תרגיל בית מספר 3 – אביתר שמש 322623182

שאלה 1 סעיף א 1) הטענה אינה נכונה.

נניח בשלילה שמתקיים, כלומר קיימים קבועים $c>0$, n_o ממשי כך שלכל $n>n_0$ מתקיים: $16^{logn} \leq c*n^3$

$16^{logn} = (2^4)^{logn} = (2)^{4logn} = 2^{log(n^4)} = n^4$

$c \geq n^4/n^3 = n$

הביטוי שקיבלנו אינו חסום ולכן c לא קבוע, לכן הטענה אינה נכונה.

שאלה 1 סעיף א 2) הטענה אינה נכונה.

$n^2log(n) + n(log(n))^2 = nlog(n)(n+logn)=log(n^n)(n+logn)=log(n^n)^{n+log(n)}=log(n)^{n^2}*logn^{nlogn}>1*logn^{nlogn}$

$=log(n^n)*log(n)=n*log(n)*log(n) = n*log(n^2) = O(n(logn)^2)$

הביטוי שהתחלנו ממנו גדול מביטוי שהסיבוכיות שלו היא $n(logn)^2$ (לא יכול להיות שווה כי לא קיים מספר טבעי שעבורו $log(n)^{n*n}$ שווה ל-1, לכן $n(logn)^2$ אינו חסם עליון של הביטוי, והטענה אינה נכונה.

שאלה 1 סעיף א 3) הטענה נכונה.

מהנתון נובע כי קיימים $c_1>0$ ו n_1 ממשי כך שלכל $n>n1$ מתקיים $f1 \leq c1*g1(n)$.

נפעיל בצורה איטרטיבית לכל k הפונקציות.

לכן, קיים n_{max} כך שלכל $n>n_{max}$ מתקיימת הנכונות לכל הפונקציות: $f_i \leq c_i*g_i(n)$

לכן, אם נסכום את כל הפונקציות, נקבל ביטוי קטן שווה ל סכום כל $c_i*g_i(n)$, ולכן הטענה נכונה.

שאלה 1 סעיף א 4) הטענה נכונה.

מהנתון נובע כי קיימים $c_1>0$ ו n_1 ממשי כך שלכל $n>n1$ מתקיים $f_1 \leq c1*g1(n)$.

נפעיל בצורה איטרטיבית לכל k הפונקציות.

לכן, קיים n_{max} כך שלכל $n>n_{max}$ מתקיימת הנכונות לכל הפונקציות: $f_i \leq c_i*g_i(n)$

ניקח את g_{max} המקיימת: $g_{max}=max\{g_1,g_2,\ldots,g_k\}$.

$(c_1+c_2+\ldots+c_k)(g_1(n)+g_2(n)+\ldots g_k(n) \leq (c_1+c_2+\ldots+c_k)*k(g_{max})$

לכן, הטענה נכונה.

שאלה 1 סעיף א 5) הטענה נכונה.

מהנתון נובע כי קיימים $c_1>0$ ו n_1 ממשי כך שלכל $n>n1$ מתקיים $f_1 \leq c_1*g_1(n)$. בנוסף, קיימים $c_2>0$ ו n_2 ממשי כך שלכל $n>n_2$ מתקיים $f_2 \leq c_2*g_2(n)$.

ניקח $n_{max}=n_1+n_2$. לכל $n>n_{max}$ מתקיים: $f_1*f_2 \leq c_1*g_1(n)*c_2*g_2(n)$.

ידוע כי $f_1*f_2 \leq (c_1*c_2)(g_1*g_2)$ ולכן $g1*g2 \epsilon O(f1*f2)$

לכן, הטענה נכונה.

שאלה 1 סעיף א 6) הטענה אינה נכונה

נפריך ע"י דוגמה נגדית.

