<u>תרגיל בית מספר 4 - להגשה עד 16 בנובמבר (יום ד׳) בשעה 23:55</u>

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

<u>הגשה</u>:

- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- 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.
- בכל השאלות ניתן להניח כי הקלט לתכנית / לפונקציות הינו תקין, אלא אם צוין במפורש אחרת.
- תשובות מילוליות והסברים צריכים להיות תמציתיים, קולעים וברורים.
להנחיה זו מטרה כפולה:
  1. על מנת שנוכל לבדוק את התרגילים שלכם בזמן סביר.
  2. 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.

שאלה 1

בהרצאה ראינו את אלגוריתם quicksort אשר משתמש הן ברקורסיה והן באקראיות. האקראיות, כזכור, הייתה בבחירת איבר הציר (pivot) שלפיו נחלק את הרשימה לשלוש רשימות (איברים שקטנים, זהים בגודלם וגדולים מהציר שבחרנו).

דיברנו גם על האפשרות לממש את האלגוריתם באופן דטרמיניסטי, כלומר, ללא שימוש באקראיות. ההצעה שלנו למימוש דטרמיניסטי הייתה פשוטה ביותר, איבר הציר יהיה האיבר הראשון ברשימה. להלן המימוש של פונקציה זו:

```
def det_quicksort(lst):
    """ sort using deterministic pivot selection """
    if len(lst) <= 1:
        return lst
    else:
        pivot = lst[0]       # select first element from list
        smaller = [elem for elem in lst if elem < pivot]
        equal = [elem for elem in lst if elem == pivot]
        greater = [elem for elem in lst if elem > pivot]
        return det_quicksort(smaller) + equal + det_quicksort(greater) #two recursive ca
lls
```

כזכור, זמן הריצה הטוב ביותר של quicksort (עם אקראיות) הוא $O(n\log n)$ כאשר n הוא אורך הרשימה, ואילו זמן הריצה הגרוע ביותר הוא $O(n^2)$. בסעיפים הקרובים נסתכל על הגרסא הדטרמיניסטית של האלגוריתם ונחשב את הסבירות שזמן הריצה יהיה הגרוע ביותר, כתלות באופן בו מסודרים איברי הרשימה.

סעיף א'

בהנתן n (טבעי, חיובי), נסמן ב-$p(n)$ את מספר הדרכים בהן ניתן לסדר את איברי הרשימה $lst = [1,2,\dots,n]$.
לדוגמא, עבור $n = 3$ ישנן שש דרכים לסדר את הרשימה:

$$[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]$$

ולכן $p(3) = 6$. מהו $p(n)$ ל-n כללי? הוכיחו את תשובתכם.

סעיף ב'

בהנתן n (טבעי, חיובי), נסמן ב-$w(n)$ את מספר הדרכים בהן ניתן לסדר את איברי הרשימה $lst = [1,2,\dots,n]$ כך שריצת det_quicksort על הרשימה תרוץ בזמן הארוך ביותר.
לדוגמא, עבור $n = 3$ ניתן לוודא כי הסידורים הבאים:

$$[1,2,3],[1,3,2],[3,1,2],[3,2,1]$$

יניבו את זמן הריצה הגרוע ביותר, ולכן $w(3) = 4$. מהו $w(n)$ ל-n כללי? הוכיחו את תשובתכם.

סעיף ג'

בהנתן שני הסעיפים הקודמים, חשבו את הגבול וציינו מהו:

$$\lim_{n\to\infty}\frac{w(n)}{p(n)}$$

מה ניתן להסיק מהגבול על הסיכוי לכך שזמן הריצה של האלגוריתם יהיה גרוע ביותר ככל ש-n גדל?

שאלה 2

סעיף א'
נתון הקוד הרקורסיבי הבא לחיפוש בינארי.

```
def rec_slice_binary_search(lst, key):
    n = len(lst)
    if n<=0:
        return None

    if key == lst[n//2]:
        return n//2

    elif key < lst[n//2]: # item cannot be in top half
        return rec_slice_binary_search(lst[0:n//2], key)

    else : # item cannot be in bottom half
        return rec_slice_binary_search(lst[n//2+1:n], key)
```

1- ציירו את עץ הרקורסיה, עבור הקלט הבא:
a. $lst = [1,2,3,4,5,6,7,8]$, $key = 8$

2- נתחו את סיבוכיות הזמן עבור קלט $lst = [1,2, \ldots\ldots, n], key = 0,$

סעיף ב'
הקוד שלעיל לא מחזיר פלט נכון במקרים רבים. הסבירו במילים באילו מקרים לא יוחזר פלט נכון, וכן תנו דוגמא מייצגת לריצת האלגוריתם והפלט השגוי שלו.

