תרגיל בית מס 5- גלי גלר 207704842

שאלה 1

סעיף ו

נרצה לנתח את סיבוכיות זמן הריצה של compose_list. כל מה שcompose_list עושה הוא לקרוא לcompose_list_with_index, לכן נרצה לנתח בעצם את סיבוכיות זמן הריצה של compose_list_with_index.
נשים לב שמלבד השורה האחרונה כל הפעולות הן(O(1- השוואת אינדקסים, גישה לאיבר ברשימה והחזרתו, לכן סיבוכיות זמן הריצה היא מספר הקריאות הרקורסיביות בcompose_list_index כפול הסיבוכיות של הפעולה compose של המחלקה Permutation המתבצעת בכל קריאה רקורסיבית.

Compose_list קוראת לcompose_list_with_index עם index = 0 ותנאי העצירה הוא index == k-1 לכן יהיו k קריאות רקורסיביות.

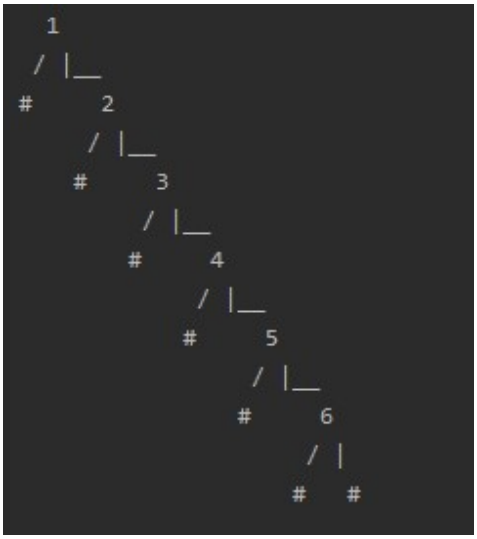
ננתח כעת את סיבוכיות זמן הריצה של הפעולה compose, המבצעת את הפעולות הבאות:
יצירה של רשימה של מספרים באורך n – O(n)
גישה לרשימות באינדקס- O(1)
יצירה של פרמוטציה- ע"פ ההנחיות בסעיף א זה יתבצע בO(n) במקרה הגרוע, לכן סה"כ הפעולה compose של Pemutation היא מסיבוכיות O(n)

כאמור, יש k קריאות רקורסיביות וכל אחת מהן מבצעת O(n) פעולות, לכן הסיבוכיות הכוללת היא **O(nk)**

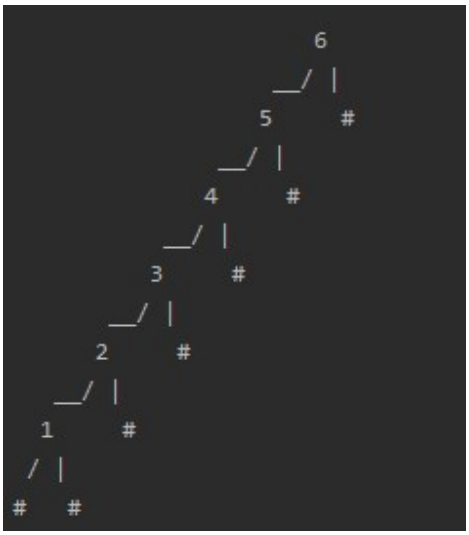
שאלה 3

סעיף א

Lst1 = [1,2,3,4,5,6] תיצור את העץ:



והרשימה lst2 = [6,5,4,3,2,1] תיצור את העץ:



אלו עצים שונים כי לשורש של העץ הראשון יש רק בנים ימניים ולשורש של העץ השני יש רק בנים שמאליים.

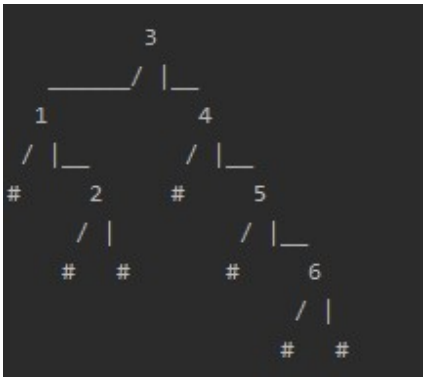
סעיף ב

נגדיר את הרשימות הבאות:

Lst1 = [3,1,4,2,5,6]

Lst2=[3,4,1,2,5,6]

משתי הרשימות מתקבל העץ הבא:



סעיף ד

סיבוכיות זמן הריצה במקרה הגרוע ביותר היא $O(n^2)$

המקרה הגרוע הוא המקרה בו הרשימות מייצגות את אותו העץ והעץ הוא עץ בו לכל צומת יש רק בן שמאלי/ ימני (עץ שנראה כמו העצים שציינתי בסעיף א)

ראשית, נרצה למצוא את סיבוכיות זמן הריצה של ריצה בודדת של הפונקציה. נשים לב שכל הפעולות שמתבצעות לפני הלולאה הן O(1), הלולאה רצה מספר פעמים השווה למספר האיברים בשתי הרשימות ובתוכה מתבצעות פעולות שסיבוכיות זמן הריצה שלהן היא O(1), לכן הסיבוכיות הכוללת היא O(len(lst1))

כעת נבדוק כמה קריאות רקורסיביות יתבצעו. בכל ריצה הפונקציה קוראת עצמה פעמיים: פעם אחת עם רשימה ריקה ופעם נוספת עם רשימה שמתקצרת בכל קריאה באיבר אחד, לכן סהכ יתבצעו 2n קריאות. נחשב את הסיבוכיות הכוללת:

$$\boldsymbol{n} + \sum_{\boldsymbol{i=1}}^{\boldsymbol{n-1}} \boldsymbol{i} \ = \boldsymbol{O}(n^2)$$

סעיף ה

לא ניתן לשפר את סיבוכיות זמן הריצה עם ממואיזציה מכיוון שבמקרה הגרוע שציינתי אין קריאות שחוזרות על עצמן, לכן אף פעם לא תתבצע גישה לזיכרון העזר והסיבוכיות תהיה זהה לסיבוכיות של הפונקציה המקורית.

שאלה 5

סעיף א

נרצה להוכיח כי $(a\ mod\ b)^c = a^c mod\ b$:

קיימים q,r כך ש $0 \le r < b$ ו $a = qb + r$, לכן מנוסחת הבינום נקבל:

$$\boldsymbol{a^c = (bq + r)^c = \sum_{i=0}^{c} \frac{c!}{i!\,(c-i)!} (bq)^i\, r^i}$$

נשים לב שלכל $i > 0$ הביטוי שבתוך הסיגמא מתחלק בb ללא שארית, לכן מתקיים:

$$\boldsymbol{a^c mod\ b = (bq + r)^c mod\ b = \left(\sum_{i=0}^{c} \frac{c!}{i!\,(c-i)!} (bq)^i\, r^i\right) mod\ b = \frac{c!}{c!} r^c mod\ b = r^c\ mod\ b}$$

כעת, נסתכל על הצד השמאלי של המשוואה:

$$(a\ mod\ b)^c = ([bq + r] mod\ c)^c = r^c$$

ולכן מתקיים:

$$(a\ mod\ b)^c mod\ b = r^c\ mod\ b$$

ואכן $(a\ mod\ b)^c = a^c mod\ b$

סעיף ב

ראשית, מכך שנתונים לנו a' , $g^b mod\ p$, p ניתן לחשב בקלות את הביטוי $(g^b mod\ p^{a'}) modp$, נראה שמתקיים $\left((g^b mod\ p)^{a'}\right) modp = g^{ab} mod\ p$

$$g^{ab} mod\ p = (g^a)^b mod\ p = (g^a mod\ p)^b mod\ p = (g^{a'} mod\ p)^b mod\ p = (g^{a'})^b mod\ p = g^{a'b} mod\ p = (g^b)^{a'} mod\ p = (g^b mod\ p^{a'}) modp$$

נימוקים לפי סדר המעברים: חוקי חזקות, סעיף א, נתון, סעיף א, חוקי חזקות, חוקי חזקות, סעיף א

קיבלנו שניתן לחשב את $g^{ab} mod\ p$ בעזרת העלאה בחזקה שידועה לנו וחישוב של מודולו, ולכן ניתן לחשב אותו ביעילות

שאלה 6

סעיף ב

נרצה לחשב את סיבוכיות זמן הריצה של הפונקציה *prefix_suffix_overlap*. המקרה הגרוע הוא המקרה בו קיימת חפיפה בין הרישא של כל מחרוזת לסיפא של כל השאר.

