תרגיל 5:

שאלה 1:

א. הוכחות חשבון מודולרי:

- חיבור מודולרי:

$Let\ a, b, c \in \mathbb{N}, s.t.\ \ a = n \cdot c + r\ , b = m \cdot c + k\ , for\ n, m, r, k \in \mathbb{N}, and\ 0 \leq r, t < c$

$On\ the\ one\ hand, (a + b)mod\ c = \big((n + m) \cdot c + (r + k)\big)mod\ c = (r + k)mod\ c.$

$On\ the\ other\ hand, (a)mod\ c = r, (b)mod\ c = k.$

$Hence, ((a)mod\ c + (b)mod\ c)mod\ c = (r + k)mod\ c$

$Q.E.D.$

- כפל מודולרי:

$Let\ a, b, c \in \mathbb{N}, s.t.\ \ a = n \cdot c + r\ , b = m \cdot c + k\ , for\ n, m, r, k \in \mathbb{N}, and\ 0 \leq r, t < c$

$On\ the\ one\ hand, (ab)mod\ c = \big((nmc + nk + mr) \cdot c + rk\big)mod\ c = (rk)mod\ c.$

$On\ the\ other\ hand, a\ mod\ c = r, b\ mod\ c = k.$

$Hence, (a\ mod\ c \cdot b\ mod\ c)mod\ c = (rk)mod\ c$

$Q.E.D.$

- חזקה מודולרית:

$Let\ a, b, c \in \mathbb{N}, s.t.\ \ a = n \cdot c + r\ , for\ n, r \in \mathbb{N}, and\ 0 \leq r < c$

$For\ b = 1, a^b\ mod\ c = (n \cdot c + r)mod\ c = \big((n \cdot c + r)mod\ c\big)mod\ c = r$

$Assume\ truth\ for\ all\ b \leq n \in \mathbb{N}, and\ prove\ for\ n + 1:$

$(a^n \cdot a)mod\ c = (a^n\ mod\ c \cdot a\ mod\ c) = (a\ mod\ c)^n \cdot a\ mod\ c = (a\ mod\ c)^{n+1}$

$Q.E.D.$

ב. הוכחת מציאת חזקה ללוג דיסקרטי:

$Let\ p, g \in \mathbb{N}\ be\ prime\ numbers, and\ our\ public\ keys\ for\ the\ D.H. key\ exchange.$
$Let\ a, b \in \mathbb{N}\ be\ Alice's\ and\ Bob's\ secrets\ s.t:$

$x = g^a\ mod\ p$
$y = g^b\ mod\ p$

$Assuming\ we\ have\ c \in \mathbb{N}\ s.t.\ x = g^c\ mod\ p,$
$show\ we\ can\ efficiently\ compute\ g^{ab}\ mod\ p.$

$Solution:$

$Alice\ would\ attain\ her\ mutual\ key\ by\ computing\ (g^a\ mod\ p)^b\ mod\ p.$
$Using\ former\ proofs\ and\ our\ known\ variables\ so\ far, we\ can\ say:$
$(g^a\ mod\ p)^b\ mod\ p = (g^c\ mod\ p)^b\ mod\ p = \left(g^b\ mod\ p\right)^c\ mod\ p = y^c\ mod\ p$

$Since\ we\ have\ y, c, and\ p\ we\ can\ compute\ the\ mutal\ key\ the\ same\ way\ Alice\ and\ Bob\ did.$

$Q.E.D.$

שאלה 2:

א. שאלות על פונק' האתחול של המחלקה:

a. בהינתן n ביטים במספר המתקבל ממכפלת הגורמים הראשוניים אנחנו יודעים כי המספר שלנו, שנסמן ב-a, חסום כך: $2^{n-1} \le a < 2^n$.
לכן טווח הערכים של k המייצגת את אורך רשימת הגורמים הוא: $1 \le k \le n-1$, כאשר רשימה באורך 1 מתקבלת כאשר המספר הוא ראשוני בטווח הנתון, ורשימה באורך n-1 מתקבלת כאשר המספר הוא החזקה הגדולה ביותר של 2 בטווח, שהיא n-1, ולכן נקבל רשימה באורך n-1 שבה כל האיברים הם 2.

