכום איבריו של מערך – מחסנית

הקריאות

```
int sumArr(int arr[], int n)
{
    if (n==0)
        return 0;
    return sumArr(arr, n-1) + arr[n-1];
}
```

```
void main()
{
    int arr[] = {1,6,2};
    sumArr(arr, 3);
    ...
}
```

arr={6,8,5}, n=0

```
int sumArr(int arr[], int n)
{
    if (n==0)
    return 0;
    return sumArr(arr, n-1) + arr[n-1];
}
```

arr={6,8,5}, n=3

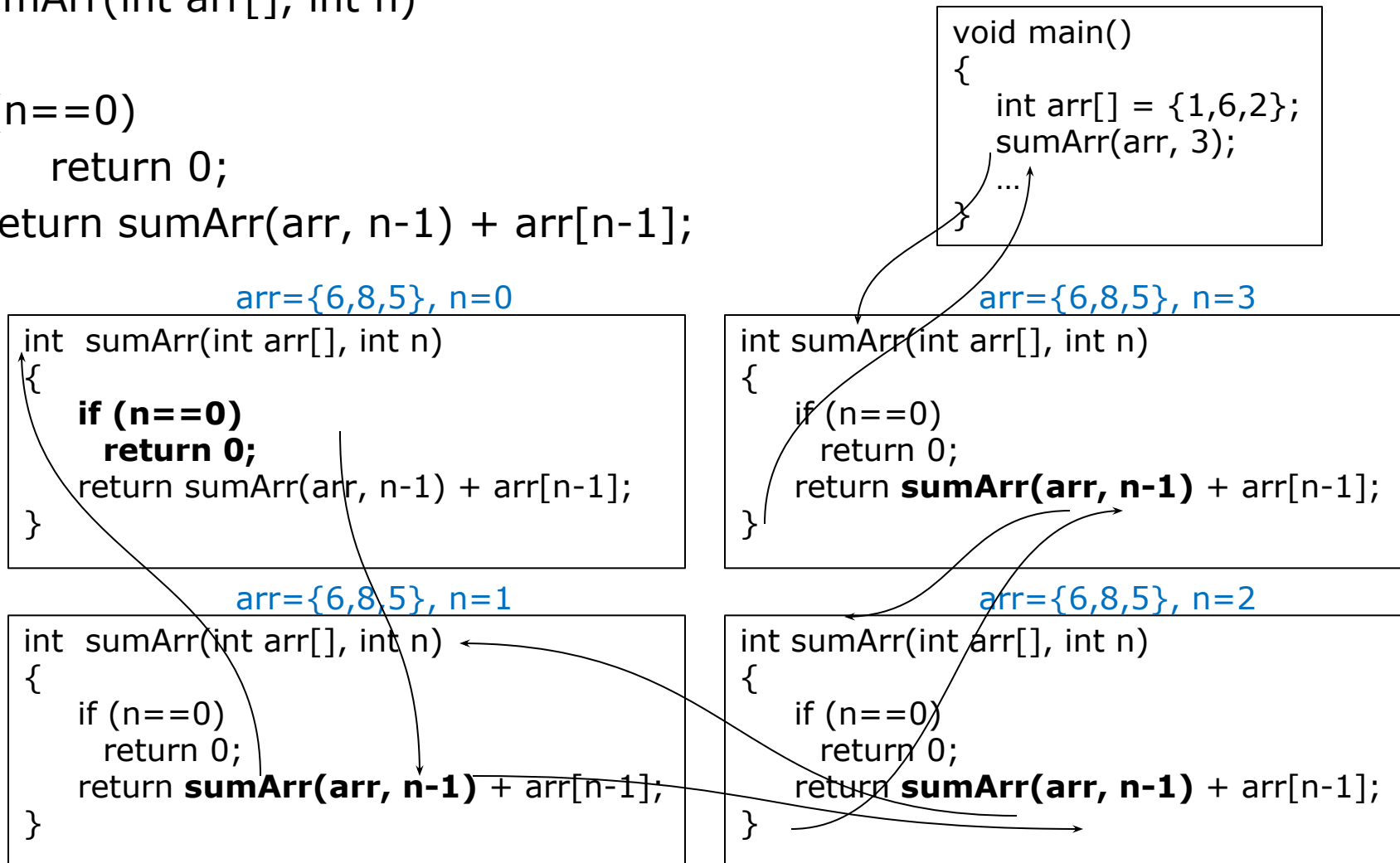
```
int sumArr(int arr[], int n)
{
    if (n==0)
        return 0;
    return sumArr(arr, n-1) + arr[n-1];
}
```

arr={6,8,5}, n=1

```
int sumArr(int arr[], int n)
{
    if (n==0)
        return 0;
    return sumArr(arr, n-1) + arr[n-1];
}
```

arr={6,8,5}, n=2

```
int sumArr(int arr[], int n)
{
    if (n==0)
        return 0;
    return sumArr(arr, n-1) + arr[n-1];
}
```



דוגמא: חישוב סכום איבריו של מערך – מחסנית

הקריאות (2)

```
int sumArr(int arr[], int n)
```

```
{  
    if (n==0)  
        return 0;  
    return sumArr(arr, n-1) + arr[n-1];  
}
```

- sum({6,8,5}, 3)
- sum({6,8,5}, 2)
- sum({6,8,5}, 1)
- sum({6,8,5}, 0)
- 0

- + arr[0]

- + arr[1]

