Q3

סעיף ב':

סיבוכיות הזמן תהיה O(n) . הפונקציה משווה בין הערכים של שתי תתי הרשימות הממוינות. כל תת רשימה היא באורך n/2 . לכן יהיו n/2 אופרציות של השוואה, עד שאחת מתתי הרשימות כבר הוכנסה לרשימה הסופית במלואה.

את מספר האיברים בתת הרשימה שאינה הוכנסה במלואה נצטרך להכניס את האיברים בדיוק ע"פ סדרם (שכן שתי תת-הרשימות ממוינות). ולכן מדובר בשארית מספר האיברים שנותרה, כלומר קטנה או שווה ל- n/2 . ומכך שהרשימה המקורית מכילה בסה"כ n איברים וגם אורך הרשימה הסופית הממוינת הוא n נובע שהסיבוכיות כאן חסומה ע"י n .

Q2

סעיף ג':

המספר הגדול ביותר הניתן לקידוד באופן זה הוא 16**2322 . אפשר להגיע לזה מהנוסחה: (16**-11)*fraction הוא לכל היותר 16 (אפשר גם להראות זאת ע"י חישוב). המספר גדל ככל שהמעריך גדל, והגודל המקסימלי של המעריך בהתאם לנוסחה ולייצוג בבסיס hex הוא 16**2321 .

המספר החיובי הקטן ביותר שניתן לייצוג בשיטה זו הוא 16**(-2047) . זה נובע מתוך הפורמולה: (16**-11)*fraction הוא לכל הפחות 1. אם מדובר במספר חיובי אז הסימן יהיה 0 . וכאמור אם 3 הניבלים שמייצגים את חזקת המעריך שווים כולם לאפס אזי המספר יהיה 16**(-2047) .

סעיף ד':

קיימים 3 ניבלים למעריך, כאשר בכל ניבל ניתן לבחור כל אחד מבין 16 המספרים בבסיס זה. כמו כן, קיימים 12 ניבלים ב-fraction ולכל ניבל יש 16 אפשרויות. ובניבל הסימן ניתן לבחור בין 0 ל-1. מתוך עקרונות בסיסיים בקומבינטוריקה קיימים 2*(16**15) אפשרויות.

אם היינו רוצים לייצג יותר מספרים אז שינוי בניבלים של fraction לא היה משנה. כנ"ל בחזקות של המעריך. לעומת זאת, אם היינו מחליטים לוותר על ייצוג של מספרים שליליים ולאפשר לניבל שאחראי על הסימן להיות ניבל של המעריך או של fraction אז

טווח המספרים שהיינו מייצגים היה גדל. כי כאמור כעת היה מדובר בהכפלה של 16 ולא של 2. (הכמות בסופו של דבר הייתה גדלה פי 8).

Q4

סעיף א (c):

הסיבוכיות היא $3log(n)=O(log(n))$ (בבסיס 2 כמובן). האלגוריתם כמעט זהה במהותו לחיפוש בינארי: חוצים את הרשימה לשניים ובודקים במקום פעם אחת, שלוש פעמים- שניים נוספים עבור הצד הימני של האמצע ועבור הצד השמאלי של האמצע.
הסיבוכיות היא כדלקמן משום שאפשר לומר שכמות האיטרציות מבוטאת ב-log(n) בחיפוש בינארי רגיל שבה מתקיימת בדיקה אחת, ואילו במקרה הזה 3 בדיקות. זה בעצם כמו לבצע חיפוש בינארי על 3 רשימות באורך n כלומר, $log(n)+log(n)+log(n)$.

סעיף ב (b) :

הסיבוכיות היא O(n) כאשר n אורך הרשימה. פשוט עוברים איבר-איבר בלולאת for כשמספר האיטרציות הוא כאורך הרשימה. מתחילים מהאיבר במקום 0 ובודקים עבור המיקום ה-i האם האיבר במיקום i+1 קטן או גדול ממנו. (בדיקה ב"זוגות"). הפתרון מסתמך על כך שהיחס סדר על האיברים ברשימה נשמר במרחק של לכל היותר אינדקס אחד אז לא נוצרים "חורים" בסדר של הרשימה. כלומר עד לאיבר ה-i (i מתקדם בכל איטרציה) באמצעות הבדיקה "בזוגות" ולפי המבנה של הרשימה בקלט הסדר על האיברים עד ל-i לא מופר.