סעיף ג'
בקובץ השלד ממשו גרסה תקינה של הפונקציה הרקורסיבית.
הנחיות:
- על הפונקציה להיות <u>רקורסיבית</u>
- יש להשתמש ב slicing.
- אין לשנות את חתימת הפונקציה, על הפונקציה לקבל רק שני פרמטרים lst, key
- נסו לבצע שינוי מינימלי בקוד. רמז: אפשר לתקן את הבעיה ע״י הוספת/שינוי מספר שורות קטן.

<u>שאלה 3</u>

בשאלה זו נרצה לעבוד עם מטריצות. מאחר שאין לנו בפייתון משתנה מטיפוס מטריצה, נשתמש ברשימות מקוננות על מנת לייצג מטריצות. הייצוג שלנו יהיה טבעי ביותר: בהנתן 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** מעניינת במיוחד – מטריצת Hadamard. למספר טבעי n מטריצת Hadamard מסדר n (אותה נסמן מעתה ואילך על ידי had(n)) מוגדרת באופן הרקורסיבי הבא:

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

  אם had(n-1) היא מטריצת Hadamard מסדר n-1, אזי:

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

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, had(n) מורכבת מ-4 עותקים של had(n-1) (זוהי מטריצת בלוקים עם שני בלוקים עליונים של had(n-1) ושני בלוקים תחתונים של had(n-1)), כאשר בבלוק הימני-תחתון ערכי המטריצה מתהפכים (כלומר, ערכי 0 הופכים ל-1 וערכי 1 ל-0).

למטריצת Hadamard התכונה המרתקת הבאה: כל שתי שורות שונות במטריצה נבדלות בדיוק בחצי מהקואורדינטות שלהן (בדקו זאת!). תכונה זו שימושית במיוחד בתחום של תיקון שגיאות, אותו נפגוש בהמשך הקורס.

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

had0 = [[0]]

had1 = [[0, 0], [0, 1]]

had2 = [[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 $O(n)$ (ולא פחות מכך).</u>

<u>סעיף א'</u>

בקובץ ה-*pdf* הוכיחו באינדוקציה כי לכל n מתקיימת התכונה הבאה: במטריצה *had(n)*, כל שורה מלבד השורה העליונה מכילה מספר זהה של אפסים ואחדות.

<u>סעיף ב'</u>

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

דוגמאות הרצה:

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

<u>הנחיות</u>:
- על המימוש להיות רקורסיבי
- יש לממש את הפונקציה בצורה יעילה ככל הניתן
- בפרט, אין לייצר את כל המטריצה $had(n)$

<u>סעיף ג'</u>

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

<u>סעיף ד'</u>

בקובץ השלד השלימו את מימוש פונקצית הלמבדא *had_complete*. הפונקציה מקבלת כקלט שלם אי-שלילי $n \geq 0$ ומחזירה את מטריצת *Hadamard* מסדר n.
עליכם להשתמש בפונקציה *had_local*.

שאלה 4

גם בשאלה זו נעבוד לפי הגדרת המטריצה מהשאלה הקודמת.

נגדיר מטריצה i-j-matrix להיות מטריצה בגודל n x n שמכילה מספרים בין i-j.
"מסלול אפסים" ב 0-9-matrix הינו רצף של אפסים סמוכים, לא באלכסון, שמתחיל בפינה השמאלית העליונה במערך ומסתיים בפינה הימנית התחתונה שלו.

<u>דוגמא א</u>:

המערך הבא הינו מערך 0-9-matrix שמכיל "מסלול אפסים":

0	0	0	5	7	9
0	9	0	3	1	5
0	0	0	8	7	6
1	6	0	0	0	0
7	3	0	6	1	0
2	4	0	0	0	0

<u>דוגמא ב</u>:

והמערך הבא הינו מערך 0-9-matrix שאינו מכיל "מסלול אפסים":

1	0	0
5	2	0
9	4	0

<u>סעיף א'</u>

ממשו בקובץ השלד את הפונקציה find_zero_paths(m) אשר מקבלת מטריצה m, מחפשת את מסלולי האפסים, כך שניתן להתקדם ימינה ולמטה בלבד, ומחזירה:

1- מספר "מסלולי אפסים" שונים שיש במערך (למסלולים מותר לחפוף בחלקם).
2- מטריצה בגודל n x n אשר כל תא בה מחזיק את מספר המסלולים השונים שעוברים בתא הזה.

