מס' ת"ז_____ מס' מחברת_____

מבחן בקורס מבוא מורחב למדעי המחשב, CS1001.py

ביה"ס למדעי המחשב, אונ' תל אביב

סמסטר א' 2015-16, מועד ב, 18/3/2016

מרצים: פרופ' עמירם יהודאי, אמיר רובינשטיין

מתרגלות: יעל ברן, מיכל קליינבורט

משך הבחינה: 3 שעות.

חומר עזר מותר: 2 דפי עזר (דו צדדיים) או 4 עמודים (חד צדדיים) בגודל A4 כל אחד.

- במבחן 10 עמודים מודפסים – בידקו שכולם בידיכם. בנוסף, בסוף הבחינה ישנו דף נוסף, ריק, לשימוש ב"מקרה חירום" בלבד.
- יש לכתוב את כל התשובות בטופס הבחינה. המחברת תשמש כטיוטה בלבד ולא תיבדק.
- יש לענות על כל השאלות.
- בכל השאלות, אלא אם נכתב במפורש אחרת:
  - אם עליכם לכתוב פונקציה, אין צורך לבדוק את תקינות הקלט שלה.
  - מותר להסתמך על סעיפים קודמים, גם אם לא עניתם עליהם.
  - ניתן לצטט טענות שנטענו בהרצאה או בשיעורי התרגול. במקרה זה יש לכתוב "בהרצאה/תרגול ראינו כי...".
- אנו ממליצים לא "להיתקע" על אף שאלה, אלא להמשיך לשאלות אחרות ולחזור לשאלה אח"כ.
- בכל סעיף ניתן לכתוב "איני יודע/ת" ולא לכתוב שום טקסט נוסף. במקרה זה יינתן 20% מציון הסעיף (מעוגל כלפי מטה).
- יש לכתוב את כל התשובות במקום המוקצב ובכתב קריא. תשובות ובהן חריגות משמעותיות מהמקום המוקצב, או תשובות הכתובות בכתב קטן מדי, לא ייקראו ולא יקבלו ניקוד, או שיקבלו ניקוד חלקי בלבד. תשובות שדורשות מאמצים רבים להבנתן גם כן עלולות לגרור הורדת ציון.

טבלת ציונים: (לשימוש הבודקים)

שאלה	ערך	ניקוד
1	25	
2	15	
3	20	
4	20	
5	20	
סה"כ	100	

בהצלחה !

מס' ת"ז_____ מס' מחברת____

שאלה 1 (25 נק')

רשימה של n מספרים שונים זה מזה תיקרא רשימה בי-טונית (bitonic list) אם קיים אינדקס $0 \leq k \leq n-1$ כך שכל האיברים עד לאינדקס k (כולל) מופיעים בסדר עולה ממש, וכל האיברים החל מאינדקס k מופיעים בסדר יורד ממש.
לדוגמא: הרשימות `[1, 2, 3, 4]` , `[-3, 1, 2, 3, 80, 100, 6]` הן רשימות בי-טוניות.

א. בסעיף זה עליכם להשלים את מימוש הפונקציה find_maximum שמקבלת כקלט רשימה בי-טונית של n מספרים שונים זה מזה, ומחזירה את האינדקס $0 \leq k \leq n-1$ שבו מתקבל הערך המקסימלי. פונקציה זו, שמומשה כבר עבורכם, הינה פונקציית מעטפת שמבצעת קריאה לפונקציה רקורסיבית בשם find_maximum_helper אותה עליכם להשלים. על סיבוכיות הזמן של הפונקציה במקרה הגרוע להיות $O(\log n)$. דוגמת הרצה:

```
>>> find_maximum([-3, 1, 2, 3, 80, 100, 6])
5
```

```
def find_maximum(blst):
    return find_maximum_helper(blst, 0, len(blst)-1)


def find_maximum_helper(blst, start, end):
    n = _____________________________________________________
    if n==1:
       return _____________________________________________
    if n==2:
       if _____________________________________________:

           return _____________________________________
       else:
           return _____________________________________


    _____________________________________________________


    if _____________________________________________:

        return _____________________________________
    else:
        return _____________________________________
```

