אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, חורף 2021

<u>תרגיל בית מספר 3 - להגשה עד 1 בדצמבר בשעה 23:55</u>

קיראו בעיון את הנחיות העבודה וההגשה המופיעות באתר הקורס, תחת התיקייה assignments. חריגה מההנחיות תגרור ירידת ציון / פסילת התרגיל.

<u>הגשה</u>:

- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- השתמשו בקובץ השלד skeleton3.py כבסיס לקובץ ה py אותו אתם מגישים.
  לא לשכוח לשנות את שם הקובץ למספר ת"ז שלכם לפני ההגשה, עם סיומת py.
- בסה"כ מגישים שני קבצים בלבד. עבור סטודנטית שמספר ת"ז שלה הוא 012345678 הקבצים שיש להגיש הם hw3_012345678.pdf ו- hw3_012345678.py.
- מכיוון שניתן להגיש את התרגיל בזוגות, עליכם בנוסף למלא את המשתנה SUBMISSION_IDS שבתחילת קובץ השלד. רק אחת הסטודנטיות בזוג צריכה להגיש את התרגיל במודל.
- הקפידו לענות על כל מה שנשאלתם.
- תשובות מילוליות והסברים צריכים להיות תמציתיים, קולעים וברורים.
  להנחיה זו מטרה כפולה:
  1. על מנת שנוכל לבדוק את התרגילים שלכם בזמן סביר.
  2. כדי להרגיל אתכם להבעת טיעונים באופן מתומצת ויעיל, ללא פרטים חסרים מצד אחד אך ללא עודף בלתי הכרחי מצד שני. זוהי פרקטיקה חשובה במדעי המחשב.

אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, חורף 2021

שאלה 1

א. הוכיחו או הפריכו את הטענות הבאות. ציינו תחילה בברור האם הטענה נכונה או לא, ואח"כ הוכיחו / הפריכו באופן פורמלי תוך שימוש בהגדרת $O(\cdot)$.

<u>הנחיה</u>: **יש להוכיח / להפריך כל סעיף בלא יותר מ- 4 שורות**.
הפתרונות הם קצרים, ואינם דורשים מתמטיקה מתוחכמת. אם נקלעתם לתשובה מסורבלת וארוכה, כנראה שאתם לא בכיוון. לאורך השאלה n הוא משתנה ואינו קבוע, כל הפונקציות הן מהטבעיים לעצמם ($f: \mathbb{N} \rightarrow \mathbb{N}$) והאופרטור $\log$ הוא לפי בסיס 2.

1. $n \log n = O(\log n!)$
2. בהינתן מספר חיובי קבוע k, קבוע חיובי $a_k \in \mathbb{R}^+$ וקבועים $a_0, \dots, a_{k-1} \in \mathbb{R}$ מתקיים:
$$n^k = O\left(\sum_{i=0}^{k} a_i n^i\right)$$
3. אם $f_1(n) = O\big(g_1(n)\big)$ וגם $f_2(n) = O\big(g_2(n)\big)$ אז $f_1(n)/f_2(n) = O\big(g_1(n)/g_2(n)\big)$ (ניתן להניח שהפונקציות במכנה לא מתאפסות)
4. אם $f_1(n) = O\big(g_1(n)\big)$ ו $f_2(n) = O\big(g_2(n)\big)$ אז $f_1 \circ f_2(n) = O\big(g_1 \circ g_2(n)\big)$
תזכורת: $f \circ h(n) = f\big(h(n)\big)$
5. טרנזיטיביות: אם $f(n) = O\big(g(n)\big)$ וגם $g(n) = O\big(h(n)\big)$ אז $f(n) = O\big(h(n)\big)$
6. לכל שני קבועים $0 < \epsilon < 1$ ו-$k \geq 1$ מתקיים: $(\log n)^k = O(n^\epsilon)$ (רמז: השתמשו בהגדרת הגבול שראינו בתרגול).
7. אם $f_1(n) = O\big(g_1(n)\big)$ וגם $f_2(n) = O\big(g_2(n)\big)$ אז $f_1(n) - f_2(n) = O\big(g_1(n) - g_2(n)\big)$

ב. לכל אחת משתי הפונקציות הבאות, נתחו את סיבוכיות זמן ריצתה במקרה הגרוע כתלות ב- n (אורך הרשימה L).
הניחו כי פעולות אריתמטיות ופעולות append רצות בזמן $O(1)$. ציינו את התשובה הסופית, ונמקו. על הנימוק להיות קולע, קצר וברור, ולהכיל טיעונים מתמטיים או הסברים מילוליים, בהתאם לצורך.
על התשובה להינתן במונחי O(…), ועל החסם להיות הדוק ככל שניתן. למשל, אם הסיבוכיות של פונקציה היא O(n) ובתשובתכם כתבתם O(nlogn), התשובה לא תקבל ניקוד (על אף שפורמלית O הוא חסם עליון בלבד).