- + arr[2]

דוגמא: חישוב סכום איבריו של מערך – ניתוח זמן ריצה

- הפונקציה `sumArray` עושה פעולה בסיסית (חיבור) שלוקחת $O(1)$ וכן קוראת לפונקציה `sumArray`
- בסה"כ נבצע n קריאות לפונקציה `sumArray`, שבכל קריאה מבצעים $O(1)$ פעולות
 - לכן זמן הריצה של `sumArray` הוא $O(n)$
- זמן הריצה של הגרסא הרקורסיבית זהה לזמן הריצה של הגירסא האיטרטיבית: $O(n)$

דוגמא: מציאת מקסימום במערך

מציאת האיבר המקסימלי במערך היא הפונקציה: `max(arr, size)`



דרך נוספת לראות חישוב זה:

■ אם `arr[size-1] > max(arr, size-1)` המקסימום הוא `arr[size-1]`

■ אחרת המקסימום הוא `max(arr, size-1)`



יש לזהות מהו הפתרון לבעיה הקטנה ביותר:

■ `n=1` הפתרון הוא `arr[0]`

דוגמא: מציאת מקסימום במערך - הקוד

```
int max(int arr[], int n)
```

```
{
```

תנאי עצירה

```
    if (n==1)
```

```
        return arr[0];
```

קריאה רקורסיבית

```
    int tempMax = max(arr, n-1);
```

- max({6,8,5}, 3)
- max({6,8,5}, 2)
- max({6,8,5}, 1)
- arr[0]

קישור

```
    if (arr[n-1] > tempMax)
```

```
        return arr[n-1];
```

- < arr[1]

```
    else
```

- < arr[2]

```
        return tempMax;
```

```
}
```

רקורסיות ופוינטרים

- ראינו שכאשר מעבירים מערך לפונקציה למעשה מועברת כתובת ההתחלה
- ראינו שניתן בצורה זו להתייחס לתת-מערך ע"י פניה לכתובת ההתחלה של איבר כלשהו במערך
 - למשל, כתובת ההתחלה של תת-המערך המתחיל באיבר השני היא $arr+1$
- ניתן להשתמש בטריק זה כדי לפתור רקורסיות

דוגמא – חישוב האיבר המקסימלי במערך

```
int max(int* arr, int size)
```

```
{
```

```
    int tempMax;
```

```
    if (size == 1)
```

```
        return arr[0];
```

```
    tempMax = max(arr+1, size-1);
```

```
    if (tempMax > arr[0])
```

```
        return tempMax;
```

```
    else
```

```
        return arr[0];
```

```
}
```

- max({6,8,5}, 3)

- max({8,5}, 2)

- max({5}, 1)

- arr[0]

- > arr[0]

- > arr[0]

דוגמא – האם מערך סימטרי - גירסא 1

```
int isSymetric(int* arr, int size)
{
    if (size <= 1)
        return 1;
    if (arr[0] != arr[size-1])
        return 0;
    return isSymetric(arr+1, size-2);
}

void main()
{
    int arr[] = {1,2,3,2,1};
    int size = sizeof(arr)/sizeof(arr[0]);
    int res = isSymetric(arr, size);
    if (res)
        printf("The array is symetric\n");
    else
        printf("The array is NOT symetric\n");
}
```

- isSym(1000, 5)
- isSym(1004, 3)
- isSym(1008, 1)
- 1

int: arr[]	1	1000
	2	1004
	3	1008
	2	1012
	1	1016
int: size	5	1020
int: res	???	1024

דוגמא – האם מערך סימטרי - גירסא 2 (1)

```
int isSymetric(int* begin, int* end)
{
    if (begin <= end)
        return 1;

    if (*begin != *end)
        return 0;

    return isSymetric(begin+1, end-1);
}

void main()
{
    int arr[] = {1,2,3,2,1};
    int size = sizeof(arr)/sizeof(arr[0]);
    int res = isSymetric(arr, arr+size-1);
    if (res)
        printf("The array is symetric\n");
    else
        printf("The array is NOT symetric\n");
}
```

- isSym(1000, 1016)
- isSym(1004, 1012)
- isSym(1008, 1008)
- 1

int: arr[]	1	1000
	2	1004
	3	1008
	2	1012
	1	1016
int: size	5	1020
int: res	???	1024

דוגמא – האם מערך סימטרי

– גירסא 2 (2)

```
int isSymetric(int* begin, int* end)
{
    if (begin <= end)
        return 1;

    if (*begin != *end)
        return 0;

    return isSymetric(begin+1, end-1);
}

void main()
{
    int arr[] = {1,2,2,1};
    int size = sizeof(arr)/sizeof(arr[0]);
    int res = isSymetric(arr, arr+size-1);
    if (res)
        printf("The array is symetric\n");
    else
        printf("The array is NOT symetric\n");
}
```

- isSym(1000, 1012)
- isSym(1004, 1008)
- isSym(1008, 1004)
- 1

int: arr[]	1	1000
	2	1004
	2	1008
	1	1012
int: size	5	1016
int: res	???	1020

בעיית sub-set sum

- נתון מערך, גודלו ומספר
- יש להחזיר TRUE במידה ויש תת-קבוצה במערך שסכומה הוא כמספר שהתקבל, FALSE אחרת.