ב. השלימו את השורות החסרות בפונקציה sort_bitonic_list(blst), שמקבלת כקלט רשימה בי-טונית blst, ומחזירה רשימה חדשה שאיבריה הם איברי blst ממויינים מקטן לגדול. השתמשו בפונקציה מסעיף א' ובפונקציה merge מהכיתה. תזכורת: merge מקבלת שתי רשימות ממוינות ויוצרת ביעילות רשימה חדשה ממוינת, המכילה את איברי שתיהן. לדוגמה:

```
>>> merge([1,3,5], [2,4])
[1, 2, 3, 4, 5]
```

דוגמת הרצה:

```
>>> sort_bitonic_list([-3, 1, 2, 3, 80, 100, 6])
[-3, 1, 2, 3, 6, 80, 100]
```

```
def sort_bitonic_list(blst):

    ____________________________________________

    return merge(_____________________ , ___________________ )
```

ג. מיכל מציעה להשתמש בפונקציה mergesort (מיון מיזוג) שראינו בכיתה, לצורך מיון רשימה בי-טונית. לדוגמה:

```
>>> mergesort([-3, 1, 2, 3, 80, 100, 6])
[-3, 1, 2, 3, 6, 80, 100]
```

לדעתה של מיכל, סיבוכיות זמן הריצה במקרה הגרוע על רשימה בי-טונית (במונחים אסימפטוטיים של O(...)) תהיה קטנה יותר מאשר זו של sort_bitonic_list מסעיף ב'. אמיר, לעומתה, חושב להיפך, כלומר שסיבוכיות mergesort במקרה הגרוע גדולה יותר. עמירם טוען שהוויכוח מיותר, כי לשתי הפונקציות אותו חסם סיבוכיות זמן ריצה עבור רשימות בי-טוניות. סמנו מי לדעתכם צודק, וציינו מה הסיבוכיות במקרה הגרוע של שתי הפונקציות הנ"ל כתלות באורך הרשימה n. תנו תשובה כחסם אסימפטוטי הדוק ככל שתוכלו.
הקיפו בעיגול:

1. מיכל צודקת
2. אמיר צודק
3. עמירם צודק

סיבוכיות sort_bitonic_list: O(_____)

סיבוכיות mergesort: O(_____)

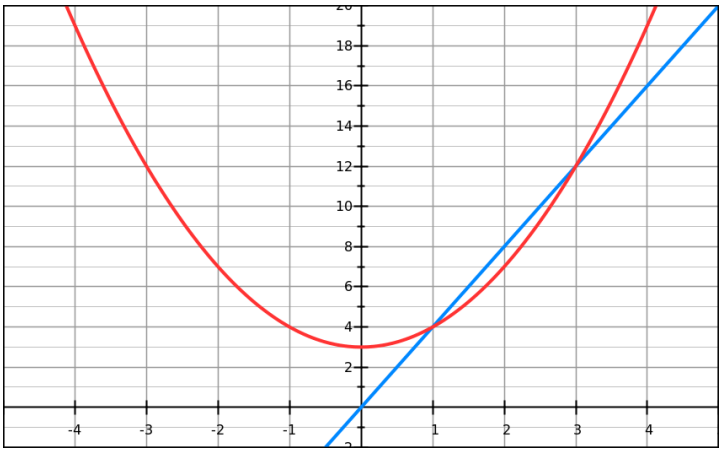
שאלה 2 (15 נק')

השאלה עוסקת בשיטת ניוטון-רפסון למציאת שורש של פונקציה (הקוד מצורף לנוחיותכם בהמשך השאלה, כמו גם הקוד של diff_param שראינו בכיתה).

