אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, חורף 2022

תרגיל בית מספר 4 - להגשה עד 12/12/2021 בשעה 23:55

קראו בעיון את הנחיות העבודה וההגשה המופיעות באתר הקורס, תחת התיקייה assignments. חריגה מההנחיות תגרור ירידת ציון / פסילת התרגיל.

הגשה:

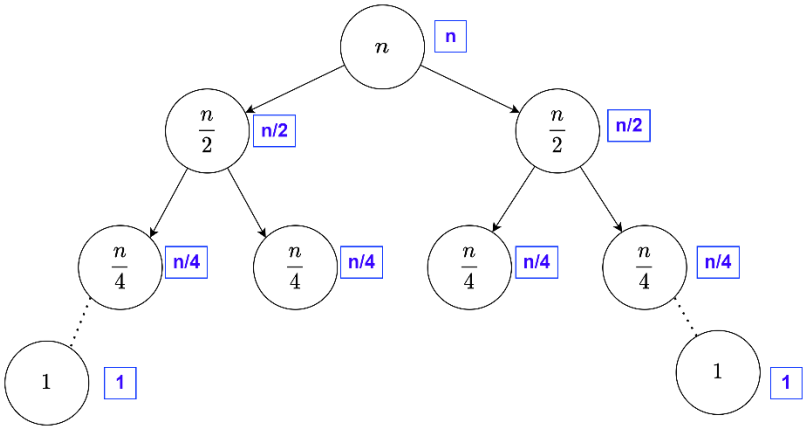
- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- התשובות בקובץ ה pdf חייבות להיות מוקלדות **ולא** בכתב יד.
- השתמשו בקובץ השלד skeleton4.py כבסיס לקובץ ה py אותו אתם מגישים.
לא לשכוח לשנות את שם הקובץ למספר ת"ז שלכם לפני ההגשה, עם סיומת py.

- בסה"כ מגישים שני קבצים בלבד. עבור סטודנטית שמספר ת"ז שלה הוא 012345678 הקבצים שיש להגיש הם hw4_012345678.pdf ו- hw4_012345678.py.
- הקפידו לענות על כל מה שנשאלתם.

- תשובות מילוליות והסברים צריכים להיות תמציתיים, קולעים וברורים.
להנחיה זו מטרה כפולה:
  i. על מנת שנוכל לבדוק את התרגילים שלכם בזמן סביר.
  ii. כדי להרגיל אתכם להבעת טיעונים באופן מתומצת ויעיל, ללא פרטים חסרים מצד אחד אך ללא עודף בלתי הכרחי מצד שני. זוהי פרקטיקה חשובה במדעי המחשב.

הערות כלליות לתרגיל זה ולתרגילים הבאים:

כאשר אתם מתבקשים לצייר עץ רקורסיה, מומלץ להיעזר בכלים דיגיטליים כמו draw.io כדי לצייר את העץ – זהו כלי מאוד אינטואיטיבי ליצירת תרשימים, ויכול לעזור לכם גם בהמשך התואר (או בחיים). עם זאת, ניתן לצייר את עצי הרקורסיה בכתב יד ולסרוק, כל עוד הציור ברור לחלוטין. **ציור שאינו ברור לא ייבדק.**
בציורכם הקפידו על הדברים הבאים: כל צומת בעץ מייצג קריאה רקורסיבית – בתוך הצומת כתבו את הקלט, או את אורך הקלט, המתאים לצומת זה, ולצד כל צומת כתבו את כמות העבודה שמתבצעת בצומת. לדוגמה, את עץ הרקורסיה עבור המקרה הטוב ביותר של Quicksort (כפי שראיתם בהרצאה) נצייר באופן הבא:



שאלה 1

בשאלה זו כל סעיף יעסוק בסוגיה הקשורה לקוד שראיתם בהרצאה או בתרגול. מומלץ לעבור על החומר הרלוונטי מההרצאה / תרגול לפני שאתם ניגשים לפתור את הסעיף.