b. נסתכל על האובייקט ע"י הייצוג לפי גורמים ראשוניים:

$P = [p_1, \dots, p_k]$
$B = [b_1, \dots, b_k]$

כאשר *P* מייצגת את הגורמים הראשוניים ו-*B* מייצגת את מספר הביטים שדרושים לייצוג הבינארי של כל גורם.
נסתכל תחילה על $p_1 \cdot p_2$:

אנחנו יודעים כי סיבוכיות הפעולה היא $O(b_1 \cdot b_2)$, כעת נשאל כמה ביטים דרושים לייצוג של $p_1 \cdot p_2$?
נציג את המספרים שלנו כפולינום המתאים לייצוג הבינארי:

$p_1 = \alpha_0 \cdot 2^0 + \dots + \alpha_{b_1-1} \cdot 2^{b_1-1}$
$p_2 = \beta_0 \cdot 2^0 + \dots + \beta_{b_2-1} \cdot 2^{b_2-1}$

אז אם נסתכל על הכפל ונזכור כי $\alpha_i, \beta_i \in \{0,1\}$ נקבל:

$$p_1 \cdot p_2 = (\alpha_0 \cdot \beta_0) \cdot 2^0 + \cdots + \left(\alpha_{b_1-1} \cdot \beta_{b_2-1}\right) \cdot 2^{b_1+b_2-2}$$

ואנחנו רואים ישירות כי דרושים $b_1 + b_2 - 1$ ביטים.
באופן כללי אנחנו יכולים לראות כי בעבור *k* גורמים, כאשר אנחנו יודעים את מספר הביטים של כל גורם, ובהמשך לסימון שקבענו בתחילת הפתרון, נקבל כי המכפלה שלהם דורשת $\left(\sum_{i=1}^{k} b_i\right) - (k-1)$ ביטים, נזכור כי אנחנו דורשים כי מכפלת הגורמים ייצגו מספר בן *n* ביטים ולכן נאמר:

$$n = \left(\sum_{i=1}^{k} b_i\right) - (k-1)$$

בתהליך דומה ניתן להגיע כי סיבוכיות הכפל של כל הגורמים כאשר אנחנו כופלים גורם אחד כל פעם לתוך המכפלה כולה היא:

$$\sum_{s=1}^{k-1} O\left(\left(\left(\sum_{i=1}^{s} b_i\right) - (s-1)\right) \cdot b_{s+1}\right)$$

ומכיוון שכל אחת מכמות הביטים של הגורמים ניתן להציג בתור איזשהו חלק יחסי מתוך *n* אפשר להגיד שבגלל הכפל יש פה סכימה של גורמים מסדר n^2 ולכן לחסום את כל הביטוי ע"י $O(n^3)$.

ננסה להראות הדיקות:
אנחנו יודעים ש-*is_prime* רצה בסיבוכיות של $O(n^3)$. זה החסם ההדוק ביותר שהראנו בעבור *is_prime* בקורס ולכן נתייחס אליו בתור חסם הדוק.
במצב כזה, וע"י היעזרות במשפט המספרים הראשוניים, אנחנו יודעים שבעבור *n* ביטים קיים מספר ראשוני *p* כך ש-

$$2^{n-1} < p < 2^n$$

למספר הזה *n* ביטים, ולכן במידה ופונק' האתחול תבדוק את הרשימה בעבור ראשוניות, נקבל את סיבוכיות הפונק' *is_prime*, שכפי שציינו החסם ההדוק ביותר שיש לנו בעבורה הוא $O(n^3)$.