הערה: הפונקציה מחזירה זוג ערכים r, m כך ש r הוא מספר המסלולים, ו m המטריצה.

<u>דוגמא:</u>

עבור הקלט של דוגמא א למעלה, הפונקציה תחזיר 4, מספר המסלולים, ואת המטריצה הבאה שמחזיקה את "מסלולי אפסים" שיש בה:

4	2	2	0	0	0
2	0	2	0	0	0
2	2	4	0	0	0
0	0	4	2	2	2
0	0	2	0	0	2
0	0	2	2	2	4

<u>הנחיות:</u>

- על הפונקציה להיות <u>רקורסיבית</u>
- -כרגיל, ניתן להשתמש בפונקציות עזר שתכתבו כרצונכם.
- רמז: לא להשתמש ב slices.

<u>סעיף ב'</u>

נתחו את סיבוכיות הזמן של הפונקציה, התייחסו למקרים הבאים:

- הקלט הינו מטריצה בגודל n x n
- הקלט הינו מטריצה בגודל קבוע, 10x10.

שאלה 5

בשאלה זו נתונה קבוצת עצמים, כך שלכל אחד מהם יש משקל ומחיר, ובנוסף נתון חסם על המשקל, שמהווה את המשקל המקסימלי ש "תיק הגב" שלנו מסוגל להכיל. המטרה היא למצוא אוסף של העצמים הנתונים שסכום משקליהם אינו עולה על החסם הנתון, וסכום מחיריהם הינו מרבי. כלומר ברצוננו למקסם את המחיר הכולל של עצמים שיכולים להיכנס לתיק עם הגבלת משקל.

נתונים n עצמים ולכל עצם i נתונים מחיר a_i ומשקל w_i, וחסם k , תת-קבוצה $|B| = m$ תקרא "תת-קבוצה מקסימלית" ביחס ל k אם המחיר הכולל של העצמים $\sum_{i=0}^{m} a_i$ מקסימלי ומשקלם הכולל של החפצים אינו עולה על החסם $\sum_{i=0}^{m} w_i \leq k$

דוגמא:

$A = \{20, 5, 10, 40, 15, 25\}$

$W = \{1, 2, 3, 8, 7, 4\}$

$k = 10$

הקבוצה הבאה הינה "תת-קבוצה מקסימלית"

$B = \{20, 40\}$

לא קשה לראות שזו הקבוצה עם הסכום הגבוה ביותר שניתן להכניס לתיק שלנו:

סכום אברי B $60 = 40 + 20$ הינו מקסימלי : והמשקלים : $1 + 8 = 9 < 10 = k$

סעיף א'

ממשו בקובץ השלד את הפונקציה find_max_profit(A, W, n, k), הפונקציה מקבלת כקלט את הרשימה A ו-W ואת הערך k. כמו כן הפונקציה מקבלת משתנה נוסף n. הפונקציה מחזירה את סכום המחירים של תת הקבוצה המקסימלית ביחס לקלט. למשל בדוגמא למעלה הפונקציה תחזיר 60.
בקריאה לפונקציה המשתנה n יכיל את אחד מהערכים הבאים :
א. כאורך הרשימה A
ב. כאינדקס הגבוה ביותר ברשימה A (כלומר, אורך הרשימה A פחות 1)
באפשרותכם לבחור באילו מהדרכים לממש את הפונקציה.
שימו לב: בקובץ השלד, הטסט מתייחס למקרה ב׳, דאגו לשנות אותו אם בחרתם לממש בהתאם למקרה א׳.

הנחות והנחיות :

- על הפונקציה להיות רקורסיבית
- A ו W מכילים מספרים חיובים שלמים, ללא חזרות
- B מכילה איברים ללא חזרות

<u>סעיף ב׳</u>

הניחו לצורך ניתוח הסיבוכיות בסעיף זה כי פעולות אריתמטיות לוקחות זמן קבוע. כמו כן, נניח שאורך כל הרשימות הוא n.

בקובץ ה pdf ציינו איזו טענה מהטענות הבאות היא נכונה והדוקה ביותר (מבין האפשרויות הנתונות) לגבי סיבוכיות זמן הריצה של הקוד שכתבתם, כפונקציה של רשימות באורך n. נמקו והסבירו את תשובתכם.

1. הסיבוכיות היא קבועה
2. הסיבוכיות היא לינארית
3. הסיבוכיות היא פולינומיאלית
4. הסיבוכיות היא אקספוננציאלית, כלומר קיים קבוע $c > 1$ כך שסיבוכיות הזמן של הפונקציה היא $O(c^n)$.