.1

```
def f1(L):
    n = len(L)
    while n > 0:
        n = n // 2
        for i in range(n):
            if i in L:
                L.append(i)
    return L
```

.2

```
def f2(L):
    n = len(L)
    res = []
    for i in range(500, n):
        m = math.floor(math.log(i))
        for j in range(m):
            k=1
            while k<n:
                k*=2
                res.append(k)
    return res
```

ג. להלן שתי פונקציות שמקבלות רשימה ואובייקט כלשהו, ומוסיפות את האובייקט לרשימה.

```
def add_to_list_1(lst, item):
    lst = lst + [item]
    return lst

def add_to_list_2(lst, item):
    lst += [item]
    return lst
```

אסף הריץ את הפקודה הבאה:

```
l = [1,2,3]
for e in l:
    l = add_to_list_1(l, "a")
```

וקיבל כפי שציפה, את הרשימה [1, 2, 3, 'a', 'a', 'a']. ואולם, כאשר קרא לפונקציה `add_to_list_2` במקום `add_to_list_1`, גילה שהתוכנית נכנסת ללולאה אינסופית. הסבירו <u>בקצרה</u> מדוע זה קרה, ומדוע קיים הבדל בין הפעלת שתי הפונקציות. היעזרו במה שלמדנו על מודל הזיכרון של פייתון.

<u>שאלה 2</u>

א. בהרצאה ראינו את הפונקציה text_2_bits שממירה מחרוזת טקסט למחרוזת שמכילה את הייצוג הבינארי של הטקסט.

a. השלימו את הפונקציה text_2_16bits בשלד, שמחזירה מחרוזת שהיא התוצאה של המרת כל תו במחרוזת הטקסט לייצוג בינארי עם בדיוק 16 ביטים (עבור תו בעל ייצוג של פחות מ 16 ביטים יש לרפד באפסים משמאל כך שיהיו בדיוק 16 ביטים בייצוג שלו). לדוגמא, עבור מחרוזת הטקסט "~b"

הפונקציה תחזיר: '00000000011111100000000001100010'

שימו לב שקידוד Unicode הוא קידוד באורך לא קבוע (variable length code), כפי שהוסבר בשיעור. יחד עם זאת בשאלה זו אתם מתבקשים לממש קידוד פשוט יותר בו כל תו מקודד ע"י בדיוק 16 ביטים.

b. השלימו את הפונקציה ההפוכה bits_2_text בשלד, שמקבלת מחרוזת ביטים שהיא תוצאה של הפונקציה text_2_16bits ומחזירה את מחרוזת הטקסט המקורית. כלומר, לכל מחרוזת text הפונקציה מקיימת

bits_2_text(text_2_16bits(text)) == text

ב. ממשו את הפונקציה float_add(a,b) בשלד, אשר מקבלת שתי מחרוזות שמכילות ייצוג נקודה צפה (64 ביט כפי שנלמד בכיתה) ומחזירה מחרוזת שמכילה ייצוג נקודה צפה של תוצאת החיבור של המספרים המיוצגים ע"י מחרוזות הקלט.

<u>הנחיות:</u>

a. ניתן להניח שאפשר לייצג את תוצאת החיבור באופן מדויק בנקודה צפה.
b. יש לממש את הפעולה ללא שימוש בפעולות מובנות של נקודה צפה. כלומר, אסור להמיר את הייצוג למשתנים מטיפוס float או לבצע כפל או חלוקה של מספרים מכל סוג שהוא.
c. ניתן להשתמש בחיבור וחיסור של מספרים שלמים (int).
d. ניתן להשתמש בפעולות הזזה וריפוד בינאריות מהצורה ('0' * a) + bin_num + ('0' * b) כאשר a, b הם מספרים מטיפוס int.
e. ניתן לבצע המרה של ייצוג בינארי למשתנה int (לדוגמא (int("10111", 2)) ושל משתנה int למחרוזות בייצוג בינארי (לדוגמא, bin(17)).

