מבוא למדעי המחשב מ' / ח' (234114 / 234117)

סמסטר אביב תשע"ב - מבחן מועד א', 6 יולי 2012

מספר סטודנט

- רשום/ה לקורס: **234114** / **234117**
- **תואר ראשון** / **לימודי חוץ** / **אחר (לפרט)** : _____

- משך הבחינה – **3 שעות.**
- בדקו שיש 13 עמודים (4 שאלות) במבחן, כולל עמוד זה.
- השימוש בחומר עזר כלשהו, כתוב או אלקטרוני, **אסור**.
- המבחן כתוב בלשון נקבה אך מתיחס לנבחנים ולנבחנות כאחד.
- ניתן להשתמש בפונקציות קלט-פלט סטנדרטיות והקצאת זיכרון ב- C. שימוש בכל פונקציה אחרת, לרבות כזו שהוגדרה במהלך הקורס, אסור. אתן יכולות להגדיר פונקציות עזר כרצונכן. אין צורך להצהיר עליהן.
- כל זיכרון שאתן מקצות, אתן חייבות בשחרורו.
- אין צורך בהוכחות לחישובי הסיבוכיות.
- ניתן לכתוב בעיפרון ולהשתמש במחק.

שאלה	ערך	ציון	בודק
1	25		
2	25		
3	25		
4	25		
סה"כ	100		

<u>צוות הקורס</u> :
<u>סמסטר אביב תשע"א</u> :
מרצים: ד"ר ניר אילון (מרצה אחראי), דן רביב, תמיר לוי
מתרגלים: חביאר טורק (מתרגל אחראי), אבישי גרץ, תהילה מייזלס, רן זילברשטיין, רועי פורן

בהצלחה !

שאלה 1 (25 נקודות):

א. נתונה הגדרה של פונקציה f. נתחו את את סיבוכיות הזמן והמקום הנוסף שלה כפונקציה של n כאשר נתון ש-n חיובי. אין צורך לפרט את שיקוליכם.

```
void f(int n) {
   int x = n;

   while (x * x > n) {
       x /= 2;
       printf ("x cubed = %d\n", x * x * x);
   }


   while (x > 0)
      x--;

   printf("hello %d\n", x);
}
```

סיבוכיות זמן: $\Theta(\sqrt{n})$ סיבוכיות מקום נוסף: $\Theta(1)$

ב. נתונה הפונקציה הבאה שמבצעת חיפוש בינארי באופן רקורסיבי. על הפונקציה להחזיר את האינדקס במערך הנתון a (באורך n) שבו מופיע הערך x, ולהחזיר -1 אם לא קיים מקום כזה. עליכן להשלים את החלק החסר (ברווחים המודגשים באפור). מתחת לקוד יש למלא את סיבוכיות הזמן והמקום הנוסף כפונקציה של n (בלי לנמק).

1. אם הערך x מופיע יותר מפעם אחת, מותר להחזיר מיקום שרירותי של x.
2. הנחת הקלט היא שהמערך a ממויין בסדר לא יורד (אין צורך לבדוק).

רמז: אופרטור טרינרי יכול לעזור.

```
int recursive_binary_search(int* a, int n, int x) {
  int mid;
  int res;
  if (n == 0)
    return      -1          ;

  mid = n/2;
  if (x == a[mid])
    return      mid         ;

  if (x > a[mid]) {
    res = recursive_binary_search(a+mid+1, n-mid-1, x);
    return   (res>-1) ? res+mid+1 : -1                ;
  } else {
    res = recursive_binary_search(a, mid  , x);
    return   res                                      ;
  }
}
```

סיבוכיות זמן: $\Theta(\log n)$ סיבוכיות מקום נוסף: $\Theta(\log n)$

שאלה 2 (25 נקודות) :

עליכן לכתוב פונקצייה שחתימתה:

```
int compress(int a[], int n);
```

הפונקציה מקבלת כקלט מערך של מספרים **חיוביים** (הפרמטר a) וכן את אורכו (הפרמטר n) ו"דוחסת" את תוכן המערך, לפי הכלל הבא. אם מספר x מופיע ברצף k פעמים במערך ו- k **לפחות** 3, אז יש להחליף את הרצף בזוג המספרים k – ואחריו x. על הפונקציה להחזיר את אורך המערך לאחר הדחיסה.

