תרגיל בית מס 3- גלי גלר 207704842

תרגיל מס 1

סעיף א

1. **הוכחה:** לכל n>1 מתקיים כי $\log n \le n$ לכל n > 1 לכן מתקיים:
$$nlogn \ \le n * n = n^2 = n(n-1) + n \ \ \le n! + n \ \le n! + n! = 2n!$$
לכן אכן nlogn = O(n!)

2. **הוכחה:**
$$\sum_{i=0}^{k} a_i n^i = \ a_k n^k + \ \sum_{i=0}^{k-1} a_i n^i = \ \frac{a_k}{2} n^k + \ \frac{a_k}{2} \ n^k + \sum_{i=0}^{k-1} a_i n^i = \ \frac{a_k}{2} n^k + \frac{a_k}{2k} \sum_{i=0}^{k-1} n^k + \sum_{i=0}^{k-1} a_i n^i = \ \frac{a_k}{2} n^k + \sum_{i=0}^{k-1} \frac{a_k}{2k} n^k + a_i n^i \ \ge \ \frac{a_k}{2} n^k$$
כי מחדו"א 1 לכל n>i מתקיים: $\lim\limits_{n\to\infty} \frac{a_k}{2k} n^k + a_i n^i = \ \lim\limits_{n\to\infty} n^k \left(\frac{a_k}{2k} + \ \frac{a_i}{n^{k-i}}\right) = \ \infty > 0$
לכן עבור $c = \frac{2}{a_k}$ מתקיים כי $n^k \le c * \sum_{i=0}^{k} a_i n^i$ ולכן אכן $n^k = O(\sum_{i=0}^{k} a_i n^i)$

3. **הוכחה:** מהנתון כי $f_1 = O(g_1)$ ו$f_2 = O(g_2)$ קיים m טבעי ו c_2, c_1 כך שלכל n>m מתקיים
$|f_1(n) \le c_1 * |g_1(n)|$ ו $|f_2(n)| \le c_2 * |g_2(n)|$, נסמן: $c = c_1 * \ c_2$
$$|f_1(n)| * |f_2(n)| = |f_1(n) * f_2(n)| \le c_1 * |g_1(n)| * |f_2(n)| \le \ c_1 * |g_1(n)| * c_2 * |g_2(n)| \ = c * |g_1(n) * \ g_2(n)|$$
לכן אכן $f_1(n) * f_2(n) = \ O(g_1(n) * \ g_2(n))$

4. **הפרכה:** נגדיר: $f(n) = g(n) = h(n) = n^2$, בבירור $f(n) \le g(n)$ וגם $h(n) \le g(n)$ לכן אכן מתקיים f(n)=O(g(n)) וגם h(n) = O(g(n)) אך n^4 $f^\circ h(n) = f(h(n)) = f(n^2) = n^4$ לכן אכן מתקיים:
$$f^\circ h(n) = O(n^4) \ne O(g(n)$$

5. **הוכחה:** מהנתון קיימים c_2, c_1 כך ש $|f(n)| \le c_1 * |g(n)|$ ו $|g(n)| \le c_2 * |h(n)|$ לכן מתקיים:
$|f(n)| \le c_1 c_2 * |h(n)|$, נסמן $c = c_1 * c_2$ ונקבל כי $f(n) \le c * |h(n)|$ לכן אכן f(n) = O(h(n))

6. **הוכחה:**
לכל n > 1 מתקיים: logn > 0 ועבור k ≥ 1 מתקיים $log^k n < 0$, כמו כן מתקיים $n^\epsilon > 0$ לכן נשתמש בהגדרת הגבול, עם הגבול $\lim\limits_{n\to\infty} \frac{logn}{n^2} = 0$ (מכלל הסנדוויץ' חדוא1: $0 \le \frac{logn}{n^2} \le \frac{n}{n^2} = \frac{1}{n}$, $n \ge \log n$)
$$\lim_{n\to\infty} \frac{log^k n}{n^\epsilon} = \lim_{n\to\infty} \left(\frac{logn}{n^{\frac{\epsilon}{k}}}\right)^k = \ \lim_{n\to\infty} \left(\frac{\log n^{\frac{\epsilon k}{\epsilon k}}}{n^{\frac{\epsilon}{k}}}\right)^k = \ \lim_{n\to\infty} \left(\frac{\log \left(n^{\frac{\epsilon}{k}}\right)^{\frac{k}{\epsilon}}}{n^{\frac{\epsilon}{k}}}\right)^k = \ \frac{k^k}{\epsilon^k} \lim_{n\to\infty} \left(\frac{\log n^{\frac{\epsilon}{k}}}{n^{\frac{\epsilon}{k}}}\right)^k =$$
$$\frac{k^k}{\epsilon^k} \lim_{n\to\infty} \left(\frac{\log n^{\frac{2\epsilon}{2k}}}{\left(n^{\frac{\epsilon}{2k}}\right)^2}\right)^k = \ \frac{k^k}{\epsilon^k} \lim_{n\to\infty} \left(\frac{2\log n^{\frac{\epsilon}{2k}}}{\left(n^{\frac{\epsilon}{2k}}\right)^2}\right)^k = \frac{k^k 2^k}{\epsilon^k} \lim_{n\to\infty} \left(\frac{\log n^{\frac{\epsilon}{2k}}}{\left(n^{\frac{\epsilon}{2k}}\right)^2}\right)^k \ = 0 < \infty \to log^k n = O(n^\epsilon)$$

