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

סמסטר חורף תשס"ח

פתרון מבחן מסכם מועד א', 7 פברואר 2008

מספר סטודנט	שם משפחה	שם פרטי

משך המבחן: 3 שעות.
חומר עזר: אין להשתמש בכל חומר עזר בכתב, מודפס או אלקטרוני.

הנחיות והוראות:

- מלאו את הפרטים בראש דף זה.
- בדקו שיש 24 עמודים (4 שאלות) במבחן, כולל עמוד זה.
- כתבו את התשובות על טופס המבחן בלבד, במקומות המיועדים לכך. שימו לב שהמקום המיועד לתשובה אינו מעיד בהכרח על אורך התשובה הנכונה.
- העמודים הזוגיים בבחינה ריקים. ניתן להשתמש בהם כדפי טיוטה וכן לכתיבת תשובותיכם. סמנו טיוטות באופן ברור על מנת שהן לא תיבדקנה.
- יש לכתוב באופן ברור, נקי ומסודר.
- אין לכתוב הערות והסברים לתשובות אם לא נתבקשתם מפורשות לכך.
- בכל השאלות, הינכם רשאים להגדיר ולממש פונקציות עזר כרצונכם.
- **אין להשתמש בפונקציות ספריה או בפונקציות שמומשו בכיתה אלא אם צוין אחרת בשאלה.**
- **פתרון שלא עומד בדרישות הסיבוכיות יקבל ניקוד חלקי בלבד.**

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

צוות הקורסים 234114/7
מרצים: פרופ'/ח' מיכאל אלעד (מרצה אחראי), סאהר אסמיר, ד"ר צחי קרני, רן רובינשטיין.
מתרגלים: אלדר אהרוני, גדי אלקסנדרוביץ', רון בגלייטר, שגיא בן-משה, אורי זבולון, מרק זילברשטיין, סשה סקולוזוב, ולנטין קרבצוב, אייל רגב, אייל רוזנברג, אנדרי קלינגר (מתרגל אחראי).

בהצלחה!

שאלה 1 (25 נקודות)

סעיף א

בכל אחד מהסעיפים הבאים מופיעות מספר שורות קוד. לכל קטע קוד, הקיפו בעיגול את התיאור המתאים והסבירו את בחירתכם בקצרה:

א. **ללא שגיאות** – הקוד יתקמפל ללא כל שגיאה וירוץ ללא תקלות.
ב. **שגיאת זמן ריצה** – הקוד יתקמפל ללא שגיאות, אולם עלול לגרום לשגיאה בזמן ריצתו (כלומר הפסקה מוקדמת של התוכנית ללא הגעה לסוף הפונקציה main).
ג. **שגיאת קומפילציה** – הקוד לא יעבור קומפילציה.

1.

```
double a[8] = {0,1,2,3,4,5};
double *p = a + 5;
a = p;
```

א. ללא שגיאות
ב. שגיאת זמן ריצה
ג. שגיאת קומפילציה

הסבר:
a הינו מערך. לא ניתן לשנות את כתובתו.

2.

```
void f(int i) {
  i = i - 1;
}
int r(int i) {
  if(i == 0) return 1;
  f(i);
  return r(i);
}
int main() {
  r(10);
  return 0;
}
```

א. ללא שגיאות
ב. שגיאת זמן ריצה
ג. שגיאת קומפילציה

הסבר:
ערכו של i לא משתנה בתוך הפונקציה r ולכן תהיה רקורסיה אינסופית שתגרום לקריסת המחסנית.
לולאה אינסופית רגילה (לא רקורסיבית) לא הייתה נחשבת לשגיאה.

3.

```
float *p, x;
p = &x;
x = *p = 4.5;
```

א. ללא שגיאות
ב. שגיאת זמן ריצה
ג. שגיאת קומפילציה

הסבר:
הקוד תקין. ההשמות בשורה שלישית מתבצעות מימין לשמאל.