<u>סעיף ג׳</u>

השלימו בקובץ השלד את הפונקציה find_max_profit_fast שהיא גירסה עם ממאוזציה לפונקציה מסעיף א׳.

הנחיות:

- אין לשנות חתימת הפונקציה
- אין להשתמש בלולאות for או while
- משתנה n זהה בהנחיות לסעיף א.

שימו לב: גם בסעיף זה, בקובץ השלד, הטסט מתייחס למקרה ב׳ של המשתנה n, דאגו לשנות אותו אם בחרתם לממש בהתאם למקרה א׳.

<u>סעיף ד׳</u>

הניחו לצורך ניתוח הסיבוכיות בסעיף זה כי פעולות אריתמטיות לוקחות זמן קבוע. כמו כן, נניח שאורך כל הרשימות הוא n.

בקובץ ה pdf ציינו מה תהיה סיבוכיות הקוד לאחר הוספת ממואיזציה, כפונקציה של n? נמקו והסבירו.

<u>שאלה 6</u>

בשאלה זו נממש אלגוריתמים **לחישוב מרחק עריכה** בין מחרוזות. מרחק עריכה בין שתי מילים מוגדר כמספר השינויים המינימלי הדרוש כדי לעבור ממחרוזת אחת לשנייה, כאשר השינויים המותרים הם החלפת אות, הוספת אות והשמטת אות.

דוגמאות:

- מרחק העריכה בין "computer" ל "commuter" הוא 1 (ניתן להחליף את p ב m).
- גם מרחק העריכה בין "sport" ל "sort" הוא 1 (ניתן להשמיט את האות p)
- מרחק העריכה בין "kitten" ל- "sitting" הוא 3, באמצעות סדרת השינויים הבאה:
  - נחליף את k ב- s (ונגיע ל "sitten")
  - נחליף את e ב- i (ונגיע ל "sittin")
  - נוסיף g בסוף המחרוזת (ונגיע ל- "sitting")

שימו לב שייתכנו רצפים שונים של פעולות שמובילות ממחרוזת אחת לשניה, ואנו מתעניינים באורך המינימלי של רצף כזה. שימו לב גם שמרחק העריכה הוא סימטרי: אין זה משנה, מבחינת אורך הרצף, אם מתחילים מהמילה הראשונה ומגיעים לשנייה או להפך.

<u>סעיף א'</u>

ממשו בקובץ השלד את הפונקציה `distance(s1,s2)` שמחשבת את מרחק העריכה בין שתי מחרוזות.

הנחיות:

- על הפונקציה להיות <u>רקורסיבית</u>
- אין להשתמש בלולאות for או while
- בסעיף זה אין להשתמש בממואיזציה.

דוגמאות הרצה:

```
>>> distance('computer','commuter')
1
>>> distance('sport','sort')
1
>>> distance('','ab')
2
>>> distance('kitten','sitting')


3
```

<u>סעיף ב'</u>

הניחו לצורך ניתוח הסיבוכיות בסעיף זה כי פעולות אריתמטיות לוקחות זמן קבוע. כמו כן, נניח שאורך כל מחרוזת הוא $\geq n$.

בקובץ ה pdf ציינו איזו טענה מהטענות הבאות היא נכונה והדוקה ביותר (מבין האפשרויות הנתונות) לגבי סיבוכיות זמן הריצה של הקוד שכתבתם, כפונקציה של n. נמקו והסבירו את תשובתכם.

5. הסיבוכיות היא קבועה
6. הסיבוכיות היא לינארית
7. הסיבוכיות היא ריבועית
8. הסיבוכיות היא אקספוננציאלית, כלומר קיים קבוע $c > 1$ כך שסיבוכיות הזמן של הפונקציה היא $O(c^n)$.

<u>סעיף ג'</u>

השלימו בקובץ השלד את הפונקציה `distance_fast(s1,s2)` שהיא גירסה עם ממאוזציה לפונקציה מסעיף א'.

הנחיות:

- הפונקציה `distance_fast` הינה פונקציית מעטפת. עליה לקרוא לפונקציה רקורסיבית בשם `distance_mem`.
- אין להשתמש בלולאות for או while

<u>סעיף ד'</u>

הניחו לצורך ניתוח הסיבוכיות בסעיף זה כי פעולות אריתמטיות לוקחות זמן קבוע. כמו כן, נניח שאורך כל מחרוזת הוא $\geq n$.

בקובץ ה pdf ציינו מה תהיה סיבוכיות הקוד לאחר הוספת ממואיזציה, כפונקציה של n? נמקו והסבירו.

<u>סוף</u>