סעיף ג (a) :

מתקיים יחס סדר על המספרים ברשימה, כלומר ניתן להשוות בין כל שני איברים ולהכריע האם $y<x$ או $x=y$ או $x<y$. מכיוון שניתן לבצע השוואה בין מספרים נוכל תיאורטית לעבור על כל האיברים ברשימה ולהשוות ביניהם. לפחות איבר אחד יהיה קטן מכל האיברים ברשימה, ובפרט קטן מהשכנים שלו. אז המינימום הגלובלי הזה יקיים שהוא מינימום מקומי "בסביבתו".

סעיף ג (c):

האלגוריתם מסיבוכיות O(logn) . מבוסס על שיטת חיפוש בינארי: לוקחים את הקצוות וחותכים באמצע. בודקים האם האמצע מקיים את התנאי של השאלה. במידה שכן- מחזירים את האינדקס. ואם לא אז בוחרים את השכן שערכו **הנמוך ביותר**. ראשית,

נרצה "להתקרב" אל האיבר המינימלי ברשימה, שהרי אם היינו מגיעים לאיבר המינימלי הוא היה כבר מוחזר (כי הוא קטן מכל איבר שברשימה). שנית, בכל חצייה נרצה להישאר עם איבר שקטן לפחות מאחד השכנים, ובכל מקרה לעצור באיטרציה שבה מתקיימים התנאים. כך, גם אם באחת החציות פוספס האיבר המינימלי ברשימה, באיטרציה האחרונה לכל הפחות נשאר עם שני איברים בקצות הקטע שאחד מהם קטן מהאחר ונוכל להחזיר אותו.

ולגבי הסיבוכיות- עורכים 3 בדיקות בצדדים כמו בסעיף א'. הטיעונים דומים, סדר גודל O(logn) לא משתנה.

<u>Q6</u>

<u>סעיף ג:</u>

```
s = "0123456789abcdefghijklmnopqrstuvwxyz"
list_1=[ random.choice(s)  for i in range(2**22)]
str_1= "".join(list_1)

s_ascii= ""
for char in str_1:
    s_ascii= s_ascii + str(bin(ord(char))[2:])

print(len(s_ascii))
print(len(code(str_1)))

28193097
23998793
```

<u>len(ascii(s))=28193097</u>

<u>len(code(s))=23998793</u>

בדוגמה שלהלן חוללתי מחרוזת תווים רנדומית באורך 2**22 . אפשר לראות כי לשיטה המוצעת תיתכן יעילות מסוימת מבחינת תפיסת מקום בזכרון. ראשית, אורך תווי הא"ב בשיטת ascii <= אורך תווי הא"ב בשיטה המוצעת מוביל לכך שהאורך הכללי של מחרוזת בשיטה זו תהיה קצרה יותר. למשל בשביל לייצג את "a" בשיטה המוצעת נדרשים פחות מספרים. שנית, בשל הדמיון בייצוג של מספרים אינטג'רים באמצעות ספרות בינאריות בשתי השיטות, הרי שלא יהיה הבדל ניכר, כזה שיורגש, בייחוד אם בוחרים רנדומית מספר רב ביותר של איברים למחרוזת.

בסיכומו של עניין ייתכן שבשל ההבדל בייצוג של האותיות, שכביכול חסכוני יותר בשיטה המוצגת, הערך $len(code(s))$ / $len(ASCII(s))$ ישאף לאפס (**<u>אבל מאוד לאט</u>**, בשל הבדלים קטנים באורך של הייצוג), ובכל מקרה נצפה שהערך הזה יהיה קטן מ-1.