זה המקרה הגרוע מכיוון שבמקרה הזה תתבצע במלואה ההשואה של הרישא של כל רשימה לסיפא של כל השאר. כעת נמצא את סיבוכיות זמן הריצה:

הלולאה החיצונית רצה *n* פעמים. בכל איטרציה רצה לולאה נוספת *n* פעמים, כאשר עבור *n-1* מהן מתבצעת ריצה של לולאה שמשווה את הרישא לסיפא ומתבצעת *k*פעמים לכן הסיבוכיות תהיה:

$n(n-1)k = O(n^2k)$

סעיף ה

נרצה לחשב את סיבוכיות זמן הריצה של *prefix_suffix_overlap_hash1*

נחלק את הפונקציה לשני חלקים: יצירת המילון שמכיל את הרישות ויצירת הרשימה שמכילה את זוגות המחרוזות החופפות. סיבוכיות זמן הריצה של הפונקציה תהיה חיבור סיבוכיות זמן הריצה של שני החלקים.

נתחיל מחישוב סיבוכיות זמן הריצה של יצירת המילון. הפעולות שמתבצעות בחלק זה הן אתחול המילון שקורא לפעולה *init* של המחלקה *dict* ויוצר רשימה בגודל *n*, לאחר מכן מתבצעת לולאה שרצה *n* פעמים ובכל איטרציה היא יוצרת את רישא בגודל *k* בעזרת *slice* ומכניסה את הרישא למילון ע"י קריאה לפעולה *insert* שמשתמשת בפונקציית ה*hash* שנתון שסיבוכיות זמן הריצה שלה היא *O(k)* לכן סיבוכיות זמן היצה הכוללת של חלק זה היא *O(nk)*

לפני שננתח את סיבוכיות זמן הריצה של החלק השני נרצה לדעת מה סיבוכיות זמן הריצה הממוצעת של המתודה *find* של המחלקה *dict* וגם מה האורך הממוצע של הרשימה ש*find* תחזיר. בגלל שאורך טבלת הערבול שווה לאורך רשימת המחרוזות לא יהיו שתי רישות שונות שימופו לאותו התא, לכן אם יש רשימה בטבלת הערבול שמכילה יותר מערך אחד בהכרח כל הערכים ברשימה יוחזרו ע"י *find*. לפי טענה שראינו בהרצאה, בממוצע *n/n = 1* ערכים ימופו לאותו התא ברשימה, לכן לולאה שעוברת על הערכים המוחזרים מ*find* תתבצע ב*O(1)*. מאותה סיבה גם סיבוכיות זמן הריצה של *find* היא *O(1)*

כעת נחשב את סיבוכיות זמן הריצה של החלק השני: הלולאה הראשונה תרוץ *n* פעמים ובכל איטרציה תיצור *slice* בגודל *k*, לאחר מכן תיקרא *find* שחישבנו את סיבוכיות זמן הריצה הממוצעת שלה קודם והלולאה האחרונה תבצע מעבר על תוצאות *find* ומכניסה אותן לרשימה הסופית שפונקציה מחזירה לכן סיבוכיות זמן הריצה של חלק זה היא *O(nk)*

סה"כ סיבוכיות זמן הריצה הממוצעת של הפונקציה היא *O(nk)*

סעיף ז

הרצתי את שלושת הפתרונות 10 פעמים על רשימה של 20 מחרוזות בגודל 2000 עם *k = 2000*. ניתן לראות כי בכל ההרצות הפתרון הראשון, הנאיבי, הוא האיטי ביותר וששני הפתרונות האחרים רצים בזמנים מאוד דומים, עם יתרון קל לטובת הפתרון האחרון שמשתמש ברשימה של פייתון.

ניתן להסביר זאת ע"י כך שבפתרון הנאיבי יש לולאה מקוננת בגודל *n* ובנוסף השוואה של הרישות והסיפות, שמתבצעות בלולאות בלתי תלויות בשני הפתרונות האחרים, וגם ע"י כך שהפתרונות הלא נאיביים לא מבצעים השוואה של רישא של כל מחרוזת לסיפות של כל שאר *n-1* המחרוזות.

את ההבדל בין המימוש עם המילון שכתבנו בעצמנו לבין המימוש עם המילון של פייתון, שיש לציין שהוא די זניח, ניתן להסביר ע"י העובדה שהמימוש שלנו יוצר מראש רשימה בגודל *n* בעוד שפייתון יוצר רשימה באופן דינאמי, כמו כן המילון של פייתון משתמש ב*open addressing* שראינו בכיתה שהוא יעיל יותר ובאופן כללי ראינו בעבר שבמקרים רבים פונקציות של פייתון ממומשות עם אופטימיזציות שמשפרות את זמן הריצה לעומת קוד שנכתוב בעצמנו