$f_1(n)=n^2$. $f_2(n)=n^2$. לכן $f_1 \circ f_2 = (n^2)^2 = n^4$. אם $f_1 \circ f_2 = o(g_1 \circ g_2)$ אז קיים $c>0$ ו n_0 ממשי כך שלכל $n>n_0$ מתקיים:

$n^4 \leq c*n^2$, לכן $c \geq n^2$. דבר זה לא יכול להתקיים כי c קבוע ו-n^2 אינה חסומה.

שאלה 1 סעיף א 7) הטענה אינה נכונה.

נניח בשלילה שהטענה נכונה, לכן קיימים $c>0$ ו n_0 ממשי כך שלכל $n>n_0$ מתקיים:

$n^n \leq c*b^{nt}$, לכן $c \geq (n/b^t)^n$ n אינסופי לכן הביטוי n/b^t גדול מ-1 ולכן הביטוי $(n/b^t)^n$ אינו חסום, בסתירה לכך

ש c קבוע. לכן, הטענה אינה נכונה.

שאלה 1 סעיף ב) נניח בשלילה שהטענה נכונה. לכן קיימים $c>0$ ו n_0 ממשי כך שלכל $n>n_0$ מתקיים:
$(2^k+\varepsilon)^{\log(n)} \leq c * n^k$ -> $(2^k+\varepsilon)^{\log(n)} \leq c * 2^{\log(n)\wedge k}$ -> $(2^k+\varepsilon)^{\log(n)} \leq c*(2^k)^{\log(n)}$

$2^k+\varepsilon \leq c^{1/\log(n)}*2^k$

כאשר n שואף לאינסוף, הביטוי $c^{1/\log(n)}$ שואף ל -1 לכן:

$2^k+\varepsilon \leq 2^k$ -> $\varepsilon \leq 0$.

סתירה לכך ש ε חיובי, לכן, הטענה אינה נכונה.

שאלה 1 סעיף ג 1) הסיבוכיות של פונקציית len היא $O(1)$.

לולאת ה-while תרוץ במקרה הגרוע $n//2$ פעמים במקרה הגרוע שבו n זוגי.

הבדיקה האם $n>0$ תיבדק $n//2$ פעמים, ככמות הריצות של לולאת ה-while, אך הסיבוכיות של פעולה זו היא $O(1)$.

בכל ריצה של ריצת ה while, לולאת ה-for רצה n-1 פעמים(בהתחשב ב n הנוכחי) ובמקרה הגרוע תכניס n-1 פעמים את i לתוך הרשימה. בדיקת האם i נמצא ברשימה היא בסיבוכיות $O(n)$. הסיבוכיות של append היא $O(1)$ אבל היא קורית במקרה הגרוע בכל ריצה n-1 פעמים.

הלולאה החיצונית (while) תרוץ $\log(n)$ פעמים.

ה- for בתוכו, מתנהג כסדרה הנדסית. בפעם הראשונה רץ $n//2$ פעמים, ובכל פעם כמות הפעמים קטנה פי 2, לכן $q=1/2$. כמות האיברים ב"סדרה" ההנדסית היא $\log(n)$ פעמים.

לכן, ניתן להשתמש בנוסחת סכום סדרה הנדסית:

$(n/2)((1/2)^{\log(n)}-1)/(1/2-1) =$

$=((n/2)*(1/2)^{\log n}-(n/2))/(1/2-1)$

$=((n/2)*(1/n)-(n/2))/(-1/2)$

$=((n/2)(1/n -1))/(-1/2)$

$= ((1/2)-(n/2))/(-1/2)$

$= (1-n)/-1$

$=n-1 =O(n)$

הלולאות מקוננות ולכן הסיבוכיות של כל הפונקציה היא: $O(n)*O(n) = O(n^2)$

שאלה 1 סעיף ג 2) מחוץ ללולאה, השמת n היא $O(1)$, כי הסיבוכיות של len(L) היא $O(1)$.

יצירת res היא גם $O(1)$, השמת רשימה ריקה

הלולאה החיצונית תרוץ n-501 פעמים.