א. מקסימום של רשימה (תרגול 6)
ראיתם בתרגול שני מימושים רקורסיביים למציאת מקסימום של רשימה. במימושים אלו השתמשנו במשתנים left, right כדי להימנע מפעולות slicing על הרשימה. לפניכם שני קטעי קוד שקולים לאלו הראיתם בתרגול, פרט לכך שהם משתמשים ב-slicing. נסמן ב-n את אורך רשימת הקלט L. עבור כל אחד מהמימושים הבאים, ציירו באופן סכמטי את עץ הרקורסיה המתאים, ונתחו את סיבוכיות זמן הריצה כתלות ב-n.

i.

```
def max_v1(L):
    if len(L) == 1:
        return L[0]

    without_left = max_v1(L[1:])

    return max(without_left, L[0])
```

ii.

```
def max_v2(L):
    if len(L) == 1:
        return L[0]

    mid = len(L) // 2
    first_half = max_v2(L[:mid])
    second_half = max_v2(L[mid:])

    return max(first_half, second_half)
```

ב. <u>סדרת פיבונאצ'י עם ממואיזציה (הרצאה 12)</u>

ראיתם בהרצאה מימוש רקורסיבי למציאת האיבר ה-n בסדרת פיבונאצ'י, המשתמש בממואיזציה. להלן מימוש אלטרנטיבי לאותה הבעיה:

```
def fib2_reverse(n, fib_dict):
    if n not in fib_dict:
        res = fib2_reverse(n-2, fib_dict) + fib2_reverse(n-1, fib_dict)
        fib_dict[n] = res
    return fib_dict[n]
```

ופונקציית המעטפת:

```
def fibonacci2_reverse(n):
    fib_dict = {0:1, 1:1} # initial dictionary
    return fib2_reverse(n, fib_dict)
```

שימו לב שההבדל בין מימוש זה לבין המימוש שראיתם בהרצאה הוא הסדר בין שתי הקריאות הרקורסיביות.

i. ציירו באופן סכמטי את עץ הרקורסיה המתאים לגרסה זו (הניחו לשם פשטות כי n זוגי).

ii. מהו עומק הרקורסיה, כתלות ב-n? האם הרצה עם קלט שגדול מ-1000 מצליחה? הסבירו.

iii. נתחו את סיבוכיות זמן הריצה בעזרת עץ הרקורסיה, כתלות ב-n.

ג. <u>המשחק זלול! (munch) (הרצאה 12)</u>

i. לפניכם וריאציה של הקוד שראיתם בכיתה שמחזיר True אם הלוח board במצב מנצח של משחק munch, או False אם הלוח במצב מפסיד. בקוד מספר שינויים המודגשים בצהוב.
<u>הסבירו מה יודפס</u> כאשר נריץ את הפקודה הבאה:

winnable([2,1,1], show=True, player=0)

<u>ציירו</u> את עץ הרקורסיה במלואו עבור הרצה זו, **<u>וסמנו על העץ</u>** את הצמתים שבהם יתבצעו ההדפסות.

```
def winnable(board, show=False, player=0):
    if sum(board) == 0:
        return True

    m = len(board)

    for i in range(m):
        for j in range(board[i]):
            munched_board = board[0:i] + [min(board[k], j) for k in range(i, m)]

            if not winnable(munched_board, show, 1-player):
                if show and player == 0:
                    print("recommended move:", board, "-->", munched_board)
                return True

    return False
```