סעיף ב

נתונה הפונקציה הבאה:

```
#define N 4

int foo(int arr[N][N], int q, int w)
{
  int s = q*w;
  int i,j,k,l;

  if((N % q != 0) || (N % w != 0)) {
    return 0;
  }

  s = (s+1)*s/2;

  for (i=0; i<=N-q; i+=q) {
    for (j=0; j<=N-w; j+=w) {

      int count = 0;
      for (k=0; k<q; k++) {
        for (l=0; l<w; l++) {
          count += arr[i+k][j+l];
        }
      }

      if (count != s) {
        return 0;
      }
    }
  }

  return 1;
}
```

1. מהי סיבוכיות הפונקציה? ניתן לבטא באמצעות הערכים w, q, N:

סיבוכיות זמן:	Θ(N^2)	סיבוכיות מקום נוסף:	Θ(1)

הסבר:
הפונקציה מחלקת את המטריצה הדו-מימדית arr למלבנים בגודל q*w.
אם זו מתחלקת במדייק וסכום האיברים בכל מלבן הוא q*w*(q*w+1)/2 (כלומר כסכום הסדרה החשבונית מ 1 עד q*w) אז הפונקציה תחזיר 1, אחרת היא תחזיר 0.
סיבוכיות הזמן היא $\Theta(N^2)$ כי הפונקציה סורקת את כל המערך.
סיבוכיות מקום נוסף היא $\Theta(1)$ כי אין רקורסיה ואין הקצאות מערכים.

נתון המערך:

```
int arr[4][4] = {{1,2,3,4},
                 {3,4,1,2},
                 {2,3,4,1},
                 {4,1,2,3}};
```

2. מה יהיה ערכו של **c** בתום הקריאה הבאה?

```
int c = foo(arr,2,2);
```

ערכו של **c** יהיה: **1**

3. מה יהיה ערכו של **c** בתום הקריאה הבאה?

```
c = foo(arr,1,4) + foo(arr,4,1) + foo(arr,1,2);
```

ערכו של **c** יהיה: **2**

<u>הסבר:</u>
אם מחלקים את המטריצה לריבועים בגודל 2*2 אז סכום האיברים בכל ריבוע הוא 10, כמו סכום סדרה חשבונית באורך 4. גם אם מחלקים את המטריצה למלבנים 4*1 או 1*4 , סכום האיברים בכל מלבן הוא 10. כאשר מחלקים את המטריצה למלבנים 2*1 אזי יש מלבנים שסכום איבריהם אינו 3.

שאלה 2 (25 נקודות)

נתונים שני מערכים: מערך a שאורכו na ומערך b שאורכו nb. איברי שני המערכים הם מטיפוס int. **אנו נניח ש- na≤nb** (כלומר המערך b ארוך יותר) ושכל מערך מכיל איברים שונים זה מזה ללא חזרות.

בשאלה זו (בשני הסעיפים) מותר להשתמש בכל פונקציה או אלגוריתם שנלמדו בכיתה. במידה ובחרתם להשתמש בפונקציות כאלה, ציינו בטבלה הבאה את החתימות שלהן, ואת סיבוכיות הזמן והמקום שלהן.

חתימה	סיבוכיות זמן	סיבוכיות מקום נוסף
void merge_sort(int a[], int n);	Θ(nlogn)	Θ(n)
int binary_search(int a[], int n, int x); **הערה: הגרסה שמחזירה 1 אם האיבר נמצא ו 0 אחרת.**	Θ(logn)	Θ(1)

סעיף א

בסעיף זה נניח כי המערכים **ממוינים**. כתבו פונקציה שמקבלת את שני המערכים, ומחזירה את מספר האיברים שמופיעים רק באחד מן המערכים אך לא בשניהם.

לדוגמה, עבור המערכים הבאים על הפונקציה להחזיר 4, כיוון שיש ארבעה מספרים בדיוק המופיעים רק באחד משני המערכים אך לא בשניהם (המספרים מסומנים בקו):

```
int a[]={1,2,3,5};
int b[]={0,2,3,5,6,7};
```

על הפונקציה לעמוד **בסיבוכיות זמן טובה ככל האפשר**, ובסיבוכיות מקום נוסף O(1). פונקציה לא אופטימלית תזכה בניקוד חלקי בלבד.

שימו לב:

- פתרונות שמגדירים מערכים באורך MAX_INT (או אורך דומה) לא יתקבלו.
- במידת הצורך, ניתן בפונקציה לשנות את תוכן המערכים.