לדוגמא, אם הקלט הוא המערך a הבא (באורך 7, משמאל לימין):

3	4	4	5	3	6	7

אז הוא לא יעבור שום שינוי, מאחר שאין רצף של מספר המופיע יותר מפעמיים. הערך שיוחזר הוא 7.

לעומת זאת, הקלט הבא:

5	5	6	3	7	8	8	8	9

יידחס בפלט למערך הבא:

5	5	6	3	7	-3	8	9

והערך שיוחזר הוא 8.

דגשים:

1. הפלט נכתב באותו מערך a שבו מגיע הקלט.
2. אין צורך להתיחס למקום העודף במערך. לדוגמא, אם הקלט באורך 8 והפלט באורך 5, אז תוכן המקומות 5, 6 ו- 7 במערך הפלט לא משנה.
3. אין להקצות זיכרון באופן דינמי. פיתרון שיקצה זיכרון לא יזכה בניקוד מלא.

(המשך בעמוד הבא)

```
int compress(int a[], int n) {
{
    int i, j, current=0, count=1;
    for (i=1; i < n+1; i++)
    {
        if(i<n && a[i]==a[i-1])
        {
            count++;
        }
        else
        {
            if(count >= 3)
            {
               a[current] = -count;
               current++;
               count = 1; /* צריך להעתיק את המספר המקורי פעם אחת*/
            }

            for (j=0; j<count; j++)
                a[current+j] = a[i-1];

            current += count;
            count = 1;
        }
    }

    return current;
}
```

שאלה 3 (25 נקודות) :

שאלה זו עוסקת בפולינומים. פולינום הוא פונקציה מתמטית מהצורה הבאה:

$f(x) = a_0 + a_1x + \ldots + a_dx^d$

כפי שרואים, פולינום נקבע ע"י מספר טבעי $d \geq 0$ הנקרא *דרגה (degree)*, ופרמטרים ממשיים $a_0..a_d$ הנקראים *מקדמים (coefficients)* .

פולינומים נייצג באמצעות מבנה (struct) בצורה הבאה:

```
struct polynomial {
  int d;
  double* a;
}
typedef struct polynomial POLY;
```

דוגמא:
כדי לייצג את הפולינום $1+2x+3x^2$ יש לאתחל את d ל- 2, ואת a למערך באורך לפחות 3 המאותחל כך ש a[0]==1, a[1]==2, a[2]==3.

כתבו פונקצייה שמקבלת כקלט מצביע לפולינום p ומשתנה ממשי x, ומחזירה את הערך של הפולינום בנקודה x. על הפונקציה לעבוד ב**סיבוכיות זמן O(d)**. סיבוכיות המקום הנוסף הדרושה היא **O(1)**. חתימתה:

```
double evaluate(POLY* p, double x);
```

לדוגמא: אם הפולינום הוא $1+2x+3x^2$ ו- x=4 אז יש להחזיר 57.

הערות: בשאלה זו אינכן נדרשות לכתוב פונקציות אתחול או שחרור של פולינום. ניתן להניח שפולינום הקלט כבר מאותחל כראוי, כלומר ש- $d \geq 0$ וששדה המקדמים מצביע למערך באורך לפחות d+1.

אם כתבתן פתרון בסיבוכיות זמן שהיא **לא O(d)**, תוכלו לקבל בחזרה חלק מהנקודות אם תכתבו כאן את הסיבוכיות של הפיתרון שלכן: _____

```
double evaluate(POLY *p, double x)  {
    int i;
    double pow = 1.0, res = p->a[0];

    for(i = 1; i <= p->d; i++)  {
        pow *= x;
        res += p->a[i]*pow;
    }

    return res;
}
```

שאלה 4 (25 נקודות) :

בבנק הישראלי הידוע capital-pig, בכל פעם שלקוח מבצע משיכה ונכנס למשיכת יתר (מינוס), הבנק גובה 5 ש״ח עמלה, **גם אם הלקוח כבר היה במשיכת יתר לפני הפעולה**. לדוגמא:

נניח שהמאזן של הלקוח הוא 30 בזכות, והוא מבצע משיכה בסכום של 40 ואח״כ עוד משיכה בסכום של 7. לאחר הפעולה הראשונה תהייה היתרה 15- (40 ירדו בגלל המשיכה עצמה, ועוד 5 עמלת משיכת היתר). לאחר הפעולה השנייה תהייה היתרה 27- (7 ירדו בגלל המשיכה עצמה, ועוד 5 עמלת משיכת היתר).

לעומת זאת, אם הלקוח היה הופך את סדר הפעולות, כלומר קודם מבצע משיכה של 7 ורק אח״כ משיכה של 40, אז היתרה שלו בסוף היתה 22- מכיוון שהפעולה הראשונה לא הכניסה אותו למשיכת יתר.

שימו לב: אין עמלות על הפקדות.

הבנק החליט לשנות את סדר הפעולות של הלקוח כדי למקסם את רווחיו (של הבנק). כדי שהלקוח לא ישים לב, אסור שאף פעולה תזוז ביותר מ- 3 מקומות. לדוגמא, בטבלה הבאה סידור א׳ הוא **חוקי** בעוד שסידור ב׳ **לא חוקי** מכיוון שהפעולה 14- עברה ממקום 6 למקום 2, כלומר זזה 4 מקומות.

i	0	1	2	3	4	5	6
transactions[i]	10	-15	3	4	20	30	-14
סידור א'	10	-15	3	4	-14	20	30
סידור ב'	10	-15	-14	3	4	20	30

עליכן לכתוב פונקציה שמקבלת מערך פעולות transactions באורך n ויתרה התחלתית x. (פעולה חיובית מציינת הפקדה, ופעולה שלילית מציינת משיכה.) על הפונקציה להדפיס את מקסימום העמלה שהבנק יכול לגבות מסידור חוקי של הפעולות. חתימת הפונקציה:

```
void max_capital_pig(int transactions[], int n, int x);
```

לדוגמא, אם n=7 ומערך ה- transactions הוא כמו בטבלה ו- x=0, אז יש להדפיס

```
Maximal fees: 10
```

(פתרון זה מתקבל מסידור א׳. אין זה הסידור היחיד שמוביל לעמלה כזו).

יש להשתמש בטכניקת ה- backtracking שנלמדה בשיעור, ויש לחתור לקיצוץ מירבי של מרחב אפשרויות החיפוש. אין להשתמש במשתנים סטטיים או גלובליים. מותר להגדיר פונקציות עזר.

```
void max_capital_pig(int transactions[], int n, int x)
{
   int *permutation = (int*)malloc(n*sizeof(int));
   int *used = (int*)malloc(n*sizeof(int));
   int i, max;

   for (i = 0; i < n; i++) {
      permutation[i] = 0;
      used[i] = 0;
   }

   max = max_capital_pig_aux(transactions,n,x,permutation,used,0);
   printf("Maximum fee: %d\n", max);

   free(used);
   free(permutation);
}

int max_capital_pig_aux(int transactions[], int n, int x,
                        int *permutation, int *used, int index)
{
   int i, temp_max = 0, max = 0;

   if (index == n) {
     return calc_fee(permutation,n,x);
   }

   for (i = index-3; i <= index+3; i++) {
     if (i < 0 || i>= n || used[i] == 1) {
        continue;
     }

     permutation[index] = transactions[i];
     used[i] = 1;
     temp_max = max_capital_pig_aux(transactions,n,x, permutation,
                                    used, index+1);
     if (temp_max > max) {
        max = temp_max;
     }
     permutation[index] = 0;
     used[i] = 0;
  }

  return max;
}
```

```
int calc_fee(int permutation[],int n,int x) {
    int i = 0, fee = 0;
    for (i = 0; i < n; i++) {
        x += permutation[i];
        if (x < 0 && permutation[i] < 0) {
            fee += 5;
            x -= 5;
        }
    }
    return fee;
}
```