א. בהינתן שתי פונקציות f1(x) ו- f2(x), שתיהן גזירות (כלומר עומדות בתנאי הנדרש עבור שיטת ניוטון רפסון), נרצה למצוא ערך x בו הפונקציות נפגשות, כלומר f1(x) == f2(x). למשל עבור:

```
f1 = lambda x:4*x
f2 = lambda x:x**2+3
```

הפונקציות נפגשות בנקודות x=1 ו- x=3 (ראו איור).



השלימו את הפונקציה equal(f1, f2) המחזירה ערך x כזה, אם קיים. הפונקציה תקרא ל- NR. כל המגבלות החלות על פעולתה של NR כמובן חלות גם על equal. למשל, יכול להיות שקיימת נקודת מפגש, אבל equal לא תמצא אותה כי NR לא החזירה תשובה כנדרש. יש להשלים שתי שורות בלבד.

```
def equal(f1, f2):
    f = lambda x:_____

    return _____
```

ב. מה יקרה כאשר יועברו ל- equal שתי פונקציות שאין להם כלל נקודת מפגש? להלן שני מקרים כאלו. עליכם להחליט עבור כל מקרה, האם:

1) equal תחזיר None ותודפס השורה מתוך NR שמתחילה ב- "zero derivative"
2) equal תחזיר None ותודפס השורה מתוך NR שמתחילה ב- "no convergence"
3) אף על פי שאין לפונקציות נקודת מפגש equal תחזיר ערך מספרי כלשהו (שכמובן אינו נכון).

```
>>> equal(lambda x:x, lambda x:x**2+1)
```
1 / 2 / 3 (הקיפו בעיגול)

```
>>> equal(lambda x:x, lambda x:x+1)
```
1 / 2 / 3 (הקיפו בעיגול)

```
def diff_param(f,h=0.001):
    return (lambda x: (f(x+h)-f(x))/h)

def NR(func, deriv, epsilon=10**(-8), n=100, x0=None):
   if x0 is None:
       x0 = uniform(-100.,100.)
   x=x0; y=func(x)
   for i in range(n):
      if abs(y)<epsilon:
          print (x,y,"convergence in",i, "iterations")
          return x
      elif abs(deriv(x))<epsilon:
         print ("zero derivative, x0=",x0," i=",i, " xi=", x)
         return None
      else:
        print(x,y)
        x=x- func(x)/deriv(x)
        y=func(x)
   print("no convergence, x0=",x0," i=",i, " xi=", x)
   return None
```

שאלה 3 (20 נק')

בשאלה זו נממש אלגוריתמים להשוואה בין מחרוזות.
לשם פשטות נניח שהמחרוזות מכילות רק אותיות קטנות (lowercase) מהא"ב האנגלי a-z .
בכל הסעיפים בשאלה אין להשתמש באופרטור ההשוואה בין מחרוזות של פייתון, אבל מותר להשוות תווים בודדים מתוך מחרוזות.
לדוגמה, מותר:

```
s1[i]==s2[i+1], s1[0]==s2[-1]
```

אסור:

```
s1==s2, s1[2:4]==s2[1:3], s1==""
```

א. השלימו את הפונקציה הרקורסיבית הבאה להשוואה בין שתי מחרוזות s1 ו s2. הפונקציה מחזירה True אם ורק אם s1==s2.
דוגמאות הרצה:

```
>>> comp('ab','ab')
True
>>> comp('','')
True
>>> comp('a','ab')
False
```

```
1 def comp(s1,s2):
2      if _____________________________________________ :
3           return True
4      if len(s1)==0 or len(s2)==0:
5           return False
6      if _____________________________________________ :
7           return False
8     return ___________________________________________
```

המשך השאלה בעמוד הבא

בשני הסעיפים הבאים נממש שתי הרחבות של הפונקציה מסעיף א'. את כל אחת מההרחבות ניתן לקבל באמצעות הוספה של שתי שורות קוד רצופות לפונקציה מסעיף א' אחרי שורה 5 ולפני שורה 6. עבור כל הרחבה השלימו את שורות הקוד החסרות.