□ דוגמאות:

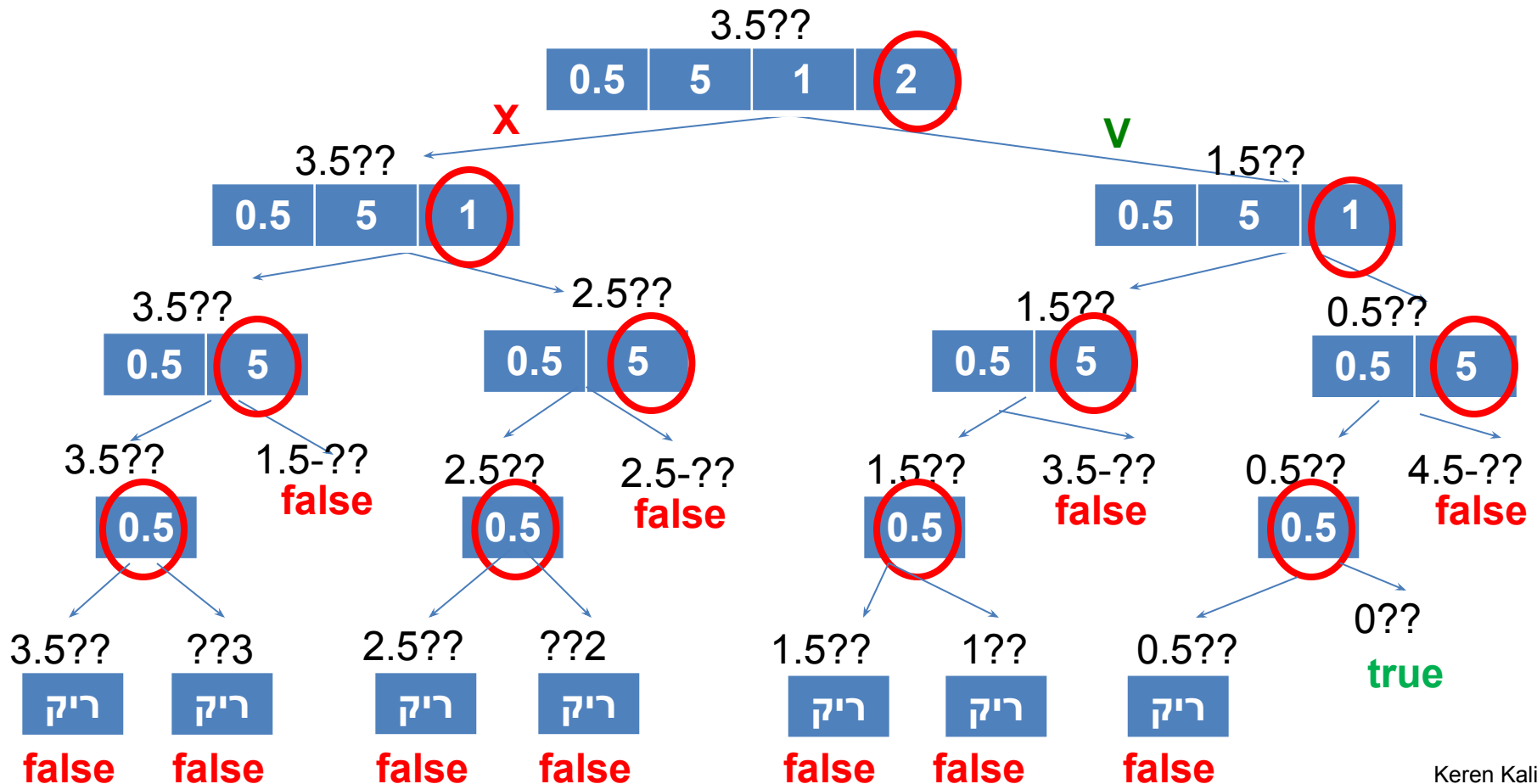
- עבור המערך $\{1,3,5\}$ והמספר 6 יוחזר TRUE מאחר ו-
 $1+5=6$
- עבור המערך $\{1,3,5\}$ והמספר 9 יוחזר TRUE מאחר ו-
 $1+3+5=9$
- עבור המערך $\{1,3,5\}$ והמספר 7 יוחזר FALSE

בעיית sub-set sum - הרעיון

- כל איבר במערך או שהוא חלק מהסכום המבוקש, או שאינו
- הבעיה היותר קטנה היא כל המערך פרט לאיבר האחרון:
 - כאשר אם האיבר האחרון חלק מהסכום יש לחפש בשאר איברי המערך את הסכום פחות ערכו של האיבר האחרון
 - אם האיבר האחרון אינו חלק מהסכום יש לחפש בתת המערך את הסכום במלואו
 - דוגמא: המערך $\{1,3,5\}$
- הבעיה הכי קטנה:
 - או כאשר מחפשים את הסכום 0, ואז התשובה היא TRUE
 - או כאשר גודל המערך הוא 0 והסכום לחיפוש גדול מ-0 ואז התשובה היא FALSE

בעיית sub-set sum - דוגמה

בארנק בו יש את המטבעות 5, 2, 1 ו-0.5, האם ישנה קומבינציה שמרכיבה בדיוק 3.5 ₪?