<u>אני מוסיף את הקטע הבא על מנת לנסות להראות חסם על פעולת כפל הגורמים הראשוניים זה בזה:</u>

לכאורה, אנחנו מבצעים פעולת כפל, שהיא $\mathrm{O}(\mathrm{k} \cdot \mathrm{m})$, כאשר k ו-m מייצגות את מספר הביטים של המספרים הנכפלים.
לבצע פעולה זו שוב ושוב כמספר הגורמים כאשר בכל פעם מספר הביטים בכפל שלנו גדל "נותן תחושה" של סיבוכיות גדולה מ-n^2.
אבל גם קיימת מגבלה על מספר הגורמים הראשוניים שמכפלה שלהם לא גדולה ממספר בן n ביטים, ולכן בהתחשב בכל האילוצים, אני תמיד מגיע לסיבוכיות של $\Theta(\mathrm{n}^2)$ בעבור פעולת מכפלת הגורמים.
אני מצרף את הניסיון המתוחכם ביותר שלי:

נראה חסם ע"י התחשבות במשפט המספרים הראשוניים.
בהינתן $m \in \mathbb{N}$ כלשהו, אנחנו יודעים שקיים $p \in \mathbb{N}$ ראשוני כך ש-

$$2^{m-1} < p \leq 2^m$$

מעולה לנו.

נתחיל לאסוף אותם: $2^0 < p_1 \leq 2^1$, $2^1 < p_2 \leq 2^2$, ...

נשים לב, שאנחנו צריכים לקיים ש-

$$n = \left(\sum_{i=1}^{k} b_i\right) - (k-1)$$

אבל כעת אנו יודעים: $b_1 = 2, b_2 = 3, \dots, b_k = k+1$

ננסה להביע את k באמצעות n.

זו סדרה חשבונית, שנראית כך:

$$n = 2 + 3 + \cdots + k + (k+1) - (k-1) \Longrightarrow$$
$$\Rightarrow n - 1 = \sum_{i=1}^{k} i = \frac{k(k+1)}{2} \Longrightarrow$$
$$\Rightarrow k^2 + k + 2 - 2n = 0 \Longrightarrow k = \Theta(\sqrt{n})$$

נחשב באופן דומה את הסיבוכיות:

$$\sum_{s=1}^{\Theta(\sqrt{n})} \Theta\left(\frac{s(s+1)}{2}\right) \cdot (s+2) = \sum_{s=1}^{\Theta(\sqrt{n})} \Theta(s^3) = \Theta\left(\left(\sqrt{n}\right)^4\right) = \Theta(n^2)$$

נשים לב כי לא כל n מתלכד עם סכום סדרה חשבונית, אבל *אני חושב* שניתן להוכיח כי בהינתן n מקיים את התנאי, אז זה ה-*worst case* מבחינת סיבוכיות.
אני לא מוכיח מכיוון שבכל מקרה הסיבוכיות לא מגיעה לדרישות השאלה.

ב. פונק' *lcm*:

```
my_set = set()
lists = [self.factors] + [x.factors for x in others]
f_i_d = [{f: 0 for f in x} for x in lists]
```

בהתחלה אנחנו מאתחלים:

- סט, ישמש אותנו אחר כך בשביל לבנות את המילון שיעזור לנו למצוא את הגורמים של ה-*lcm*.
- רשימה של רשימות הגורמים הראשוניים של כל מספר המעורב בחישוב.
- רשימה של מילונים, כאשר כל מילון מתאים לרשימה אחת של פירוק לגורמים.
  אנחנו משתמשים בגורמים הראשוניים בתור מפתחות, כאשר הערך של כל מפתח הוא מספר ההופעות של אותו גורם ראשוני ברשימה המתאימה לאותו המילון.

```
for i in range(len(lists)):
    for f in lists[i]:
        my_set.add(f)
        f_i_d[i][f] += 1
```

אנחנו עוברים על כל רשימה באיטרציה נפרדת, ובכל מעבר על כל האיברים של כל הרשימה אנחנו מעדכנים שני דברים:

- אנחנו רושמים את הגורם הראשוני שאנחנו רואים לתוך הסט שלנו. חשוב לזכור שסט מונע כפילויות, כך שבסוף התהליך כל גורם ראשוני שהופיע באיזשהי רשימה - יופיע פעם אחת בסט.