ii. בסעיף זה תוסיפו ממואיזציה לקוד שראיתם בכיתה (ולא לקוד מסעיף i). בקובץ השלד ממומשת עבורכם פונקציית המעטפת winnable_mem(board) המאתחלת מילון ריק ומבצעת קריאה לפונקציה winnable_mem_rec.
ממשו את הפונקציה הרקורסיבית winnable_mem_rec(board, d), כך שבהינתן רשימה board המייצגת לוח חוקי של מאנץ', פונקציית המעטפת תחזיר True אם הלוח במצב מנצח, או False אם הלוח במצב מפסיד. על הפונקציה להשתמש במילון d כדי לחסוך קריאות רקורסיביות חוזרות על קלטים שחושבו קודם לכן (כלומר, על הפונקציה להשתמש בממואיזציה). כמו כן, שימו לב <u>לא לבצע הדפסות</u> בשום שלב במימוש שלכם (בשונה מהקוד שראיתם בכיתה).
<u>תזכורת</u>: מפתחות במילון חייבים להיות אובייקטים מסוג immutable. בפרט, לא ניתן להשתמש באובייקט מטיפוס רשימה כמפתח במילון (נבין מדוע המגבלה הזו קיימת בהמשך הקורס). ניתן להמיר את הרשימה לאובייקט אחר שהוא immutable (לדוגמה tuple), ולהשתמש בו כמפתח במילון.

iii. מדדו את זמני הריצה של הפונקציה שמימשתם בסעיף הקודם ואת זמני הריצה של הפונקציה ללא הממואיזציה שראיתם בכיתה, על הקלטים 2*[5], 4*[5], 8*[5], וכתבו את התוצאות בטבלה בקובץ ה-PDF.

iv. עבור הפונקציה שמימשתם, ועבור כל אחד מן הקלטים 4*[5] , 8*[5], 16*[5], כתבו בקובץ ה-PDF בטבלה:

1) כמה חיפושים בדיוק נעשו במילון בסך הכל לאורך החישוב של הפונקציה?
(כל פקודה מהצורה "key in d" נחשבת לחיפוש במילון, ללא תלות בהאם key נמצא במילון d או לא).
2) מתוך כל החיפושים – כמה חיפושים בדיוק הסתיימו בהצלחה? כלומר כמה פעמים החיפוש החזיר ערך שחושב קודם לכן.

שאלה 2

בשאלה זו נרצה לעבוד עם מטריצות. מאחר שאין בפייתון ערכים מובנים מטיפוס מטריצה, נשתמש ברשימות מקוננות (רשימות בתוך רשימות) על מנת לייצג מטריצות. הייצוג שלנו יהיה טבעי ביותר: בהינתן מטריצה M בגודל $n \times k$ (כלומר, מטריצה בת n שורות ו-k עמודות), נבנה רשימה מקוננת lst_m המכילה n תתי-רשימות, כל אחת באורך k.

כעת, את האיבר $M_{i,j}$ נמקם ב-$lst\_m[i][j]$. להלן דוגמא למטריצה בגודל 2×3:

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

והרשימה שתייצג את M היא הרשימה הבאה:

$$lst\_m = [[1, 2, 3], [4, 5, 6]]$$

נגדיר עתה מטריצה **בינארית** (כלומר, שערכיה הם 0 או 1) מעניינת במיוחד – מטריצת Hadamard. למספר טבעי n מטריצת Hadamard מסדר n (אותה נסמן מעתה ואילך על ידי $H(n)$) מוגדרת באופן הרקורסיבי הבא:

- עבור $n = 0$, $H(0)$ היא מטריצה מגודל 1×1 המכילה את האיבר אפס: $H(0) = (0)$
- עבור $n > 0$, $H(n)$ היא מטריצה ריבועית מגודל $2^n \times 2^n$ אשר בנויה באופן הרקורסיבי הבא:
  אם $H(n-1)$ היא מטריצת Hadamard מסדר $n-1$, אזי:

$$H(n) = \begin{pmatrix} H(n-1) & H(n-1) \\ H(n-1) & \overline{H(n-1)} \end{pmatrix}$$

כאשר עבור מטריצה בינארית M נסמן ב-$\overline{M}$ את המטריצה שבה מחליפים ערכי 0 ב-M ב-1 וערכי 1 ב-M ב-0.
כלומר, $H(n)$ מורכבת מ-4 עותקים של $H(n-1)$ (זוהי מטריצת בלוקים עם שני בלוקים עליונים של $H(n-1)$ ושני בלוקים תחתונים של $H(n-1)$), כאשר בבלוק הימני-תחתון ערכי המטריצה מתהפכים (כלומר, ערכי 0 הופכים ל-1 וערכי 1 ל-0).