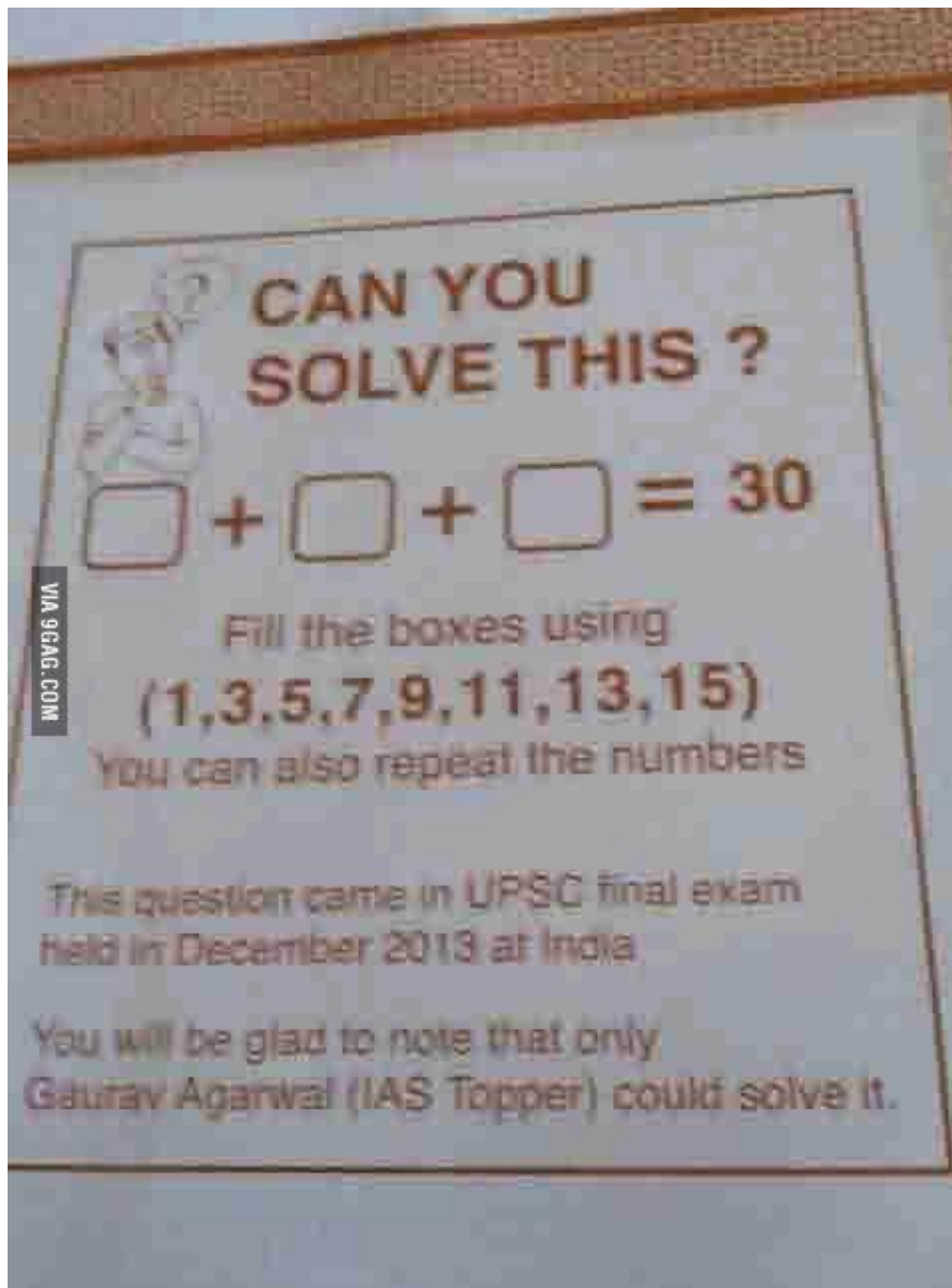
בעיית sub-set sum - הפתרון

```
int subsetSum(int arr[], int size, int sum)
{
    if (sum == 0)
        return 1;

    if (size == 0 || sum < 0)
        return 0;

    return subsetSum(arr, size-1, sum-arr[size-1]) ||
           subsetSum(arr, size-1, sum);
}
```

האם sub-set sum פותר את השאלה הבאה?



CAN YOU SOLVE THIS ?

$\square + \square + \square = 30$

Fill the boxes using
(1,3,5,7,9,11,13,15)
You can also repeat the numbers

This question came in UPSC final exam
held in December 2013 at India

You will be glad to note that only
Gaurav Agarwal (IAS Topper) could solve it.