<u>דוגמא:</u>

מספר ב float

a = "0100000000101000000000000000000000000000000000000000000000000000" => 12.0

b = "0011111111101000000000000000000000000000000000000000000000000000" => 0.75

float_add(a, b) = "0100000000101001100000000000000000000000000000000000000000000000" => 12.75

שאלה 3

נתאר את אלגוריתם המיון הבא עבור רשימה של n איברים ומספר שלם חיובי k נתון:

1. ניצור $\left\lceil \frac{n}{k} \right\rceil$ רשימות קטנות באורך k כל אחת (למעט הרשימה האחרונה שאולי קטנה יותר) כך שכל איבר מרשימת הקלט יופיע ברשימה קטנה אחת בדיוק.
2. נמיין כל רשימה קטנה באמצעות מיון בחירה (selection sort).
3. נמזג את הרשימות הקטנות לרשימה אחת חדשה, וזו תהיה רשימת הפלט.

א. השלימו את הפונקציה generate_sorted_blocks(lst, k) שמקבלת כקלט רשימה lst של n איברים ומספר $k \leq n$, ומחזירה רשימה חדשה שבה האיבר ה-i הינו הרשימה הקטנה ה-i-ית שמתקבלת משלבים 1 ו-2 לעיל. כלומר הפלט הינו רשימה של $\left\lceil \frac{n}{k} \right\rceil$ רשימות קטנות ממויינות. יש להשתמש ב selection_sort מהכיתה ללא שינוי (מופיעה כבר בקובץ השלד).

```
>>> import random
>>> lst = [random.choice(range(1000)) for i in range(10)]
>>> lst
[610, 906, 308, 759, 15, 389, 892, 939, 685, 565]
>>> generate_sorted_blocks(lst, 2)
[[610, 906], [308, 759], [15, 389], [892, 939], [565, 685]]
>>> generate_sorted_blocks(lst, 3)
[[308, 610, 906], [15, 389, 759], [685, 892, 939], [565]]
>>> generate_sorted_blocks(lst, 10)
[[15, 308, 389, 565, 610, 685, 759, 892, 906, 939]]
```

ב. מהי סיבוכיות זמן הריצה של generate_sorted_blocks כפונקציה של n, k. תנו ביטוי במונחי $O(\cdot)$ הדוק ככל האפשר, ונמקו.

ג. ממשו בקובץ השלד את הפונקציה merge_sorted_blocks(lst) שמקבלת כקלט רשימה lst של m רשימות קטנות ממויינות ומחזירה רשימה אחת שמכילה את אוסף כל האיברים מכל הרשימות בסדר ממויין. נסמן ב m את אורך רשימת הקלט, כלומר את מספר הרשימות הקטנות, ונסמן ב k את האורך המקסימלי של רשימה קטנה.

על הפתרון להיות מסיבוכיות זמן $O(m \cdot k \cdot \log m)$. יש להשתמש ב merge מהכיתה ללא שינוי (הפונקציה מופיעה כבר בקובץ השלד).

```
>>> import random
>>> block_lst1 = [[610, 906], [308, 759], [15, 389], [892, 939], [565, 685]]
>>> merge_sorted_blocks(block_lst1)
[15, 308, 389, 565, 610, 685, 759, 892, 906, 939]
```

ד. הוכיחו שהפונקציה merge_sorted_blocks שכתבתם אכן רצה בסיבוכיות הזמן המבוקשת. לצורך נוחות, ניתן להניח ש m הוא חזקה שלמה של 2.

ה. להלן מימוש הפונקציה sort_by_block_merge(lst, k) שמממשת את האלגוריתם הנ"ל:

```
def sort_by_block_merge(lst,k):
    return  merge_sorted_blocks(generate_sorted_blocks(lst, k))
```

מהי סיבוכיות זמן הריצה הכוללת של sort_by_block_merge כפונקציה של n, k? הסבירו בקצרה.

ו. בקרוב נלמד על שיטת מיון יעילה הקרויה merge_sort שממיינת רשימה של n איברים בסיבוכיות זמן $O(n \log n)$.
מהו הערך האסימפטוטי (במונחי $O(.)$) הגדול ביותר של k כפונקציה של n כך שסיבוכיות זמן הריצה של האלגוריתם הנ"ל (sort_by_block_merge) תהיה זהה לזו של merge_sort?

<u>שאלה 4</u>

בשאלה זו הניחו כי פעולות אריתמטיות והשוואת מספרים מתבצעות בזמן קבוע, וכי הקלט תקין.
כמו כן, על זמן הריצה של המימוש שלכם בכל אחד מהסעיפים להיות נמוך ככל הניתן במונחים אסימפטוטיים.