7. ***a: הפרכה:*** נגדיר: $f_i(n) = n$ לכל i בין 1 ל n אז מתקיים כי $\max\{f_1, \ldots, f_n\}(n) = n$ ולכן
O(max$\{f_1, \ldots, f_n\}$(n)) = O(n) אבל $\sum_{i=1}^{n} f_i\,(n) = n * n = \ n^2 = O(n^2) \ne O(n)$

b: הפרכה: נגדיר: $f_i(n) = 1$ לכל i בין 1 ל n-1 ו$f_n(n) = 1$ אז מתקיים: $\max\{f_1, \ldots, f_n\}(n) = n$ ולכן
O(max$\{f_1, \ldots, f_n\}$(n)) = O(n) אבל $\sum_{i=1}^{n} f_i\,(n) = n - 1 + n = 2n - 1 \ \le 2n = O(n)$

<u>סעיף ב</u>

1. ראשית, ננתח את הסיבוכיות של כל פעולה בקוד:

```
def f1(L):
    n = len(L)  O(1)
    for i in range(n): O(n) – the loop runs n times
        if i in L:  O(n) – explanation below
            L.append(i)  O(1) - given

    return L
```

הפעולה i in L עוברת על איברי L ובודקת אם i ברשימה- במקרה שבו i לא נמצא יתבצעו len(L) מעברים כמו כן, len(L) משתנה כאשר i נמצא- לכן הפעולהappend תקרה לכל היותר n פעמים, לכן len(L) הוא לכל היותר 2n וסהכ פעולה היא O(n)

הלולאה רצה n פעמים ועבור כל ריצה שלה מתבצעת פעולה שהסיבוכיות שלה היא O(n) הסיבוכיות של הפונקציה היא **O(n^2)**

2. ננתח את הסיבוכיות של כל פעולה בקוד:

```
def f2(L):
    n = len(L) O(1)
    res = [] O(1)
    for i in range(500, n): O(n) – the loop runs n – 500 <= n times
        for j in range(i,3*i):  O(n)  - explanation below  (a)
            k=1 O(1)
            while k < n: O(logn) – explanation below (b)
                k*=2
                res.append(k)

    return res
```

a-הלולאה השנייה רצה 2i פעמים לכל i, כמו כן i < n לכן 2i <= 2n והסיבוכיות של הלולאה היא O(n)

b-ראינו בתרגול שהלולאה השלישית רצה בlogn

לכן סה"כ הסיבוכיות של הפונקציה היא **O($n^2 logn$)**

<u>סעיף ג</u>

הפונקציה הראשונה יוצאת רשימה חדשה המורכבת מעותק של הרשימה המקורית והאיבר החדש, בעוד שהפונקציה השניה משנה את הרשימה שהיא מקבלת כקלט.

כאשר אלחנן קורא לפונקציה הראשונה, לולאת הfor עוברת על הרשימה המקורית גם לאחר שהמשתנה l כבר לא מצביע עליה- הוא מצביע על הרשימה החדשה שהחזירה הפונקציה, ולכן הלולאה תרוץ רק עד סוף הרשימה המקורית.

לעומת זאת, כאשר אלחנן יקרא לפונקציה השנייה, היא תוסיף איברים לרשימה המקורית בכל איטרציה של הלולאה, ולכן הלולאה לעולם לא תגיע לסוף הרשימה.