- אנחנו מעדכנים את מספר ההופעות של אותו גורם במילון המתאים לרשימה שאנחנו עוברים עליה. מוסיפים פלוס 1 במקום הרלוונטי.

```python
f_m_d = {x: 0 for x in my_set}
```

אנחנו מאתחלים מילון שמטרתו תהיה לספור את מספר ההופעות של הגורמים הראשוניים בפירוק של ה-*lcm*. הוא יהיה חייב להתחלק בכל המספרים שקיבלנו ולכן הוא יהיה מכפלה של מספר ההופעות המקסימלי של כל גורם ראשוני שראינו בכל שאר המספרים.

כאן הסט משמש אותנו בשביל לאתחל את המילון – מובטח לנו שנצטרך את כל הגורמים הראשוניים, ונוח לנו לקבל אותם בפורמט שלא מאפשר כפילויות.

```python
for f_d in f_i_d:
    for f in f_d.keys():
        if f_d[f] > f_m_d[f]:
            f_m_d[f] = f_d[f]
```

אנחנו עוברים על כל המילונים שלנו ומכניסים למילון של ה-*lcm* את הערך המקסימלי בעבור כל גורם ראשוני ביחס לכל הערכים שמופיעים בעבור אותו גורם ראשוני בכל המילונים של כל המספרים שקיבלנו.

```python
result = []

for k, v in f_m_d.items():
    result += [k] * v

return FactoredInteger(result, verify=False)
```

כעת רק נותר להמיר את הפורמט ממילון שסופר את מספר ההופעות של כל גורם ראשוני לרשימה שמכילה את כל הגורמים הראשוניים ברשימה כפי שנדרש בשביל האובייקט שלנו.

מאתחלים את האובייקט עם הרשימה המתאימה ומחזירים.

ג. סיבוכיות פונ' *lcm*:

נסמן את מספר הגורמים הראשוניים שהפונק' מקבלת כולל ריבויים בתור *m*. (משמע אורך כל הרשימות המתקבלות יחדיו הוא m).
כבר באתחול המשתנים יש לנו מעבר על כל גורם ראשוני שמופיע בכל רשימה פעם אחת לכן אנחנו כבר ב-$O(m)$.
גם במעבר הראשון לצורך ספירת הופעות הגורמים, אנחנו מבצעים $O(1)$ עבודה לכל גורם ראשוני ולכן אנחנו בסיבוכיות $O(m)$.
בסט שלנו יש לכל היותר *m* גורמים ראשוניים שונים, ולכן גם אתחול המילון של ה-*lcm* הוא $O(m)$.
ולאחר מכן כאשר אנחנו עוברים על כל מפתח בכל מילון, יש לכל היותר *m* מפתחות בכל המילונים מכיוון שהם סופרים את ההופעות של הגורמים הראשוניים ולכן חסומים ע"י *m*. בנוסף אנחנו שוב עושים $O(1)$ עבודה לכל מפתח ולכן הסיבוכיות היא $O(m)$.

לבסוף, כאשר אנחנו בונים את הרשימה מהמילון של *lcm* אנחנו מבצעים בכל איטרציה $O(v)$ עבודה, כאשר *v* היא הערך של כל גורם ראשוני ברשימה של *lcm*, אבל נשים לב שסכום כל ה-*v*ים חסום ע"י *m*, שכן במקרה הגרוע ביותר ה-*lcm* היה פשוט כפל של כל המספרים זה בזה ולכן היה פשוט רשימת כל הגורמים של כל המספרים יחדיו, וזו בדיוק רשימה באורך *m*. לכן אנחנו שוב בסיבוכיות $O(m)$. מהנחות התרגיל אנחנו יכולים להזניח את סיבוכיות האתחול של המחלקה.

אז קיבלנו בסה"כ סיבוכיות של: $5 \cdot O(m) = O(m)$.