א. רשימה L היא <u>כמעט ממוינת</u> אם כל איבר בה נמצא לכל היותר במרחק אינדקס אחד מהמיקום שלו ברשימה הממוינת. כלומר, אם arg_$sort(i)$ הוא האינדקס של $L[i]$ ברשימה הממוינת sorted(L) אז

$$\text{arg\_}sort(i) \in \{i-1, i, i+1\}$$

לדוגמא, הרשימה lst_example = [2, 1, 3, 5, 4, 7, 6, 8, 9] היא כמעט ממוינת.

a. השלימו את הפונקציה find בשלד, שמקבלת את רשימה כמעט ממוינת L ומספר שלם s ומחזירה את האינדקס i כך ש $L[i] == s$ אם s הוא איבר ברשימה LR, אחרת מחזירה $None$. למשל, עבור L מהדוגמה ו-$s = 5$, הפונקציה תחזיר 3 (כי המספר 5 נמצא באינדקס 3 ברשימה L). עבור $s = 11$ הפונקציה תחזיר `None` (כי המספר 11 לא נמצא ברשימה L).

b. מה היא סיבוכיות זמן הריצה? הסבירו בקצרה.

ב. נרצה למיין רשימה <u>כמעט ממוינת</u> ללא שימוש ברשימת עזר.

a. השלימו את הפונקציה sort_from_almost(lst) בשלד, שמקבלת רשימה כמעט ממוינת וממיינת אותה ללא שימוש ברשימת עזר (או כל מבנה בעל גודל יותר מ O(1)). שימו לב, הפונקציה לא מחזירה פלט אלא רק ממיינת את רשימת הקלט. לדוגמא, אם נריץ את הפונקציה על lst_example שמוגדרת בתחילת הסעיף אז אחרי הריצה נקבל ש lst_example ממויינת.

b. הסבירו בקצרה את הפתרון שלכם ואת סיבוכיות זמן הריצה שלו.

ג. מינימום מקומי ברשימה L הוא כל אינדקס i שקטן או שווה לשכניו המידיים. כלומר, כל אינדקס המקיים:

$$(i == 0 \; or \; L[i] \le L[i-1]) \; and \; (i == n-1 \; or \; L[i] \le L[i+1])$$

לדוגמא, ברשימה [5, 6, 7, 5, 1, 1, 99, 100] האינדקסים 5 ,4 ,0 הן מינימום מקומי.

a. האם בכל רשימה של מספרים יש מינימום מקומי? נמקו את תשובתכם.

b. השלימו את הפונקציה find_local_min בשלד, שמקבלת רשימה של מספרים (לא ממוינים וייתכנו חזרות) ומחזירה אינדקס i של מינימום מקומי (אם יש יותר מאחד אז ניתן לבחור שרירותית). למשל, עבור הרשימה מהדוגמה תשובה של 4 ,0 או 5 תהיה תקינה.

c. מהי סיבוכיות זמן הריצה? הסבירו בקצרה.

<u>שאלה 5</u>

בכיתה ראינו את האלגוריתם מיון-בחירה (selection sort) למיון רשימה נתונה. האלגוריתם כזכור רץ בסיבוכיות זמן $O(n^2)$ עבור רשימה בגודל n. בקרוב נראה גם אלגוריתם מיון-מהיר יעיל יותר (quicksort), שרץ בסיבוכיות זמן ממוצעת $O(n\log n)$. לפעמים, כאשר יש לנו מידע נוסף על הקלט, אפשר למיין בסיבוכיות זמן טובה מזו. למשל, בשאלה זו, נעסוק במיון של רשימה שכל איבריה מוגבלים לתחום מצומצם יחסית: מחרוזות באורך k , עבור k>0 נתון כלשהו, מעל האלפבית a,b,c,d,e שמכיל 5 תווים.

ההשוואה בין זוג מחרוזות תהיה לקסיקוגרפית, כלומר השוואה מילונית רגילה.

<u>הערות</u>:

1. בשאלה זו אסור להשתמש בפונקציות מיון מובנות של פייתון.
2. בניתוח הסיבוכיות בשאלה זו נניח שהשוואה של זוג מחרוזות באורך k מבצעת בפועל השוואה של התווים של המחרוזות משמאל לימין, ובמקרה הגרוע תהיה מסיבוכיות זמן $O(k)$.
3. לשם פשטות ניתוח הסיבוכיות נתייחס הן לפעולות אריתמטיות והן לפעולות העתקה של מספרים ממקום למקום בזכרון כפעולות שרצות בזמן קבוע.