VIA 9GAG.COM

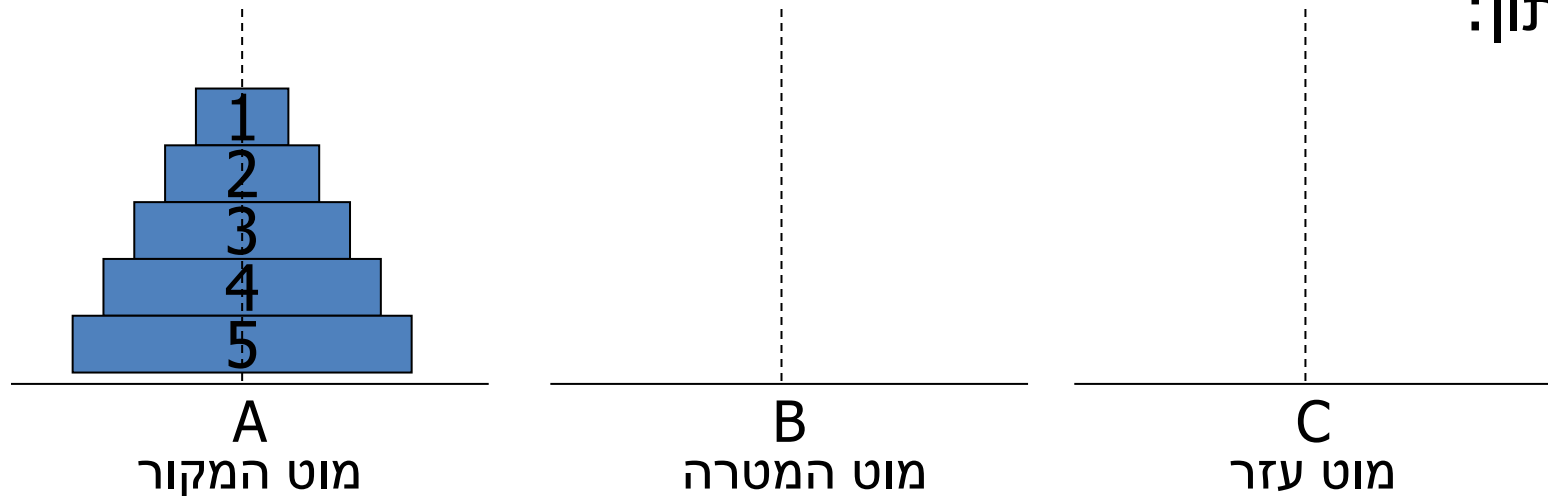
משולש סרפינסקי

- מתחילים ממשולש שווה-צלעות, וחותכים ממנו את המשולש המרכזי, כמו שנראה באיור
- עתה נוצרו שלושה משולשים מלאים קטנים
- בשלב הבא לכל אחד מהם חותכים את המשולש האמצעי
- כל שלב נקרא איטרציה ולאחר אין-סוף איטרציות נוצר



מגדלי הנוי – הצגת הבעיה

נתון: □



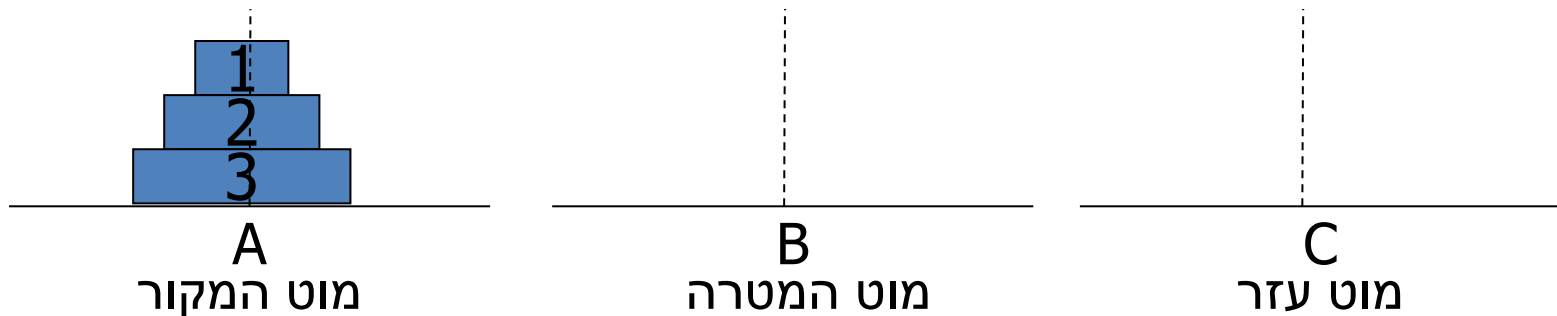
המטרה: להעביר את החישוקים ממוט A למוט B תוך שימוש במוט C, לפי הכללים הבאים: □

- בכל צעד מותר להעביר חישוק בודד ממוט למוט
- אסור מצב שבו חישוק גדול מונח מעל חישוק קטן
- בכל רגע נתון חישוק חייב להיות מונח על אחד המוטות

מגדלי הנוי - פתרון

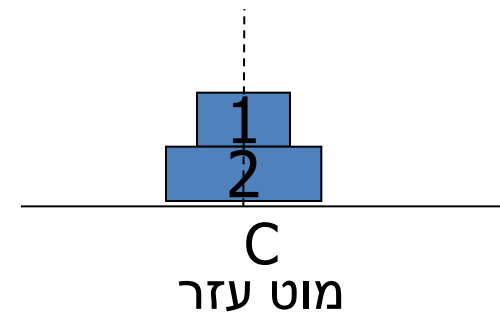
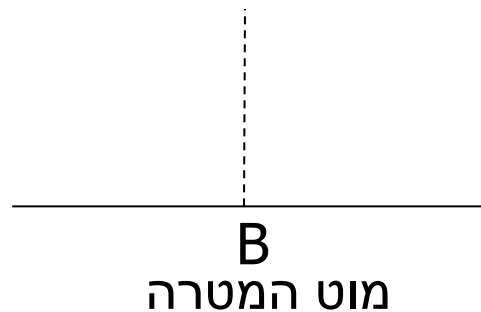
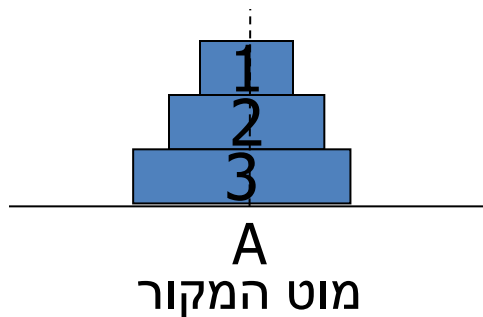
□ נניח כי ידוע כיצד מעבירים $n-1$ חישוקים ממגדל מקור למגדל יעד תוך שימוש במגדל עזר