ב. s1 יכולה להכיל את התו '+' (0 או יותר פעמים), והוא מפורש כתו מיוחד, שמתאים לכל תו יחיד ב s2.

דוגמאות הרצה של הפונקציה המורחבת:

```
>>> comp('+','a')
True
>>> comp('+','')
False
>>> comp('a+b','axb')
True
>>> comp('a+b','axxb')
False
```

הקוד להוספה עבור הרחבה ב' בלבד:

```
if _____________________________________________________ :

      return _____________________________________________
```

ג. s1 יכולה להכיל את התו '*' (0 או יותר פעמים), והוא מפורש כתו מיוחד שמתאים לכל רצף של תוים באורך 1 או יותר ב s2. בסעיף זה ההרחבה הקודמת מבוטלת, כלומר התו '+' אינו מופיע ב s1.

דוגמאות הרצה של הפונקציה המורחבת:

```
>>> comp("a*xyz","abcxyz")
True
>>> comp("a*","a")
False
```

הקוד להוספה עבור הרחבה ג' בלבד:

```
if _____________________________________________________ :

      return _____________________________________________
```

שאלה 4 (20 נק')

השאלה עוסקת בטבלאות hash. בכיתה ראינו מימוש אפשרי לטבלאות hash, כאשר הטיפול בהתנגשויות היה בשיטת השרשראות: הטבלה מיוצגת ע"י רשימה (של פייתון) שבכל כניסה שלה ישנה רשימה (של פייתון) גם כן. הרשימות ה"פנימיות" מכילות איברים שממופים לכניסה זו בטבלה הראשית.

להלן הצעה למימוש חלופי: במקום הרשימות ה"פנימיות", נשמור טבלאות hash "פנימיות". כלומר בכל כניסה של הטבלה הראשית ישנה רשימה של רשימות. בנוסף, נצטרך להגדיר שתי פונקציות hash במקום אחת: $h_1(x) = x \ mod \ m_1$ תמפה מפתחות לכניסות בטבלה הראשית שגודלה m_1, ואילו $h_2(x) = x \ mod \ m_2$ שתמפה מפתחות לכניסות בטבלאות ה"פנימיות", שגודלן m_2.

נגדיר "התנגשות חיצונית" בין שני מפתחות $x \ != y$ כמצב בו מתקיים $h_1(x) = h_1(y)$, ו"התנגשות פנימית" כמצב בו מתקיים $h_1(x) = h_1(y) \ and \ h_2(x) = h_2(y)$.

א. נניח כי הטבלה הראשית היא בגודל $m_1=5$ וכל טבלה פנימית היא בגודל $m_2=3$. מכניסים לטבלה את המפתחות 0, 5, 12, 15. האם תתרחש התנגשות פנימית בין שני מפתחות כלשהם? אם כן, ציינו מיהם המפתחות שיתנגשו ומדוע, אחרת הסבירו מדוע לא.

_____

_____

_____

ב. מהי סיבוכיות החיפוש <u>במקרה הגרוע</u> של מפתח במימוש הנ"ל של טבלת hash כתלות במספר האיברים n שנמצאים בטבלה? תנו תשובה כחסם אסימפטוטי הדוק ככל שתוכלו:

O(_____).