השמת m בתוך לולאה זו היא $O(1)$, כי זו פעולה מתמטית של מחלקה מובנית בפייתון.

הלולאה הפנימית תרוץ בכל פעם log(n) פעמים ותבצע את הפעולות הבאות:

השמת k=1, $O(1)$.

לולאת while שתרוץ log(n) פעמים(כי k מוכפל ב-2 בכל פעם).

res.append(k) בסיבוכיות $O(1)$.

לכן, כאשר נסכום את הלולאה הפנימית, נקבל סכום מ 500 עד n של log(n) * log(n). ראינו כי O(nlog(n))=O(n!), וכך נתייחס לסכום מ 500 עד n.

לכן: $O(1)+O(1)+O(n\log(n))+O(n\log(n)*\log(n)) = O(n\log(n)*\log(n)) = O(n(\log(n))^2)$

שאלה 2 סעיף א) כל מספר שניתן לייצוג ב 32 ביטים, ניתן לייצוג גם ב64 ביטים , כי:

מבחינת סימן, התנהגותם זהה.

מבחינת האקספוננט, ב 32 ביטים האקספוננט מייצג חזקות של 2 מ 127- עד 127, בעוד שב 64 ביטים האקספוננט מייצג חזקות של 2 מ 1023- ועד 1023, ובפרט מ 127- ועד 127.

מבחינת ה- fraction, ב 32 ביטים מיוצגות כל החזקות השליליות של 2 מ - 1- ועד 23-, בעוד שב 64 ביטים הfraction- מיוצגות כל החזקות השליליות של 2 מ 1- ועד 52-, ובפרט בין 1- ל 23-.

לכן, כל מספר שמיוצג ב32 ביטים ניתן לייצוג ב 64 ביטים.

שאלה 2 סעיף ב) $1+\frac{1}{2^{30}}$. מספר זה אינו ניתן לייצוג ב32 ביטים כי ה fraction מגיע עד 2^{-23}, בעוד שב 64 ביטים המספר ניתן לייצוג.

שאלה 2 סעיף ו 1) המספרים בטווח $[2^k,2^{k+1})$ מיוצגים ע"י אותו אקספוננט, וההבדל ביניהם יהיה ה fraction.

נבדוק כמה מספרים שונים ה-fraction ב32 ביטים יכול לייצג.

אנחנו יודעים שכמות הביטים ב-fraction במקרה זה הוא 23.

כל ביט יכול לייצג 0 או 1. לכן, ניתן לייצג 2^{23} מספרים בטווח $[2^k,2^{k+1})$.

שאלה 2 סעיף ו 2) נחלק כל קטע של 2^k עד 2^{k+1} בטווח 3 עד 10.

3-4: בקטע זה יש בדיוק מחצית מהמספרים בטווח 2-4. לכן, נשתמש בהוכחה מסעיף ו1:

$2^{23}/2 = 2^{22}$

4-8: בקטע זה כל המספרים מיוצגים במלואם, לכן : 2^{23}.

8-10 – מיוצג כרבע מהקטע 8-16(החזקה הבאה של 2). לכן: $2^{23}/4 = 2^{23}*2^{-2} = 2^{21}+1$.

נחבר את כמות המספרים בכל קטע: $2^{22}+2^{23}+2^{21} = 2^{21}(1+2+4) = 7*2^{21}+1$

שאלה 2 סעיף ו 3) במקרה זה איננו יכולים להשתמש באופן מלא בחישוב שנעשה בסעיף א', כי אין חזקה של 2 ששווה ל-0. לכן, נשתמש באקספוננטים. מבחינת סימן, יש רק אפשרות אחת כי מספר שלילי לא ייתן לנו 0.

עבור האקספוננט, יש לנו 127 אפשרויות, מכיוון וצריך את החזקה של הביטוי שלילית(כדי שיהיה קטן מ -1).

עבור ה- fraction, לכל אקספוננט נוכל להשתמש בכל ה-fractions, לכן 2^{23} עבור כל אקספוננט.