שאלה 2 סעיף ט

n	23	53	101	199	401
complete_graph	72.9	221	539.5	1347	2635
cycle	267.8	1188.1	4852.8	20621.4	87783
Inv_cycle	122.3	424.3	1118.2	2583.4	5451.9

נתאר את זמן הכיסוי של כל אחד מהגרפים במושגי big O notation:

Complete_graph

O(n) – הקלטים גדלים באופן לינארי, ולפי הטבלה כך גם גדלי הפלטים- חלוקה של כל תוצאה בתוצאה שלפניה תהיה בין 2 ל3, כפי שניתן לראות בגרף

הסבר אפשרי לכך הוא שבגרף המלא ניתן להגיע לכל צומת מכל צומת, והסיכוי של כל צומת להיבחר שווה לכן ככל הנראה יידרש מספר קטן יחסית של צעדים כדי לכסות את כל הגרף

Cycle

O(n^2)- הקלט גדל פי 2 והתוצאה גדלה בערך פי 4, ומתקיים: (2n)^2 / n^2 = 4n^2/n^2 = 4 לכן אפשר להסיק שזמן הכיסוי הוא מסדר גודל n^2

הסבר אפשרי לכך הוא שלכל צומת יש שני שכנים, להם סיכויים שווים להיבחר, כלומר לכל צומת ניתן להגיע רק משני צמתים אחרים, ובגלל שניתן לצעוד קדימה ואחורה בגרף סביר להניח שיידרשו יותר צעדים כדי לכסות את גרף המעגל מאשר בגרף המלא

Inv_cycle:

O(nlogn)- הקלטים גדלים באופן לינארי, לעומת זאת המנות בין הפלטים הולכים וקטנים- מתחילים במנה של 3.7 בערך ומגיעים למנה של 2.1 אז כפי שראינו בסעיף א הסיבוכיות היא יותר גדולה מסיבוכיות לינארית, אך ניתן לראות מהצמצום של המנות שקיימת פונקציה עולה f(x) כך שהסיבוכיות הכללית היא O(n*f(n)) ואכן ניתן לראות כי:

$$\lim_{n\to\infty}\frac{2nlog2n}{nlogn}=\lim_{n\to\infty}\frac{2log2n}{logn}=\lim_{n\to\infty}\frac{2(log2+logn)}{logn}=\lim_{n\to\infty}\frac{2log2}{logn}+2=2$$

כמו כן, ניתן לראות כי זמן הכיסוי של inv_cycle נמצא בין cycle לcomeplete_graph, לכן סביר שהסיבוכיות תהיה בין הסיבוכיות של שניהם

הסבר לכך יכול להיות שהגרף מכיל את גרף המעגל, ולכן זמן הכיסוי של גרף המעגל הוא חסם עליון, אך קיימים צמתים עם 3 שכנים, מה שעשוי לקצר את זמן הכיסוי

טבלה עבור random_graph:

n	125	250	500	1000	2000
0.3	746.4	1484.4	3448.8	7384.7	15998.3
0.5	643.1	1570.0	3152.0	7851.0	16449.3
0.7	723.6	1448.5	3719.2	7330.6	15840.0

להערכתי, זמן הכיסוי הוא O(n) כי לכל p המנה בין תוצאת הרצה לזו שקדמה לה נעה בין 1.9 ל2.5 עבור קלט שגדל פי 2 בכל הרצה. מהתוצאות האמפיריות עושה רושם שp גדול יותר אינו משפר את זמן הכיסוי, למרות שהייתי מצפה לכך שהוא כן ישפר מכיוון שp גדול יותר אמור להעיד על גרף עם יותר קשתות בין הצמתים ושיערתי בסעיפים קודמים שזמן הכיסוי של גרפים כאלו קצר יותר. סיבה אפשרית היא צמתים "מקושרים" פחות מאחרים שמכריחים אותנו לחזור אחורה, שכן קיום הקשתות הוא רנדומלי ומספרן משתנה בין צמתים

<u>Return graph:</u>

ראשית, הn המקסימלי עבורו הפונקציה רצה בפחות מדקה הוא 25, לכן ניתן להשתמש בערכי n קטנים יחסית על מנת להבין את סדר הגודל.

n	2	4	8	16
Return_graph	2	12.9	160.4	39068.5

O(2^n)- ניתן לראות כי לכל קלט המנה בין הפלט לפלט הקודם היא מספר גדול יחסית (לפחות 6), כמו כן המנה גדלה- מ6 ל 243, לכן ניתן להסיק שזמן הכיסוי אקספוננציאלי, כלומר קיים מספר k כלשהו כך ש k^n חוסם את הפונקציה, אני משערת שk=2 כי לכל צומת חוץ מהצומת הראשונה יש שני שכנים עם סיכוי שווה להיבחר