שאלה 4:

א. קוד

ב. אנחנו משמרים את תכונות העץ הבינארי מכיוון שבסופו של דבר, מה שמשותף למפתחות לפני ואחרי העדכון של *cumsum* הוא הסדר הלקסיקוגרפי של המפתחות בעץ, זה לא באמת משנה כמה תווים, או אפילו אילו תווים יש בכל מפתח בעץ - כל עוד כל מפתח משמר את הסדר הלקסיקוגרפי ביחס למיקום שלו בעץ. זה כמובן יהיה עץ שונה, אבל עדיין עץ בינארי.

שאלה 5:

א. קוד
ב. סיבוכיות:

```python
def prefix_suffix_overlap(lst, k):
    overlap = []

    for i, si in enumerate(lst):
        for j, sj in enumerate(lst):
            if i != j and si[:k] == sj[-k:]:
                overlap.append((i,j))

    return overlap
```

בלולאה החיצונית אנחנו עוברים על כל המחרוזות ברשימה, ובכל איטרציה אנחנו עושים זאת שוב, לכן אם נסמן ב-*n* את אורך הרשימה יש כבר $O(n^2)$ עבודה. כעת נתייחס למקרה הגרוע ביותר, כאשר כל הרשימה שלנו מורכבת ממחרוזות זהות באורך *k*, באופן זה, בכל השוואה אנחנו נאלץ לעשות $O(k)$ עבודה. לכן בסה"כ נקבל $O((n^2 - n) \cdot k) = O(n^2 \cdot k)$.

ג. קוד
ד. קוד
ה. סיבוכיות ריצה ממוצעת:

נסמן כי *n* מייצגת את אורך הרשימה וכי *k* מסמנת את אורך הרישא וסיפא שאנחנו מחפשים.
בקוד שלנו אנחנו מאתחלים מילון באורך של *2n* ולכן אנחנו יודעים להגיד כי מקדם העומס שלנו הוא:

$$\alpha = \frac{n}{2n} = \frac{1}{2} = O(1)$$

כעת, אם אנחנו דורשים שאין שום התאמות ברשימה בין רישאות וסיפאות וכל מה שביניהן, אז אנחנו יכולים להגיד בנימוקים זהים לאלו שאמרנו בשיעור, עם מקדם העומס הנתון, ועם הפיזור האחיד של פונק' ה-*hash* של פייתון, כי אנחנו מצפים לרשימות ב-*Dict* באורך $O(1)$.
כעת נניח כמבוקש כי במקרה הגרוע של השוואת מחרוזות אנחנו מבצעים $O(k)$ פעולות (למשל כאשר כל התווים זהים למעט התו האחרון) וכי חישוב *hash* על *k* תווים הוא גם $O(k)$.

```python
def prefix_suffix_overlap_hash1(lst, k):
    overlap = []
    reisha = Dict(2 * len(lst))

    for i, s in enumerate(lst):
        reisha.insert(s[:k], i)

    for j, s in enumerate(lst):
        matches = reisha.find(s[-k:])
        for i in matches:
            if i != j and s[-k:] == lst[i][:k]:
                overlap.append((i,j))

    return overlap
```

אתחול המילון דורש כבר בהתחלה $O(n)$ עבודה, אמנם כבר מיד אחרי נקבל כי הכנסה של n ערכים, כאשר לכל ערך אנחנו מחשבים *hash*, לכן בהתאם להנחה נקבל כבר $O(nk)$.

כעת בלולאה האחרונה אנחנו עוברים על כל איבר ברשימה פעם אחת, בכל פעם אנחנו מחפשים במילון, מה שדורש חישוב של *hash* ולכן אנחנו כבר מקבלים שוב $O(nk)$.

כעת נכנסים הטיעונים שכבר הראינו – בכל פעם אנחנו מקבלים רשימה באורך $O(1)$, לכן הלולאה השנייה היא בממוצע $O(1)$ עבודה, ובכל איטרציה אנחנו משווים מחרוזות שבמקרה הגרוע ביותר דורשת $O(k)$ פעולות, ולכן אנחנו בכל מקרה מקבלים בממוצע שסיבוכיות הפונק' היא $O(nk)$.