על מנת לא לקבל את 0, נחסיר מספר אחד. כדי לקבל את 1, נוסיף מספר אחד שהוא 2^{0}.

לכן, בסה"כ: $127*2^{23} +1-1 = 127*2^{23}$

שאלה 2 סעיף ו 4) ראינו בתרגול הדגמה עבור 2 ביטים ל-fraction. נשתמש בה על המקרה שלנו: 23 ביטים לייצוג ה-fraction, בעוד ה-exponent הוא 32. ניקח את ההפרש בין 32*(1+ רק אפסים) ל 32*(1+אפסים ו -1 בביט האחרון). כלומר: $2^{-18} = 2^{5}*2^{-23} = 32*(2^{-23}) = 32*(1+2^{-23}-1)$.

לכן, מידת הדיוק עבור מספרים בקטע [32,64) היא 2^{-18}.

שאלה 3 סעיף ב 2)

N	Result
100	2.67
1000	2.7
10000	2.7109

שאלה 3 סעיף ב 3) במקרה הגרוע(ושסבירותו נמוכה מאוד), הלולאה תרוץ לעד, כי טכנית יכול להתקיים מקרה בו נגריל לעד 0 ב random.random(), והסכום לא יעבור את 1 לעולם, וכך הלולאה תרוץ לעד ☹.

שאלה 4 סעיף א 2) סיבוכיות זמן הריצה היא בדומה ל binary_search(רק הוספת מספר בדיקות קטנות שיותאמו לכך שהרשימה כמעט ממוינת ולא ממוינת). לכן, סיבוכיות זמן הריצה שלה היא O(logn).

שאלה 4 סעיף ב 2) סיבוכיות זמן הריצה היא O(n), אנחנו עוברים על כל איבר בלולאה פעם אחת אל מול האיבר הבא אחריו.

לכן, הסיבוכיות היא כאורך הרשימה lst.

שאלה 4 סעיף ג 1) כן.

אם ברשימה יש רק איבר אחד, הוא מינימום מקומי.

אם יש 2 איברים, אחד מהם לפחות חייב להיות מינימלי.

אם הרשימה ממויינת, האיבר הראשון הוא איבר מינימלי.

אם כל איברי הרשימה שווים, אז כולם מינימום מקומי.

כעת, אם הרשימה לא עומדת באף אחת מהתנאים לעיל, נוכיח באמצעות אינדוקציה.

אם האיבר הראשון לא מינימום מקומי, כלומר האיבר השני קטן ממנו. אם זה יימשך עד סוף הרשימה, האיבר האחרון הוא מינימום מקומי, כי הוא קטן מהאיבר שלפניו ועומד בתנאי. אם מצאנו במיקום k כלשהו כאשר k<len(list), אזי k הוא מינימום מקומי, כי הוא גדול מהאיבר שלפניו וקטן מהאיבר שאחריו.

לכן, לכל רשימה סופית של איברים, חייב להיות מינימום מקומי.

שאלה 4 סעיף ג 3) סיבוכיות זמן הריצה היא $O(n)$, כי במקרה הגרוע אנחנו עוברים על כל איבר בלולאה פעם אחת אל מול האיבר לפניו ואחריו.

לכן, הסיבוכיות היא כאורך הרשימה lst.

שאלה 5 סעיף ד) דרישת הסיבוכיות לפונקציה זו היא $O(kn + 5^k)$.

אנתח את הסיבוכיות עבור המקרה הגרוע בפונקציה שלי.

ראשית, אנחנו מאתחלים רשימה באורך 5^k, לכן הסיבוכיות של פעולה זו היא 5^k.

לאחר מכן, אנחנו עוברים על כל האיברים ברשימה lst. מהגדרת השאלה, n איברים.

על כל s ברשימה, אנחנו מחשבים מה ערך ה-int שלו מהפונקציה string_to_int מסעיף א.