סעיף ד:

בשיטה הקודמת היה ניתן לזהות מתי מספר מסוים נגמר ומתי מספר חדש מתחיל כי בכל פעם הייתה התייחסות ל"שרוולים" של 5 ביטים. כעת לא ידוע מהו גודל ה"שרוול" והיכן ומתי עוברים למספר הבא. הסדר לא נשמר בייחוד כאשר פתאום מופיעה מילה באמצע הטקסט, על אף שייתכן כי שכיחותה נמוכה. כי מאותו רגע כיצד נדע להמשיך לספור על סמך כמות הביטים בכל "שרוול".

למשל המספר "100" ואחריו המספר "0". האם זה "1000"? או שזה 100 בנפרד ו-0 בנפרד?

בנוסף, "0" "a" "101" - שייצוגו לפי ההצעה בסעיף תהיה: 000010000000110000000000 – כיצד ניתן לדעת שלא מדובר במספר "102" ושהתחילה ספירה לאות "a" ? וזה כמובן ישפיע על הקריאה של המספר "0" ועל קריאת המשך המחרוזת.

כלומר התפקיד של הוספת "0" לפני כל ספרה (בשיטה של הסעיף הראשון) היא על מנת שנוכל לזהות את המיקום של הסימבולים על פני המחרוזת ועל מנת שנדע כיצד להמשיך אחרי קריאה של כל סימבול.

סעיף ה:

בשיטה זו נדרשים **7** ביטים אם מתקיים ייצוג אחיד לכל התווים. זאת כיוון ש: 2^7<35<2^8 וסופרים מ-0.

הקידוד של מיכל יעיל יותר כי לפיה נזדקק לקידוד שמשתמש גם באותיות וגם בספרות בכמות קטנה יותר של ביטים (**5,6**).

Q1

סעיף ג (המטריצה):

אסף ציפה שיודפס הפלט הבא: [[0, 0, 0], [0, 2, 0], [0, 0, 0]]. אבל זה לא קרה כי הבעיה בפונקציה make_mat: היא יוצרת העתק של רשימה **אחת** מסוימת ! זאת אומרת שכשמבצעים השמה בפונקציה set השינוי הוא על הרשימה היחידה עצמה, שפשוט מועתקת 3 פעמים (ואם היה n פעמים אז הייתה משוכפלת n פעמים) ולכן כשמתבצעת ההשמה אז היא על אותו אובייקט, וכל מיקום ברשימה מצביע על אותו האובייקט ולכן אנו עדים לשינוי המתואר ב"מטריצה".

מציע תיקון לכך:

```
def make_mat(n):
    return [ [0]*n  for i in range(n)]
```

כאן מבטיחים שנוצרות באמת שורות **שונות**.

והפלט יהיה:

```
print(set(m,1,1,2))
[[0, 0, 0], [0, 2, 0], [0, 0, 0]]
```

סעיף א- חסמים:

1. לא נכון.

$$2^{3logn} = 2^{logn^3} > 2^{logn^2} = n^2$$

וכמובן *log* פונקציה מונוטונית עולה, כלומר $logn^2 < logn^3$

2. לא נכון. $nlogn = logn^n$

$$n^n > n!$$

(n-1 times) → *n*n*n…*n > (n-1)(n-2)….1*

(הסבר): כל איבר בצד שמאל גדול מכל איבר בצד ימין (לאחר צמצום של *n*) ומספר ההכפלות בשני הצדדים זהה.

ולכן *log(n!)=O(nlogn)*

3. נכון. מספיק להביט באיבר האחרון בסכום: $a_k * n^k$. נסמן : $f(n)=n^k$, $g(n)= a_k * n^k$ כאשר a_k קבוע כלשהו. ואז לפי הגדרה מתקיים *f(n)=O(g(n))*

4. נכון.
לפי הגדרה, אם $f_1 = O(g_1)$ אזי מתקיים $f_1 \leq c_1 g_1$
אם $f_2=O(g_2)$ אזי מתקיים $c_2 g_2 \geq f_2$
ומכיוון שמדובר בגדלים חיוביים אזי: $c_1 c_2 g_1 g_2 \geq f_1 f_2$ כלומר לפי הגדרה
$f_1 f_2=O(g_1 g_2)$