□ בסיס הרקורסיה: כאשר $n=0$, לא עושים דבר



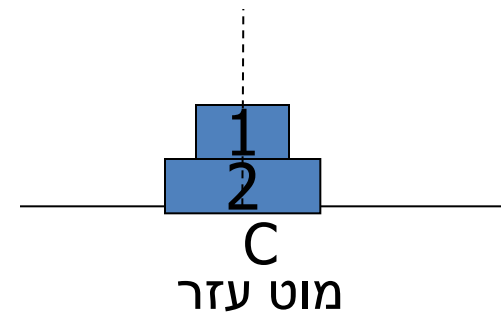
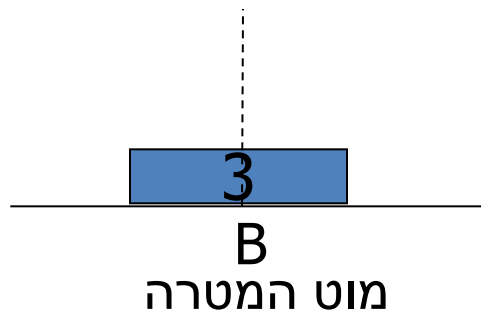
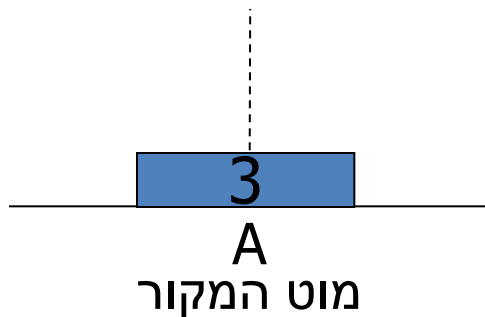
מגדלי הנוי - פתרון

- נעביר את החישוקים 1, 2, ..., $n-1$ ממגדל A למגדל C תוך שימוש במגדל B



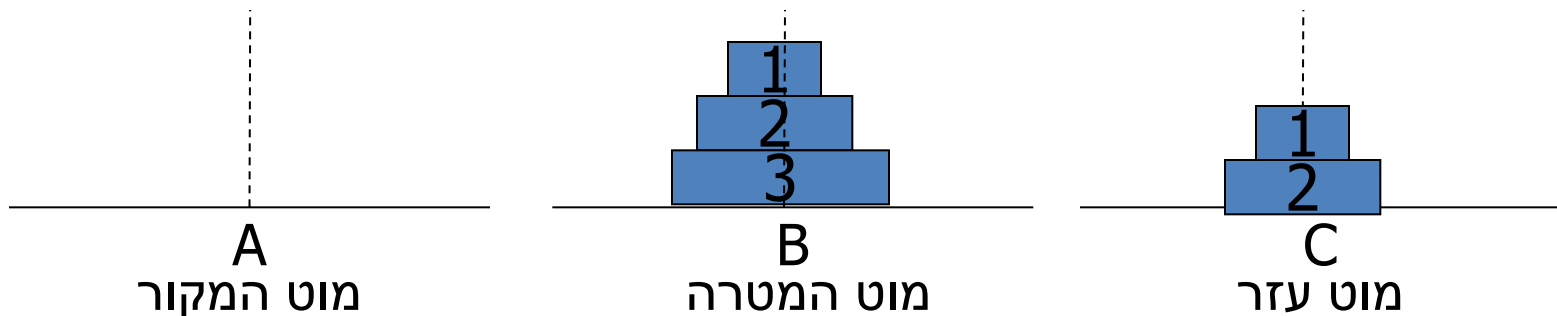
מגדלי הנוי - פתרון

- נעביר את החישוקים 1, 2, ..., $n-1$ ממגדל A למגדל C תוך שימוש במגדל B
- נעביר את חישוק n ממגדל A למגדל B



מגדלי הנוי - פתרון

- נעביר את החישוקים 1, 2, ..., $n-1$ ממגדל A למגדל C תוך שימוש במגדל B
- נעביר את חישוק n ממגדל A למגדל B
- נעביר את החישוקים 1, 2, ..., $n-1$ ממגדל C למגדל B תוך שימוש במגדל A



מגדלי הנוי – דוגמאות פלט

How many discs? 0

How many discs? 1

Move disc 1 from A to B

How many discs? 2

Move disc 1 from A to C

Move disc 2 from A to B

Move disc 1 from C to B

How many discs? 3

Move disc 1 from A to B

Move disc 2 from A to C

Move disc 1 from B to C

Move disc 3 from A to B

Move disc 1 from C to A

Move disc 2 from C to B

Move disc 1 from A to B

How many discs? 4

Move disc 1 from A to C

Move disc 2 from A to B

Move disc 1 from C to B

Move disc 3 from A to C

Move disc 1 from B to A

Move disc 2 from B to C

Move disc 1 from A to C

Move disc 4 from A to B

Move disc 1 from C to B

Move disc 2 from C to A

Move disc 1 from B to A

Move disc 3 from C to B

Move disc 1 from A to C

Move disc 2 from A to B

Move disc 1 from C to B

מגדלי הנוי - הקוד

```
void hanoi(int n, char src, char dest, char help)
{
    if (n == 0) return;
    hanoi(n-1, src, help, dest);
    printf("Move disc %d from %c to %c\n",
        n, src, dest);
    hanoi(n-1, help, dest, src);
}
```