ג. עבור טבלה ראשית בגודל $m_1=5$ וכל טבלה פנימית בגודל $m_2=3$, מכניסים לטבלה 5 מספרים שונים זה מזה כלשהם. לאחר מכן מחפשים בטבלה את המספר 2, וידוע שהתקבל המקרה הגרוע של זמן החיפוש. רישמו דוגמה אפשרית ל- 5 המספרים שהוכנסו לטבלה טרם החיפוש:

_____

ד. יעל הציעה לבחור את גודל הטבלה הראשית להיות חזקה שלמה של גודל כל טבלה פנימית, כלומר אם גודל טבלה פנימית הוא m_2 אז גודל הטבלה הראשית יהיה $m_1 = {m_2}^k$ עבור $k \geq 2$ שלם כלשהו. מהו החיסרון העיקרי בבחירת פרמטרים כזו? הסבירו.

_____

_____

_____

מס' ת"ז_____ מס' מחברת____

שאלה 5 (20 נק')

שאלה זאת עוסקת בגנרטורים אינסופיים. בכל הסעיפים ניתן להניח שהגנרטורים הנתונים הם אינסופיים.

<u>תזכורת</u>: קריאה לפונקצית גנרטור fgen מחזירה גנרטור (סוג של איטרטור) ,gen , וכל קריאה ל (next(gen מחזירה ערך כלשהו .נאמר שפונקצית הגנרטור fgen מייצרת ערך x אם x מתקבל (מוחזר) אחרי מס׳ סופי של קריאות ל (next(gen. נאמר ש fgen מייצרת קבוצה (אולי אינסופית) של ערכים אם היא מייצרת כל ערך בקבוצה.

הפונקציה addGen מקבלת כפרמטרים שני גנרטורים אינסופיים iter1 ו iter2, ומחזירה גנרטור שהאבר ה n שלו הוא סכום האבר ה n של iter1 והאבר ה n של iter2 .

```
def addGen(iter1,iter2):
    while True:
        yield(next(iter1)+next(iter2))
```

לדוגמה:

```
def ones():
    while True:
        yield 1

def naturals():
    n = 1
    while True:
        yield n
        n += 1

>>> o = ones()
>>> [next(o) for i in range(10)]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>> n = naturals()
>>> [next(n) for i in range(10)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> o = ones()
>>> n = naturals()
>>> sumGen = addGen(o,n)
>>> [next(sumGen) for i in range(10)]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

א. רוצים להכליל את הפונקציה addGen כך שתאפשר יצירת גנרטור אינסופי מהפעלת פונקציה כלשהי על האברים של שני גנרטורים אינסופיים נתונים בהתאמה. לדוגמה:

```
>>> o = ones()
>>> n = naturals()
>>> sumGen = operGen(lambda x,y: x+y,o,n)
>>> [next(sumGen) for i in range(10)]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

כתבו את הפונקציה operGen:

```
def operGen(op,iter1,iter2):
```

ב. בסעיף הזה נכתוב פונקציה once שמקבלת כפרמטר גנרטור אינסופי iter, ומחזירה גנרטור שיוצר את כל האיברים שנוצרים על ידי iter, אבל כל איבר נוצר רק פעם אחת. לדוגמה:

```
def naturals_3():
    n = 1
    while True:
        yield n
        yield n
        yield n
        n += 1
```

```
>>> n3 = naturals_3()
>>> [next(n3) for i in range(10)]
[1, 1, 1, 2, 2, 2, 3, 3, 3, 4]
>>> n3 = naturals_3()
>>> n = once(n3)
>>> [next(n) for i in range(10)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
def once(iter):
```

ג. מה יקרה לאחר הקריאה השנייה ל next בהרצות הבאות? סמנו את התשובה הנכונה.

```
>>> o = ones()
>>> o1 = once(o)
>>> next(o1)
1
>>> next(o1)
```

1. יוחזר 1
2. יוחזר ערך מספרי שאינו 1
3. תיזרק שגיאת ריצה (exception) מסוג (StopIteration)
4. לא יוחזר שום ערך, והאינטרפרטר יציג את <<< ויחכה לפקודה הבאה
5. לא יוחזר שום ערך ולא יוצג <<< , כי האינטרפרטר יתקע

דף נוסף למקרה הצורך