5. נכון. מוכיח בה"כ $f_2 \circ f_1$ (שומר על הכיוונים)
נסמן $f_1(n)=k_1$. לפי הגדרה יתקיים $g_1(n)=c_1 k_1$. *c is constant. k is real number.*
נסמן $f_2(k_1)=k_2$. לפי הגדרה יתקיים $g_2(c_1 k_1) \geq g_2(k_1) = c_1 k_2$ כאשר המעבר של השוויון נובע מכך ש: $f_2=O(g_2)$ והמעבר של הא"ש נובע מכך ש-c_1 הוא קבוע שרק מגדיל את הקלט. ואז: $(f_2 \circ f_1)(n)=f_2(f_1(n))=f_2(k_1)=k_2$
$(g_2 \circ g_1)(n)=g_2(g_1(n))=g_2(c_1 k_1) \geq g_2(k_1)=c_2 k_2 > k_2$
ולכן $f_2 \circ f_1=O(g_2 g_1)$

6. א.
לא נכון. לדוגמה: *x is real number.* *g(n)=n**2 , f(n)=n ,*
וכמובן שלא מתקיים *g(n)=O(f(n))*

ב. לא נכון. לדוגמה: *x is a real number, g(n)=2n, f(n)=n*

אז מובן שמתקיים *f(n)=O(g(n))* אבל עבור הקבוע *c=3* יתקיים גם ש: *g(n)≤3f(n)*

כלומר *g(n)=O(f(n))*

<u>7.</u> לא נכון.

$$n^{\varepsilon} = 2^{log_2 n^{\varepsilon}} = 2^{\varepsilon log_2 n} \leq 2^{k log_2 n} = 2^{log_2 (logn)^k} = (logn)^k$$

$k \geq \varepsilon$ וגם 2^x היא מונוטונית עולה עבור *x>1* ויורדת עבור *x>1*

כלומר $logn^k \neq O(n^{\varepsilon})$

<u>סעיף ב:</u>

<u>פונקציה 1:</u>

$$\sum_{i=1}^{log_2 n} \frac{n}{2^i}$$

<u>פונקציה 2:</u>

$$(n-500) \quad \sum_{j=0}^{log_2(n-500)} \frac{n-500}{2^j}$$

<u>*Q6*</u>

<u>סעיף ד:</u>

הפונקציה בנויה משתי לולאות. בלולאה הראשונה *n* איטרציות מתבצעות ובכל איטרציה יש קריאה לפונקציה *string_to_int* . יחד עם הקריאה לפונקציה סיבוכיות מסדר גודל של *O* (*n*k*) . הלולאה השנייה עוברת על רשימה בגודל *5**k* . בכל איטרציה מתקיימת בדיקת תנאי שהיא *O(1)* . נקבל *O(5**k)* . בנוסף יהיו לכל היותר *n* קריאות לפונקציה *int_to_string* ולכן יהיה חסום על ידי *O(n*k)* , ועל זה יתווסף תוספת של *n* איברים לרשימה הסופית. כלומר סך סיבוכיות לולאה שנייה תהיה *O(5**k+ n*k +n)* ואם נסכם

את הסיבוכיות בשתי הלולאות נקבל $O(5^{**}k + n^*k + n^*k) = O(5^{**}k + n^*k)$ עד כדי כפל בקבוע.

<u>סעיף ו:</u>

הפונקציה מורכבת מ-2 לולאות מקוננות. הלולאה הפנימית הלולאה הפנימית רצה על רשימת הקלט כך שבכל איטרציה מתבצעת קריאה לפונקציה *string_to_int* שעולה לנו סיבוכיות זמן של $O(k)$. יחד עם ההשוואה של *int* והוספת איבר לרשימה יצא סה"כ $O(n^*k)$. באשר ללולאה החיצונית- היא מקיימת $5^{**}k$ איטרציות על כל n^*k איטרציה של הפנימית. סה"כ $n^*k^*5^k$, כלומר $O(n^*k^*5^k)$