השלימו את סיבוכיות הזמן של הפונקציה שלכם:

סיבוכיות זמן: Θ(_____)

יש 2 פתרונות:

```
int diff(int a[], int b[], int na, int nb) {
   int i=0,j=0,count=0;
   while (i<na && j<nb)
   {
      if (a[i]==b[j])
      {
         i++;
         j++;
      }
      else if (a[i]<b[j])
      {
         i++;
         count++;
      }
      else
      {
         j++;
         count++;
      }
   }
   return count+(na-i)+(nb-j);
}
```

הסבר:
דומה לפעולת merge. עוברים על שני המערכים בו זמנית. אם האיברים זהים – מדלגים עליהם, אחרת סופרים את הקטן וממשיכים.

סיבוכיות זמן: Θ(nb)

```
int diff(int a[], int b[], int na, int nb) {
   int i, count=na+nb;
   for (i=0; i<na; i++)
   {
      if (binary_search(b,nb,a[i]))
         count-=2;
   }
   return count;
}
```

הסבר:
יש סה"כ na+nb איברים בשני המערכים יחד.
עוברים בלולאה על המערך a ומחפשים כל איבר שלו במערך b.

סיבוכיות זמן: Θ(na*log(nb))

הערה:
לא ניתן לקבוע לאיזו משני הפתרונות סיבוכיות טובה יותר, ולכן שני הפתרונות נכונים.

סעיף ב

בסעיף זה **אין להניח** כי המערכים ממוינים. עליכם לממש פונקציה שמקבלת את שני המערכים, ומחזירה את מספר האיברים שמופיעים בשני המערכים בו-זמנית (כלומר, את גודל קבוצת החיתוך). לדוגמה, עבור המערכים הבאים הפונקציה תחזיר את הערך 2 (המספרים שמסומנים בקו משותפים לשני המערכים):

```
int a[]={1,9,5,4};
int b[]={5,10,18,1,3};
```

על הפונקציה להיות בעלת **סיבוכיות זמן טובה ככל האפשר**. בנוסף, במידת האפשר יש לחסוך גם בזיכרון, אך בכל מקרה לא על חשבון סיבוכיות הזמן. פתרונות לא אופטימאליים יזכו בניקוד חלקי בלבד.

שימו לב:

- כיוון ש- nb≥na, אנו נחשיב את θ(na) בתור **טוב יותר** מ- θ(nb) לצרכי השוואת יעילות.
- פתרונות המגדירים מערכים באורך MAX_INT (או אורך דומה) לא יתקבלו.
- במידת הצורך, ניתן בפונקציה לשנות את תוכן המערכים.

השלימו את סיבוכיות הפונקציה שכתבתם:

סיבוכיות זמן: (**nb*log(na)**)Θ סיבוכיות מקום נוסף: (**na**)Θ

```
int intersect(int a[], int b[], int na, int nb) {
   int i, count=0;
   merge_sort(a, na);
   for (i=0; i<nb; i++)
   {
      if (binary_search(a, na, b[i]))
         count++;
   }
   return count;
}
```

הסבר:

ממיינים את המערך a ומחפשים כל איבר של המערך b בתוכו.

שאלה 3 (25 נקודות)

נתון מערך a באורך n, המכיל מספרים שלמים אי שליליים. עליכם לממש פונקציה שממיינת את המערך באופן הבא:

- בתחילת המערך יופיעו כל המספרים שמתחלקים ב- 3 ללא שארית.
- לאחריהם יופיעו כל המספרים שמתחלקים ב- 3 עם שארית 1.
- בסוף המערך יופיעו כל יתר המספרים (אלו שמתחלקים ב- 3 עם שארית 2).

אין חשיבות לסדר הפנימי של המספרים בתוך כל קבוצה.

על הפונקציה לעבוד בסיבוכיות זמן O(n) וסיבוכיות מקום נוסף O(1).

ניתן בשאלה זו להשתמש בפונקציה הבאה, המקבלת מצביעים לשני מספרים שלמים ומחליפה את תוכנם:

```
void swap(int *p, int *q);
```

```
void sortby3(int a[], int n) {
   int i, p=0;
   for (i=0; i<n; i++){
      if((a[i]%3) == 0){
          swap(&a[i], &a[p]);
          p++;
      }
   }

   for(i=p; i<n; i++){
      if((a[i]%3) == 1){
          swap(&a[i], &a[p]);
          p++;
      }
   }
}
```

הסבר:
בלולאה הראשונה מזיזים את כל האיברים שמתחלקים ב 3 ללא שארית לתחילת המערך.
בלולאה השנייה מזיזים את כל האיברים שמתחלקים ב 3 עם שארית 1 לאמצע המערך החל מאותו מקום בו סיימנו למקם את האיברים שמתחלקים ב 3 ללא שארית.