תנאי עצירה

קריאה רקורסיבית

קישור

קריאה רקורסיבית

```
void main()
{
    int discs;

    printf("How many discs? ");
    scanf("%d", &discs);
    hanoi(discs, 'A', 'B', 'C');
}
```

מגדלי הנוי –

עץ הקריאות עבור

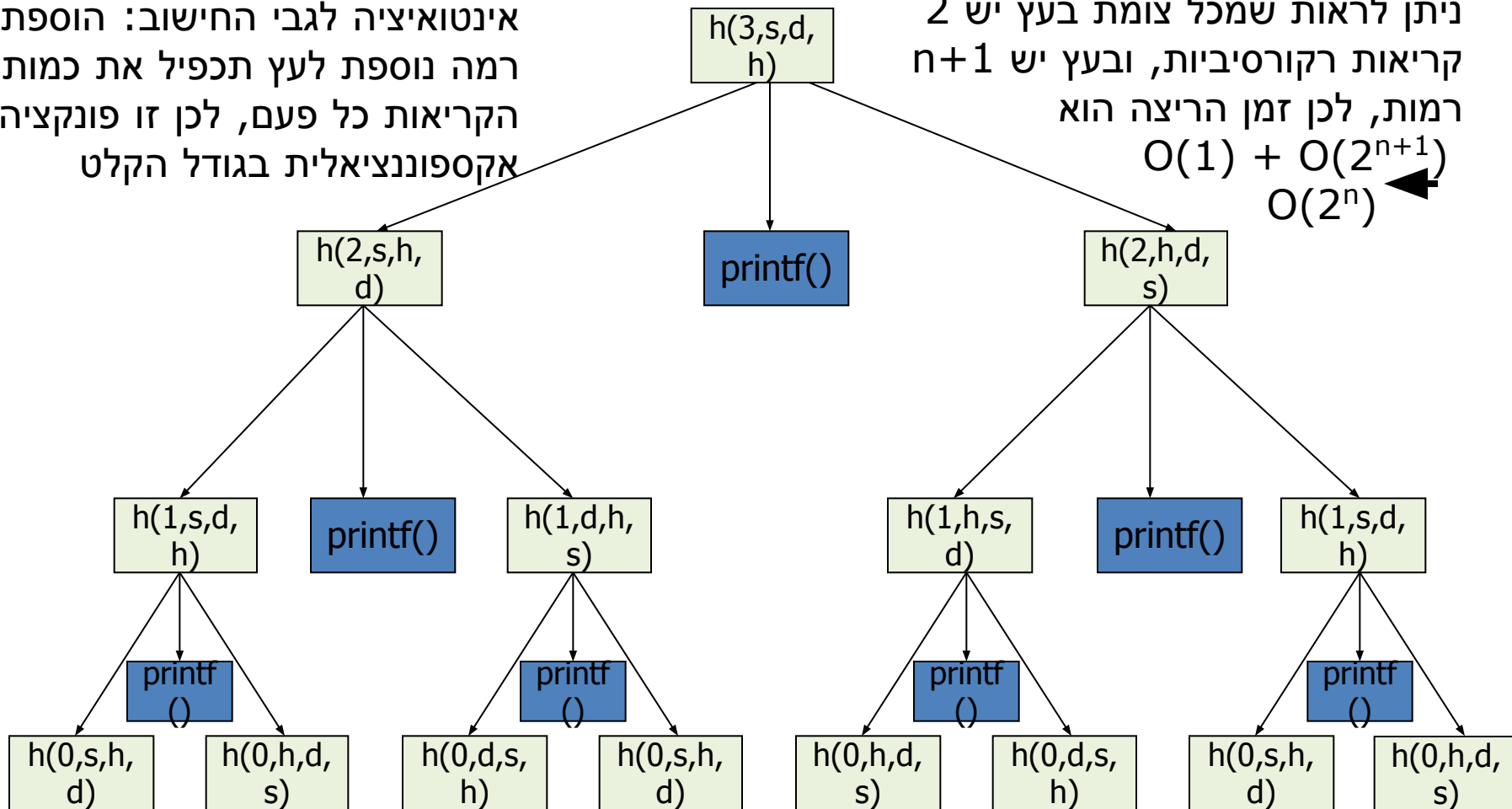
בעיה בגודל 3

```
void hanoi(int n, char src, char dest, char help)
```

```
{  
    if (n == 0) return;  
  
    hanoi(n-1, src, help, dest);  
    printf("Move disc %d from %c to %c\n", n, src, dest);  
    hanoi(n-1, help, dest, src);  
}
```

אינטואיציה לגבי החישוב: הוספת רמה נוספת לעץ תכפיל את כמות הקריאות כל פעם, לכן זו פונקציה אקספוננציאלית בגודל הקלט

ניתן לראות שמכל צומת בעץ יש 2 קריאות רקורסיביות, ובעץ יש $n+1$ רמות, לכן זמן הריצה הוא $O(1) + O(2^{n+1})$
 $O(2^n)$



חשוב לזכור בעת פתרון רקורסיבי:

□ רקורסיה מוצלחת מורכבת מ- 3 חלקים:

- תנאי עצירה
- קריאה רקורסיבית
- קישור

□ נורות אדומות בפתרון:

- שימוש במשתנים גלובלים או סטטיים
- אי שימוש בערך המוחזר מהרקורסיה (במקרה שהפונקציה אינה void)
- פונקציה רקורסיבית היא פונקציה שקוראת לעצמה, ולא פונקציה הקוראת לפונקציות רקורסיביות אחרות

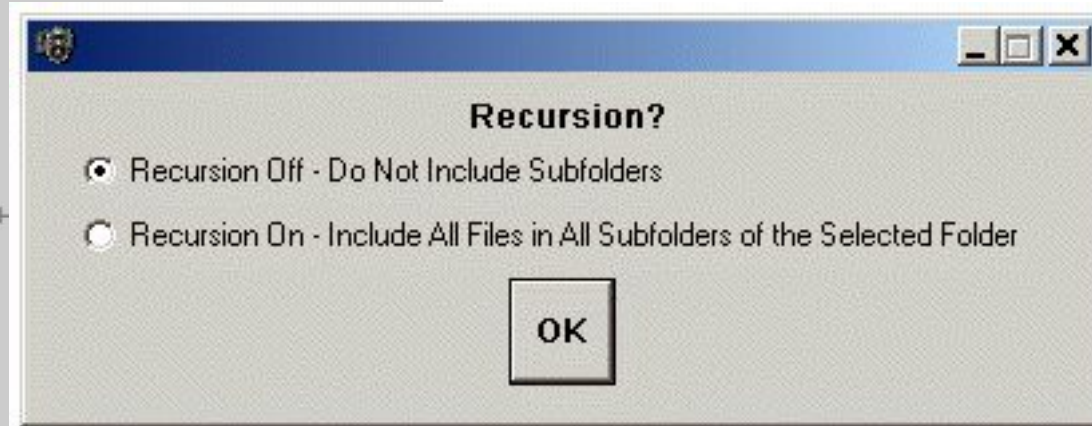
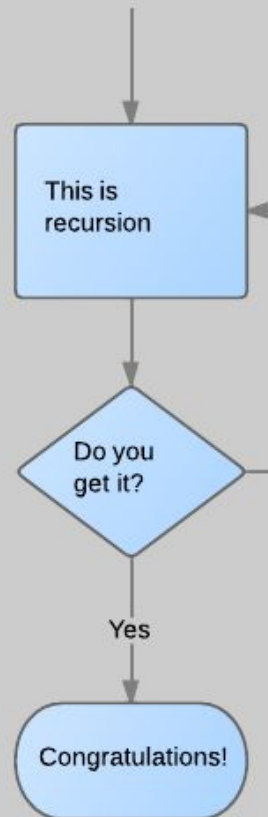
ויש סיכוי שתתקלו במחשבות הבאות...



http://doblelol.com/thumbs/funny-cartoon-pictures-english-programmer_5041094350342269.jpg

Learn Recursion

ולסיום..



<http://home.roadrunner.com/~dick/Recursion.gif>

<https://www.lucidchart.com/documents/thumb/4d4c1d6c-4f28-4436-a32e-53c60ac17605/0/190994/NULL/690/NULL/Learn-Recursion.png>

ביחידה זו למדנו:

- מהי רקורסיה
- מרכיבי הרקורסיה
- ניתוח זמן ריצה
- רקורסיות ומערכים
- רקורסיות ומצביעים

תרגיל 1: סכום ספרותיו של מספר

□ כתוב פונקציה רקורסיבית המקבלת מספר ומחזירה את סכום ספרותיו

□ דוגמאות:

- עבור המספר 123 יוחזר 6
- עבור המספר 9834 יוחזר 24

תרגיל 2: האם ספרות המספר בסדר עולה

□ כתוב פונקציה רקורסיבית המקבלת מספר ומחזירה 1 אם ספרותיו מסודרות בסדר עולה, ו- 0 אחרת

□ דוגמאות:

- עבור המספר 123 יוחזר 1
- עבור המספר 9834 יוחזר 0
- עבור המספר 231 יוחזר 0
- עבור המספר 2589 יוחזר 1

תרגיל 3: כמה פעמים מופיע מספר במערך

□ כתוב פונקציה רקורסיבית המקבלת מערך, גודלו ומספר. הפונקציה תחזיר כמה פעמים מופיע המספר במערך

□ דוגמאות:

■ עבור המערך **23,65,43,23,87** גודלו 5 והמספר 23 יוחזר 2

■ עבור במערך **3,2,6,3,8,3,3** גודלו 7 והמספר 3 יוחזר 4

תרגיל 4: האם תווי המחרוזת זהים

□ כתוב פונקציה רקורסיבית המקבלת מחרוזת. הפונקציה תחזיר 1 אם כל תווי המחרוזת זהים ו- 0 אחרת

□ דוגמאות:

- עבור המחרוזת aa יוחזר 1
- עבור המחרוזת aaAa יוחזר 0
- עבור המחרוזת aaba יוחזר 0
- עבור המחרוזת 3333 יוחזר 1

תרגיל 5: ציור ריבוע

□ כתוב פונקציה רקורסיבית המקבלת אורך צלע של ריבוע וצייר את הריבוע

□ דוגמא:

■ עבור צלע באורך 4 יש לצייר את הריבוע:

```
* * * *  
* * * *  
* * * *  
* * * *
```