למטריצת Hadamard התכונות המרתקות הבאות:

- כל שתי שורות שונות במטריצה נבדלות בדיוק בחצי מהקואורדינטות שלהן (בדקו זאת!). תכונה זו שימושית במיוחד בתחום של תיקון שגיאות, אותו נפגוש בהמשך הקורס.
- לכל n מתקיים כי במטריצה $H(n)$, כל שורה מלבד השורה העליונה מכילה מספר זהה של אפסים ואחדות.

להלן שלוש המטריצות $H(0), H(1), H(2)$ מיוצגות כרשימות מקוננות על פי הייצוג שקבענו לעיל:

```
H0 = [[0]]

H1 = [[0, 0], [0, 1]]

H2 = [[0, 0, 0, 0], [0, 1, 0, 1], [0, 0, 1, 1], [0, 1, 1, 0]]
```

שימו לב: בשאלה זו יש להתחשב בסיבוכיות זמן הריצה של פעולות אריתמטיות (כלומר, אין להניח כי פעולות אלו רצות זמן קבוע). <u>ניתן להניח כי החישוב של $pow(2, n)$ לוקח זמן $\Theta(n)$.</u>

א. בקובץ השלד השלימו את מימוש הפונקציה $H\_local(n, i, j)$ אשר מקבלת כקלט שלושה מספרים שלמים, $n \geq 0$ ו-$0 \leq i, j < 2^n$ ומחזירה את $H(n)_{i,j}$ (כלומר, את הכניסה בשורה ה-i והעמודה ה-j במטריצת Hadamard מסדר n).

<u>הנחיות</u>:

- על המימוש להיות רקורסיבי.
- יש לממש את הפונקציה בצורה יעילה ככל הניתן.
- בפרט, אין לייצר את כל המטריצה $H(n)$, אלא "להתביית" על הערך הרצוי בצורה חכמה.

דוגמאות הרצה:

```
>>> H_local(2, 1, 2)
0
>>> H_local(2, 2, 2)
1
>>> H_local(0, 0, 0)
0
>>> H_local(1, 1, 1)
```

ב. בקובץ ה-PDF ציינו מהו זמן הריצה של הפונקציה שמימשתם בסעיף הקודם במונחים אסימפטוטיים, כתלות ב-n. נמקו את תשובתכם.

ג. בקובץ השלד השלימו את מימוש פונקצית הלמבדא H_complete. הפונקציה מקבלת כקלט שלם אי-שלילי, $n \geq 0$ ומחזירה את המטריצה $H(n)$.
עליכם להשתמש בפונקציה H_local.
בסעיף זה אין דרישה לממש את הפונקציה באופן היעיל ביותר – כל פתרון נכון יתקבל.

שאלה 3

הנחיות לכל הסעיפים בשאלה זו:

- על הפונקציה שאתם מממשים להיות רקורסיבית, או להיות פונקציית מעטפת שקוראת לפונקציה רקורסיבית.
- ניתן להניח שפעולות אריתמטיות לוקחות זמן קבוע.

א. בהינתן רשימה L של מספרים שלמים וחיוביים, ומספר שלם s, נאמר ש**ניתן ליצור את s מ-L** אם ניתן להגיע ל-s על ידי חיבור וחיסור של איברי L.
בסעיף זה, נבדוק האם ניתן ליצור את s מ-L תחת ההגבלה שאנו משתמשים בכל איבר ב-L **בדיוק פעם אחת.**
לדוגמה, אם $L = [5,2,3]$ ו- $s = 6$ אז ניתן ליצור את s מ-L תחת ההגבלה, מכיוון ש-

$$5 - 2 + 3 = 6$$

כמו כן ניתן ליצור מ-L את $s = -10$ מכיוון ש-

$$-5 - 2 - 3 = -10$$

לעומת זאת, לא ניתן ליצור מ-L את $s = 9$ או את $s = 7$ תחת ההגבלה.