שאלה 4 (25 נקודות)

בשאלה זו נרצה לפתור את בעיית **מגדלי הוואי**. בדומה לבעיית מגדלי הנוי, בבעיה יש שלושה מגדלים (נסמנם 1, 2 ו-3), ומספר טבעות המונחות על אחד המגדלים. ואולם כיאה להוואי, ובניגוד לבעיית מגדלי הנוי, הטבעות הן צבעוניות: מכל גודל טבעת יש שתי טבעות, האחת אדומה והשנייה לבנה, ובסה"כ יש m טבעות אדומות ו-m טבעות לבנות.

במצב ההתחלתי הטבעות ממוקמות זו מעל זו במגדל מס' 1 ומסודרות לפי גודלן (הגדולה ביותר למטה). מכל זוג טבעות באותו הגודל, זו הלבנה ממוקמת מתחת לזו האדומה. למשל, עבור m=3 המצב ההתחלתי הוא:

כללי הבעיה נותרים כמו בבעיה המקורית:

- אין להניח טבעת גדולה מעל טבעת קטנה ממנה. עם זאת, **מותר** להניח שתי טבעות מאותו הגודל האחת מעל לשנייה, ללא קשר לצבע (כלומר מותר לבנה על אדומה וגם אדומה על לבנה מאותו הגודל).
- אין להזיז יותר מטבעת אחת בו זמנית.

בשאלה זו אין הגבלה על סיבוכיות הפונקציות. לשם הזזת הטבעות, ניתן להשתמש בפונקציות הבאות:

```
void movered(int from, int to) {
  printf("move red ring from %d to %d\n", from, to);
}
```

```
void movewhite(int from, int to) {
  printf("move white ring from %d to %d\n", from, to);
}
```

סעיף א

כתבו פונקציה שמקבלת את m – מספר זוגות הטבעות במגדל המקור (כלומר סה"כ ישנן 2m טבעות, שתיים בכל גודל), וכן את מספר מגדל המקור ומספר מגדל היעד. הפונקציה מדפיסה את ההוראות להעברת כל הטבעות ממגדל המקור למגדל היעד, תוך שימוש במגדל השלישי כמגדל עזר. במצב הסופי, על הטבעות להיות **באותו הסדר** כמו שהיו במגדל המקור – כלומר לבנה מתחת לאדומה. לדוגמה, עבור m=1, מגדל המקור 1 ומגדל היעד 3, הפונקציה תדפיס:

```
move red ring from 1 to 2
move white ring from 1 to 3
move red ring from 2 to 3
```

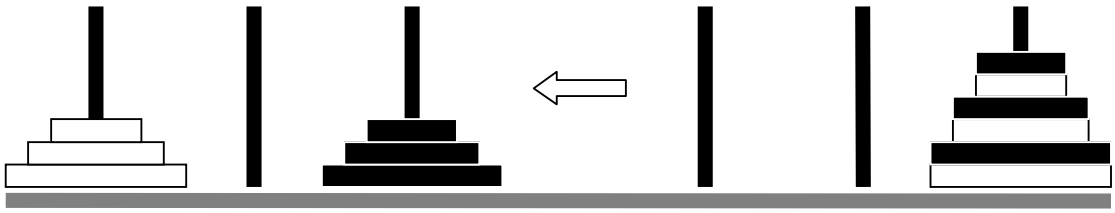
```
void hawaii1(int m, int from, int to) {
   int via=1+2+3-from-to;
   if (m<=0) return;
   hawaii1(m-1,from,to);
   move_red(from,via);
   move_white(from,via);
   hawaii1(m-1,to,from);
   move_white(via,to);
   move_red(via,to);
   hawaii1(m-1,from,to);
}
```