לדעתי זה קורה כי לכל צומת יש שני שכנים- הצומת הבא והצומת הראשון, לשניהם סיכוי זהה להיבחר. כאשר הצומת הראשון נבחר צריך "ללכת" שוב את כל הגרף מההתחלה, לכן זמן הכיסוי ארוך משמעותית מזמן הכיסוי של שאר הפונקציות

<u>שאלה 3 סעיף ב</u>

בקובץ skeleton כתבתי את הפונקציה הבאה:

```
def generate_sorted_blocks(lst, k):
    result = [] # O(1)

    for i in range(0, len(lst), k): # O(n/k)
        sublist = lst[i: i + k].copy() # O(k) + O(k)
        selection_sort(sublist) # O(k^2)
        result.append(sublist) # O(1)

    return result
```

הלולאה החיצונית רצה n/k פעמים

סיבוכיות זמן הריצה של slice ושל copy היא k, ביחד 2k

ראינו בכיתה שסיבוכיות slelection_sort עבור קלט בגודל k היא k^2

הפעולה append היא O(1)

סה"כ קיבלנו:

נתון כי $k \geq n$, k,n שלמים וחיוביים לכן מתקיים:

$\left\lceil\frac{n}{k}\right\rceil(k^2+2k) \le \left(\left\lfloor\frac{n}{k}\right\rfloor+1\right)(k^2+2k) \le 2\left\lfloor\frac{n}{k}\right\rfloor(k^2+2k) \ \le 2\frac{n}{k}(k^2+2k) = 2nk+2n \ \le 2nk + 2nk = 4nk = O(nk)$

<u>שאלה 3 סעיף ד</u>

קודם מופעלת הפעולה generate_sorted_blocks שראינו שהיא O(nk) ואחכ מופעלת הפעולה merge_sorted_blocks שנתון שהיא O(m*t*log n) על הרשימה שנוצרה בפעולה הקודמת, לכן

סהכ מתבצעות $\left\lceil\frac{n}{k}\right\rceil * k * log\left\lceil\frac{n}{k}\right\rceil$ nk + m*t*logm = nk + פעולות. כמו כן, נתון כי $\left\lceil\frac{n}{k}\right\rceil \ge 1$ ו $\left\lceil\frac{n}{k}\right\rceil = m \ge$ t = k

$nk + \left\lceil\frac{n}{k}\right\rceil klog\left\lceil\frac{n}{k}\right\rceil \ \le nk + \left(\left\lfloor\frac{n}{k}\right\rfloor+1\right)klog\left(\left\lfloor\frac{n}{k}\right\rfloor+1\right) \ \le nk + k\ \frac{2n}{k}log\frac{2n}{k} = nk + 2nlog\frac{2n}{k} = nk + 2nlog2 + 2nlog\frac{n}{k} \le nk + 2nlog2 + 2lnogk^2 \le nk + 4n + 2lnogk^2 \le nk + 4nk + 2lnogk^2 = 5nk + 4nlogk \ \le nk + 8nk = 9nk = O(nk)$

<u>שאלה 3 סעיף ה</u>

הסיבוכיות של *merge_sort* היא *O(nlogn)*, כדי ש*sort_by_block_merge* תהיה *O(nlogn)* צריך להיות *c* כך שמתקיים *9cnk* $\le nlogn$ כלומר הערך האסימפטוטי הגדול ביותר של *k* כפונקציה של *n* הוא ***logn***

<u>שאלה 4</u>

<u>סעיף א</u>

b. הפונקציה שכתבתי בסעיף *a*:

```
def find_missing(lst, n):
    left = 0
    right = n

    while left < right:
        middle =  (right + left) // 2

        if lst[middle] > middle:
            right = middle
        else:
            left = middle + 1

    return right
```

סיבוכיות הפונקציה היא ***O(logn)*** – לפני הלולאה ובתוכה מתבצעות פעולות שהסיבוכיות שלהן היא *O(1)* כמו פעולות אריתמטיות ותנאים, והלולאה עצמה רצה *logn* פעמים, בדומה לאלגוריתם החיפוש הבינארי שהיא מבוססת עליו, כי בכל ריצה של הלולאה הפונקציה חוצה את הרשימה וממשיכה לעבוד רק עם מחצית אחת.

סעיף ב

d. הפעולה *find* קוראת ל *find_pivot* ואז ל*binary search*, אז הסיבוכיות שלה היא סכום הסיבוכיות של שתי שפונקציות האלו.