i. ממשו את הפונקציה can_create_once(s, L) אשר מחזירה True אם אפשר ליצור את s מ-L, ו-False אחרת, תחת הגבלה זו.
הנחיה: על הפונקציה להיות מסיבוכיות זמן $O(2^n)$ כאשר n הוא אורך הרשימה L. שימו לב שקוד שרץ בזמן $n \cdot 2^n$ אינו עונה על דרישה זו.

ii. **ציירו** את עץ הרקורסיה **והסבירו** מדוע הפונקציה שמימשתם עומדת בדרישת הסיבוכיות.

ב. בסעיף זה נממש פונקציה דומה לזו מסעיף א', אבל הפעם נרשה להשתמש בכל איבר ב-L **לכל היותר פעמיים.** לדוגמה, אם $L = [5,2,3]$ אז הפעם ניתן ליצור את $s = 9$ תחת הגבלה זו, מכיוון ש-

$$5 + 2 + 2 = 9$$

כמו כן ניתן ליצור את $s = 9$ גם בדרך הבאה

$$5 - 2 + 3 + 3 = 9$$

i. ממשו את הפונקציה can_create_twice(s, L) אשר מחזירה True אם אפשר ליצור את s מ-L, ו-False אחרת, תחת הגבלה זו.
הנחיה: על הפונקציה להיות מסיבוכיות זמן $O(5^n)$ כאשר n הוא אורך הרשימה L. שימו לב שקוד שרץ בזמן $n \cdot 5^n$ אינו עונה על דרישה זו.

ii. **ציירו** את עץ הרקורסיה **והסבירו** מדוע הפונקציה שמימשתם עומדת בדרישת הסיבוכיות.

ג. מומלץ לקרוא על הפונקציה המובנית eval שיכולה לסייע לכם בסעיף זה.
בהינתן רשימה L של מספרים שלמים חיוביים וביניהם המחרוזות '+', '*', ו'-', נחשוב על הרשימה כעל ביטוי מתמטי של חיבור, חיסור וכפל בין מספרים. לדוגמה, הרשימה $L = [6\ ,\ '-',\ 4\ ,\ '*'\ ,\ 2\ ,\ '+'\ ,\ 3]$ תייצג את הביטוי:

$$6 - 4 \times 2 + 3$$

בהינתן רשימה L כזו, ומספר שלם s, נרצה למצוא האם יש דרך למקם סוגריים על הביטוי המתמטי שמייצג L כך שערך הביטוי בהתאם לחוקי הקדימות של הסוגריים יהיה שווה ל-s. לדוגמה, עבור ה-L הנ"ל ו-$s = 10$:

$$(6 - 4) \times (2 + 3) = 10$$

כמו כן ניתן להגיע ל $s = 1$ על ידי:

$$\big(6 - (4 \times 2)\big) + 3 = 1$$

i. ממשו את הפונקציה valid_braces_placement(s, L) המקבלת מספר שלם s ורשימה L כמתואר[1], ומחזירה True אם קיימת השמת סוגריים מתאימה, ואחרת מחזירה False.

לשם פשטות הניתוח, נסמן ב-n את כמות המספרים ברשימה L (כלומר, מספר איברי הרשימה לא כולל המחרוזות).
הנחיה: על הפונקציה להיות מסיבוכיות זמן $O(n!)$

ii. **ציירו** את עץ הרקורסיה **והסבירו** מדוע הפונקציה שמימשתם עומדת בדרישת הסיבוכיות.