הסבר:
יש מספר פתרונות אפשריים, ואנו נציג כאן שניים. הפתרון הראשון פועל על צמדי טבעות במשותף ולא מפריד אותן במהלך פעולתו, והפתרון השני מאפשר פיצול כזה. הפתרון הראשון יעיל יותר בכמות ההעברות בשל שימוש ב-3 קריאות ל-hawaii1 בגודל m-1 בעוד שהשני מחייב 4 קריאות כאלה. למרות זאת, שני פתרונות אלו קבילים ונחשבים למענה מלא לשאלה.

```
void hawaii1(int m, int from, int to) {
   int via=1+2+3-from-to;
   if (m<=0) return;
   hawaii1(m-1,from,to);
   move_red(from,via);
   hawaii1(m-1,to,via);
   move_white(from,to);
   hawaii1(m-1,via,from);
   move_red(via,to);
   hawaii1(m-1,from,to);
}
```

סעיף ב

כתבו פונקציה שמקבלת את m – מספר זוגות הטבעות במגדל המקור, וכן את מספר מגדל המקור ומספר מגדל היעד. הפונקציה מדפיסה הוראות להעברת הטבעות, כך שבסוף התהליך הטבעות מופרדות על פי צבען – כל הלבנות נמצאות במגדל היעד (הגדולה למטה) וכל האדומות נמצאות במגדל המקור (הגדולה למטה). המצב ההתחלתי והמצב הסופי נראים כך:



לדוגמה, עבור m=1, מגדל המקור 1 ומגדל היעד 3, הפונקציה תדפיס:

```
move red ring from 1 to 2
move white ring from 1 to 3
move red ring from 2 to 1
```

בסעיף זה ניתן להשתמש בפונקציה מהסעיף הקודם, גם אם לא מימשתם אותה.

```
void hawaii2(int m, int from, int to) {
   int via=1+2+3-from-to;
   if (m<=0) return;
   hawaii1(m-1,from,to);
   move_red(from,via);
   hawaii1(m-1,to,via);
   move_white(from,to);
   hawaii1(m-1,via,to);
   move_red(via,from);
   hawaii1(m-1,to,from);
   hawaii2(m-1,from,to);
}
```

הסבר:
גם כאן קיימים מספר פתרונות אפשריים, ואנו נתמקד באחד מהם שנראה הכי טבעי. בפתרון זה אנו מעבירים m-1 צמדים לעמודת to, מעבירים את האדום ל-via ומעליו שמים את כל הטבעות הקטנות שהיו ב-to. בשלב זה הטבעת הלבנה הגדולה ביותר נמצאת ב-from ויכולה לעבור ל- to- יעדה הסופי. לאחר מכן, m-1 צמדי הטבעות הקטנות מועברות ל- to על מנת לאפשר לטבעת האדומה הגדולה לעבור ליעדה הסופי (from). בסיום יש להחזיר את כל הטבעות הקטנות למקומן המקורי ב-from. אסור לשכוח לקרוא בשלב זה ל-hawaii2 על מנת להמשיך תהליך רקורסיבי שיפעל על m-1 הצמדים הקטנים.

פתרון אחר מתבסס על העברת כל הטבעות (ע"י hawaii1) ל-to, כך שהטבעת הלבנה הגדולה נמצאת במקומה. לאחר מכן יש להעביר m-1 צמדים ל-via, כך שהטבעת האדומה הגדולה תוכל להתמקם ב-from – מיקומה הסופי. שוב נדרש להעביר את m-1 הצמדים הקטנים ל-from, והתהליך ימשיך באופן רקורסיבי עליהם.

משפחת פתרונות שלישית שאותה נזכיר בקצרה מציעה מבנה בו hawaii2 אינו מבוסס רקורסיה, אלא על לולאת for או while. בפתרון זה מאתחלים את התהליך בהעברת כל הטבעות ל-via. לאחר מכן, פועלים בלולאה מ-i=0 ועד m-1, וכנגד כל ערך של i נעביר m-i צמדים ל-to (אז הטבעת הלבנה שבתחתית נמצאת במקומה הרצוי), m-i+1 מאלה ל-via, ואז את האדומה שב-to נעביר ל- from – מקומה הסופי. בשלב זה בעמודת via ממתינים m-i+1 צמדי טבעות לטיפול דומה, ולכן הלולאה.