Find_pivot היא מסיבוכיות *O(logn)* כי היא חוצה את הרשימה בכל איטרציה, *binary_search* גם היא *O(logn)*- היא מקבלת רשימה באורך *n*, אינדקס תחתון בין 0 ל*n-1* ואינדקס עליון באותו התחום, לכן במקרה הגרוע בו האינדקס התחתון הוא 0 והעליון הוא *n-1* סיבוכיות זמן הריצה של הפונקציה תהיה כמו זו של *binary_search* שנלמדה בכיתה, ולכן סיבוכיות זמן הריצה של הפונקציה היא *O(logn)*, לכן הסיבוכיות של *find* כולה היא ***O(logn)***

סעיף ג

b. נסתכל על המקרה הגרוע והמקרה הטוב:

לצורך נוחות ההסבר, נניח ש*n* אי זוגי ויש *n-1* ערכים השווים ל*x* ושונים מ*s* וערך אחד השווה ל*s* כך ש:

Lst = [x]$(\frac{n-1}{2}+1)$ *+ [s] += [x]*$(\frac{n-1}{2}-1)$

אז התנאי *lst[left]==lst[middle] == lst[right]* יתקיים $(\frac{n-1}{2}-1)$ פעמים ולכן הסיבוכיות תהיה ***O(n)***, לעומת זאת, נציין שבמקרה הטוב הרשימה ממויינת או מסובבת ללא חזרות – במקרה זה הפונקציה תעבוד בדיוק כמו פונקציית החיפוש הבינארי או הפעולה מסעיף ב, שהסיבוכיות שלה היא ***O(logn)***

שאלה 5

סעיף ד

נפרט את הפעולות שלא תלויות זו בזו, כלומר יש חיבור ביניהן:

יצירת *help_list* היא 5^k פעולות

לולאת *for* על *lst* שקוראת ל*string_to_int*, שהסיבוכיות שלה היא *O(k)*, סה"כ קיבלנו *nk* פעולות

לולאה שעוברת על *help_list*, עבור לכל היותר *n* ערכים תתבצע הפעולה *int_to_sring* ועבור *n* ערכים בדיוק תתבצע השורה *sorted_list.append(string)*, לכן הסיבוכיות הכללית תהיה $O(5^k+nk)$ – ניתן להשתמש בחיבור ולא בכפל כי הפעולות שבתוך הלולאה שעוברת על *help_list* קורות עבור עד *n* ערכים ספציפיים

סעיף ו

הלולאה החיצונית תרוץ 5^k ועבור כל איטרציה תרוץ הלולאה הפנימית שעוברת על *n* איברי הרשימה, עבור כל איבר ברשימה הפונקציה קוראת ל*string_to_int*- פעולה שהיא *O(k)* ומבצעת השוואה של שני מספרים שמתבצעת ב*O(1)* ואת פעולת ה*append* לרשימה הממוינת, שגם היא *O(1)* סה"כ קילבנו שהסיבוכיות הכוללת היא ***O(***5^k****nk)***

שאלת בונוס:

מיכל משתמשת בחיפוש בינארי, אבל במקום "לזרוק" את החצי הלא רלוונטי היא מחזירה אותו לערימה ככה שכשתמצא את הקלף של אמיר הוא יהיה באינדקס שאמיר בחר.

אם נסמן את הקלפים בחבילה ההתחלתית ב0-7, מיכל מחלקת את החבילה ל-2: אינדקסים זוגיים ואי זוגיים, ואמיר מצביע האם הקלף שלו הוא בחצי הזוגי או האי זוגי.

לאחר מכן מיכל מסדרת את החצאים כך שהחצי עם הקלף של אמיר (נניח לצורך העניין שזה החצי הזוגי) למעלה ושוב מחלקת ל2 חבילות- כעת היא יודעת ששני הקלפים הראשונים בכל ערימה הם מאינדקס זוגי. אמיר בוחר ערימה בפעם השנייה, כעת למיכל יש שתי אופציות: הקלף הראשון או השני.

היא שוב מחברת את החבילות- כדי שיצא שהקלף של אמיר הוא אינדקס 6 היא צריכה שהחבילה שאין בה את הקלף תהיה עליונה. היא שוב מחלקת את החבילה ל-2 והפעם מחיפוש בינארי היא יודעת שהקלף שלה יהיה השלישי באחת החבילות, אמיר בוחר חבילה ומיכל שמה אותה מתחת לחבילה השנייה ויודעת שהקלף של אמיר נמצא באינדקס 6

אם אמיר היה בוחר באינדקס אחר מיכל היתה צריכה לשנות את הסדר של חיבור החבילות