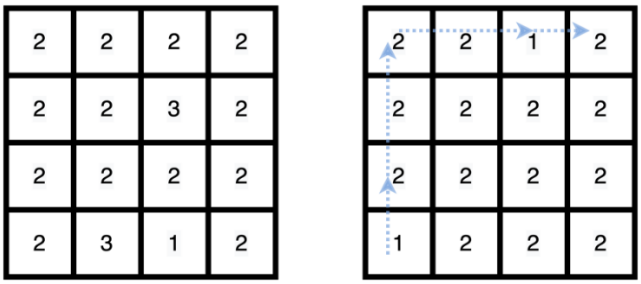
[1] באופן יותר מדויק, מתקיים שהאינדקסים הזוגיים ברשימה (מתחילים מ-0) תמיד יהיו מספרים שלמים חיוביים, האינדקסים האי זוגיים יהיו אחת המחרוזות '+', '*', או '-', והרשימה תהיה באורך אי זוגי גדול או שווה ל-3.

שאלה 4

בשאלה זאת נתון לוח דו-מימדי B בגודל n על n אשר מכיל מספרים שלמים אי-שליליים (כולל 0). בדומה למטריצה, הלוח מיוצג ע"י רשימות מקוננות. מטרתנו היא להכריע האם ניתן להגיע מהמשבצת $(0,0)$ למשבצת $(n-1,n-1)$ לפי חוקי המשחק אשר יתוארו בכל סעיף של השאלה.

א. בסעיף זה חוק ההתקדמות בלוח הוא:
אם אנחנו במשבצת ה (i,j) אז אנחנו יכולים להתקדם "קדימה" או ל $(i+B[i][j],j)$ או ל $(i,\ j+B[i][j])$. כלומר, אנחנו יכולים לזוז בדיוק $B[i][j]$ משבצות בכיוון החיובי של אחד מצירי הלוח.

ממשו את הפונקציה grid_escape1(B) אשר מקבלת לוח B ומחזירה True אם ניתן להגיע מהמשבצת $(0,0)$ למשבצת $(n-1,n-1)$ ו False אחרת. לדוגמא שני לוחות משחק והפלט עבורם:



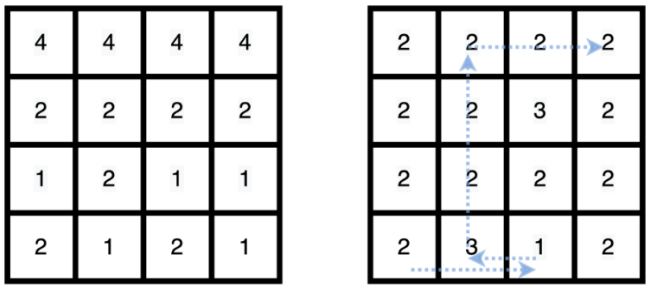
בציור השמאלי ניתן להשתכנע שאכן אין מסלול חוקי מ-$(0,0)$ ל-$(3,3)$: אנחנו מתחילים ב-$(0,0)$, ועלינו ללכת $B[0][0]=2$ צעדים. נניח שבחרנו ללכת שני צעדים למעלה, ל-$(0,2)$. כעת שוב עלינו ללכת $B[0][2]=2$ צעדים – לא ניתן ללכת למעלה, לכן נלך ימינה ל-$(2,2)$. כעת עלינו ללכת $B[2][2]=3$ צעדים – אבל אין אף כיוון חוקי ללכת אליו. באופן דומה ניתן להיווכח שכל מסלול אחר מסתיים ללא מוצא, ולכן על הפונקציה להחזיר False.

הנחיה: על הפתרון להיות רקורסיבי

ב. בסעיף זה מותר גם לזוז "אחורה" כך שחוקי ההתקדמות בלוח הם:

- אם אנחנו במשבצת ה i,j אז אנחנו יכולים להתקדם או ל $(i+B[i][j],j)$ או ל $(i,\ j+B[i][j])$ או ל $(i-B[i][j],j)$ או ל $(i,\ j-B[i][j])$. כלומר, אנחנו יכולים לזוז בדיוק $B[i][j]$ משבצות בכיוון החיובי או השלילי של אחד מצירי הלוח.
- אסור לצאת מגבולות הלוח כלומר צריך להישאר בתחום 0 עד $n-1$ בכל ציר.