א. השלימו בקובץ השלד את הפונקציה string_to_int(s) שמקבלת כקלט מחרוזת s באורך k <u>בדיוק</u> שמורכבת מהתווים a,b,c,d,e ומחזירה מספר שלם בין 0 ל 5^k-1 כולל, המייצג את הערך הלקסיקוגרפי היחסי של המחרוזת. על הפונקציה להיות חד-חד-ערכית. סיבוכיות הזמן שלה צריכה להיות $O(k)$.

ב. השלימו בקובץ השלד את הפונקציה int_to_string(k, n) , ההפוכה לזו מסעיף א׳, שמקבלת כקלט מספר שלם k גדול מ0 , וכן מספר שלם n בין 0 ל 5^k-1 כולל ומחזירה מחרוזת s באורך k <u>בדיוק</u> שמורכבת מהתווים a,b,c,d,e . גם על פונקציה זו להיות חד-חד-ערכית. סיבוכיות הזמן שלה צריכה להיות $O(k)$.

שימו לב שפונקציה זו צריכה לקיים לכל $0\le i\le 5^k-1$:

string_to_int(int_to_string(k, i)) == i

דוגמת הרצה:

```
>>> for i in range(5**3):
        if string_to_int(int_to_string(3, i)) != i:
            print("Problem with ", i)
>>> alphabet = ["a","b","c","d","e"]
>>> lst = [x+y+z for x in alphabet for y in alphabet for z in alphabet]
>>> for item in lst:
        if int_to_string(3, string_to_int(item)) != item:
            print("Problem with ", item)
>>> #Nothing was printed
```

ג. השלימו בקובץ השלד את הפונקציה sort_strings1(lst, k) שמקבלת כקלט רשימה lst של n מחרוזות כמתואר ומספר חיובי k כך שכל מחרוזת ברשימה הינה באורך k בדיוק. (הניחו כי הקלט תקין ואין צורך לבדוק את תקינותו.) על הפונקציה להחזיר רשימה <u>חדשה</u> ממויינת בסדר עולה (ולא לשנות את lst עצמה). בנוסף לרשימת הפלט שהיא בגודל n כמובן, על הפונקציה להשתמש ברשימת עזר (list) בעלת 5^k איברים.

עליכם להשתמש בפונקציות מסעיפים א', ב'.

על הפונקציה sort_strings1 להיות מסיבוכיות זמן $O(kn + 5^k)$.

ד. בקובץ ה pdf הסבירו מדוע הפונקציה מסעיף ג' עומדת בדרישות הסיבוכיות.

ה. השלימו בקובץ השלד את הפונקציה sort_strings2(lst, k) שמקבלת קלטים כמו הפונקציה מסעיף ג', ובדומה לפונקציה הקודמת עליה להחזיר רשימה חדשה ממויינת בסדר עולה (ולא לשנות את lst עצמה).

הפעם מותר להשתמש בזכרון עזר מגודל $O(k)$, לא כולל רשימת הפלט שעליכם לייצר שגודלה הוא n. על הפונקציה להיות מסיבוכיות זמן $O(5^k \cdot kn)$.

ו. בקובץ ה pdf הסבירו מדוע הפונקציה מסעיף ה' עומדת בדרישות סיבוכיות הזמן והזיכרון.

דוגמת הרצה:

```
>>> import random
>>> k = 4
>>> lst = ["".join([random.choice(["a", "b", "c", "d", "e"]) for i in
range(k)]) for j in range(10)]
>>> lst
['aede', 'adae', 'dded', 'deea', 'cccc', 'aacc', 'edea', 'becb', 'daea',
'ccea']
>>> sort_strings1(lst, k)
['aacc', 'adae', 'aede', 'becb', 'cccc', 'ccea', 'daea', 'dded', 'deea',
'edea']
>>> sort_strings2(lst, k)
['aacc', 'adae', 'aede', 'becb', 'cccc', 'ccea', 'daea', 'dded', 'deea',
'edea']
>>> sorted(lst) == sort_strings1(lst, k)
True
>>> sorted(lst) == sort_strings2(lst, k)
True
```

שאלה 6

השתמשו בשיטת ה- bisection שלמדתם בכיתה כדי לקרב את ערכו של יחס הזהב, $\phi = (1+\sqrt{5})/2$. על 3 הספרות הראשונות שמימין לנקודה להיות נכונות, אך מותר שתהיה טעות החל בספרה הרביעית מימין לנקודה והלאה. צרפו לקובץ ה pdf -את ההרצה שביצעתם ואת התוצאה, והסבירו מדוע הטעות חסומה כנדרש. בפרט, בחרו ערך TOL מספיק גדול, והסבירו מדוע בחרתם בו.

<u>הערה</u>: אסור להשתמש בפונקציית השורש של פייתון לצורך החישוב.

סוף