הסיבוכיות של פונקציה זו היא $O(k)$:

הגדרת מילון, $O(1)$.

השמת משתנה rslt, $O(1)$.

השמת משתנה k, $O(1)$, כי len(s) גם הוא עם סיבוכיות $O(1)$.

הלולאה תרוץ k פעמים (על כל תו ב s), ותמקמם ב lst באינדקס ספציפי את הערך המילוני של התו הנוכחי.

לכן, $O(k)$.

לאחר כן, תעדכן את n, $O(1)$.

ומחזירה את ערכי הרשימה lst כ str, לכן $O(k)$.

בסה"כ, $O(1)+O(1)+O(1)+O(1)+O(k)+O(1)+O(k) = O(k)$.

נחזור לפונקציה המקורית. לאחר שמחושב הערך ($O(k)$) ומושם למשתנה num($O(1)$), בודקים האם הערך במיקום ספציפי ב lst1 הוא None.

בגלל שעוברים n פעמים על הרשימה, יוצא $O(nk)$.

המקרה הגרוע והטוב כאן הם זהים, כי אם לא, נאריך את הרשימה הנוכחית ב lst1[num] בעוד איבר זהה לנוכחי, ואם לא אז תושם רשימה רק עם S. לכן $O(1)$.

לאחר מכן, נגדיר רשימה חדשה lst_out, $O(1)$.

לאחר מכן, נעבור על כל איברי הרשימה lst1, שאורכה $O(5^k)$. על כל איבר ברשימה זו תערך בדיקה האם הוא None, ובמקרה הגרוע יש n איברים כאלו, ותוסיף ל- lst_out את האיברים הללו. לכן, $O(n)$.

בחלק האחרון, אנחנו עוברים על כל הרשימה lst_out פעם נוספת, ובודקים אם יש בה רשימות שמורכבות מכמה איברים, כלומר ברשימה המקורית הופיעה אותו s כמה פעמים. המקרה הגרוע והמקרה הטוב זהים כאן, כי אם כל האיברים ברשימה המקורית אותו עבר, אז ב for הקודם היינו שמים רק איבר אחד בתוך lst_out, ועכשיו משימים אותו n פעמים ברשימה.

במקרה שבחרנו, כל ה-s ברשימה המקורית שונים, לכן $O(n)$.

בסופו של דבר, מחזירים את הרשימה.

סיבוכיות כוללת: $O(5^k)+O(n)+O(nk)+O(1)+O(1)+O(1)+O(5^k)+O(n)+O(n) = O(5^k+nk)$

שאלה 5 סעיף ו)מבחינת סיבוכיות הזיכרון, לא השתמשתי ברשימת עזר באורך k. לכן, סיבוכיות הזיכרון היא כסיבוכיות הזיכרון של string_to_int(), שהיא:

יצירת מילון – $O(1)$.

יצירת lst – $O(1) = O(k)$

לכן, סיבוכיות הזיכרון של הפונקציה שלי היא $O(k)$.

מבחינת סיבוכיות הזמן. השמת lst_out היא $O(1)$.

השמת num היא $O(1)$.

לולאה שרצה 5^k פעמים, ובתוכה לולאה שרצה n פעמים. בתוך הלולאה הפנימים הסיבוכיות היא:

הפעולה string_to_int בסיבוכיות $O(k)$(כמו שהוכחתי בסעיף א). בדיקת if ראשונה, $O(1)$. הפעולה append בתוכה, גם היא $O(1)$. בדיקת if שנייה גם היא $O(1)$, כי סיבוכיות len היא $O(1)$.

לכן, הסיבוכיות של הלולאה הפנימית היא: $O(k)+O(1)+O(1)+O(1) = O(k)$

לכן, בסך הכל: $O(1)+O(1) + (5^k)*n*O(k) = O(5^k*nk)$.

לכן, הפונקציה שכתבתי עומדת בדרישות סיבוכיות הזמן והזיכרון של השאלה.