ממשו את הפונקציה grid_escape2(B) אשר מקבלת לוח B ומחזירה True אם ניתן להגיע מהמשבצת $(0,0)$ למשבצת $(n-1,n-1)$ ו False אחרת. לדוגמא שני לוחות משחק והפלט עבורם:



הנחיה: על הפתרון להיות רקורסיבי

רמז: בסעיף זה יש סכנה להיכנס לרקורסיה אינסופית (כמו לולאה אינסופית). מומלץ להשתמש בזכרון עזר (בדומה לממואיזציה) בגודל הלוח כדי לסמן באילו תאים כבר ביקרנו במהלך הרקורסיה ולהשתמש במידע זה כדי להימנע מרקורסיה אינסופית.

שאלה 5

בחלק זה נדון בסיבוכיות של פעולות חיבור, כפל והעלאה בחזקה של מספרים בכתיב בינארי.

חיבור וכפל של מספרים בינאריים יכול להתבצע בצורה דומה מאוד לזו של מספרים עשרוניים. להלן המחשה של אלגוריתם החיבור והכפל של שני מספרים בינאריים A, B:

```
  1 1 1 1     (carried digits)
      1 1 1 1 (A)
+   1 1 0 1 0 (B)
-------------
= 1 0 1 0 0 1
```

להלן המחשה של אלגוריתם ההכפלה $A \times B$:

```
            1 0 1    (A)
        × 1 0 1 0    (B)
        ---------
            0 0 0    ← Corresponds to a zero in B
  +       1 0 1      ← Corresponds to a one in B
  +     0 0 0
  +   1 0 1
  ---------------
  =   1 1 0 0 1 0
```

בדוגמה הנ"ל יש 19 פעולות. 12 עבור הכפל והשאר עבור החיבור.

בהרצאה נלמד אלגוריתם iterated squaring להעלאה בחזקה של מספרים שלמים חיוביים (ללא מודולו). הקוד מופיע בהמשך.

הניחו כי מספר הביטים בייצוג הבינארי של a הינו n ומספר הביטים בייצוג הבינארי של b הינו m.

עליכם לנתח את סיבוכיות זמן הריצה של הפעולה a^b בשיטה זו כתלות ב-m ו/או n ולתת חסם הדוק ככל האפשר במונחי $O(\cdot)$.

שימו לב שהמספרים המוכפלים גדלים עם התקדמות האלגוריתם, ויש לקחת זאת בחשבון. כמו כן, בניתוח נתייחס רק לפעולות הכפל המופיעות באלגוריתם. אמנם באלגוריתם ישנן פעולות נוספות מלבד כפל. אך לפעולות אלו סיבוכיות זמן מסדר גודל זניח לעומת פעולות הכפל. למשל, ההשוואה $b > 0$ רצה בזמן $O(1)$ (לא נכנסנו לעומק של ייצוג מספרים שלמים שליליים, אבל ראינו ברפרוף את שיטת המשלים ל- 2 ושם ברור שמספיק לקרוא את הביט השמאלי כדי להכריע האם מספר שלם הוא חיובי או שלילי).

הפעולה $b \% 2$ דורשת בדיקת הביט הימני של b בלבד ולכן רצה בזמן $O(1)$. פעולת החילוק $b//2$ כוללת העתקה של b ללא הביט הימני ביותר למקום חדש בזיכרון, פעולה שגם כן רצה בזמן ליניארי בגודל של b. על ההסבר להיות תמציתי ומדויק. מומלץ לחשוב על המקרה ה"גרוע" מבחינת מספר פעולות הכפל ולתאר כיצד הגעתם לחסם עבור פעולות אלו.

```
def power(a,b):
    """ computes a**b using iterated squaring
        assumes b is nonnegative integer """
    result = 1
    while b>0: # b is nonzero
        if b%2 == 1: # b is odd
            result = result*a
        a = a*a
        b = b//2
    return result
```

סוף