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

סמסטר חורף תשס"ח

פתרון מבחן מסכם מועד א'-חדש, 17 אפריל 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 22 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, אנדרי קלינגר (מתרגל אחראי), ולנטין קרבצוב, אייל רגב, אייל רוזנברג.

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.
```
char s[] = "Moed";
s[4] = "A";
```

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

הסבר:הטיפוסים לא מתאימים – "A" הינו מטיפוס *char ואילו s[4] מטיפוס char.

2.
```
int a[10] = {10};
a[sizeof(a)-1] = 3;
```

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

הסבר: sizeof(a) יחזיר את גודל המערך a <u>בבתים</u> שזה יותר מ 10, ולכן תהיה כתיבה מחוץ לגבולות המערך.

3.
```
char c;
scanf("%d", &c);
c++;
```

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

הסבר: scanf תכתוב נתון בגודל של int (בלי קשר למה שהמשתמש הקליד) ולכן תהיה דריסת זכרון כי גודלו של c הינו בית בודד.

4.
```
int x=0, y=5;
int b = (0<=y<=3)?1:1/x;
```

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

הסבר: התנאי תמיד מתקיים (ללא תלות בערכו של y) ולכן 1/x לא יתבצע.

5.
```
int a;
int* b = &a;
void* c = b;
*c = 3;
```

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

הסבר: אי אפשר להפעיל אופרטור * על מצביע מטיפוס *void.

סעיף ב

נתון קוד הבא:

```
#include <stdio.h>

int foo(int n)
{
     if (n <= 0)
         return 1;
     else
         return zzz(n-1);
}

int zzz(int n)
{
     if (n <= 0)
         return 0;
     else
         return foo(n-1);
}

int main()
{
     int n;
     if ( scanf("%d", &n) <1 )
         printf("input error");
      else
         printf("the result is %d", foo(n));
      return 0;
}
```

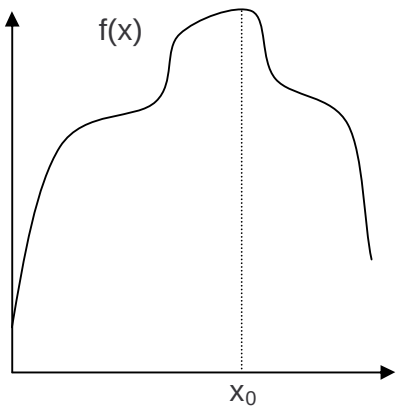
מה סיבוכיות הזמן והמקום של התוכנית כפונקציה של n?

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

הסבירו במשפט אחד מה התוכנית מחשבת (מה משמעות התוצאה שהיא מדפיסה?)

התוכנית מדפיסה 1 אם המשתמש הכניס מספר שלילי או מספר זוגי ו 0 אם הוא הכניס מספר חיובי ואי-זוגי.

שאלה 2 (25 נקודות)



בשאלה זאת נרצה למצוא נקודת מקסימום של פונקציה (מתמטית) יונימודלית f(x).
פונקציה יונימודלית הינה פונקציה בעלת מקסימום יחיד (בנקודה x_0 אותה אנחנו רוצים למצוא) ואשר הנגזרת שלה חיובית ממש לכל $x< x_0$ ושלילית ממש לכל $x> x_0$.
בנקודה $x= x_0$ הנגזרת שווה לאפס.

בשאלה זאת אנו מניחים ש:

- הפונקציה והנגזרת שלה מוגדרים עבור ערכי x **שלמים בלבד, בין 0 ל n-1 כולל**.
- x_0 – נקודת המקסימום, הינה מספר **שלם**.

סעיף א

בסעיף זה לצורך מציאת המקסימום של הפונקציה (המתמטית) f אתם יכולים להשתמש בפונקציות (של שפת C) הבאות:

```
double f(int x);     // מחזירה את ערך הפונקציה בנקודה
double df(int x);    // מחזירה את נגזרת הפונקציה בנקודה
```

עליכם לממש את הפונקציה (של שפת C) `find_maxA()` שתחזיר את הנקודה x_0 בה הפונקציה f מקבלת מקסימום. הפונקציה מופיעה בדף הבא.

דרישות סיבוכיות: עליכם למזער את מספר הקריאות לפונקציות f ו-df הנ"ל (בשאלה זו לא נבדיל בין קריאה ל-f וקריאה ל-df לצרכי סיבוכיות). כמו כן השלימו את סיבוכיות מספר הקריאות ל-f ו-df (גם יחד) במקום המתאים למטה. פתרון בעל מספר קריאות לא אופטימאלי יזכה לנקוד חלקי בלבד.

סכום מספר הקריאות ל f ו df יחד: $\Theta(\log(n))$

```
int find_maxA(int n) {
  int low=0, high=n-1;
  while (low<=high) {
    int mid = (low+high)/2;
    double d_f = df(mid);
    if (d_f==0)
      return mid;
    else if (d_f<0)
      high = mid-1;
    else
      low = mid+1;
  }
  return low;
}
```

עושים חיפוש בינארי על הנגזרת. אם היא חיובית אז אנחנו משמאל למקסימום ואם היא שלילית אז מימין.

סעיף ב

בסעיף זה ניתן להשתמש רק בפונקציה f() ולא בפונקציה df(). עליכם לממש את הפונקציה find_maxB() להלן, שמחזירה את נקודת המקסימום של f אך ללא השימוש בפונקציה df().

דרישות סיבוכיות: בדומה לסעיף א', עליכם למזער את מספר הקריאות לפונקציה f. השלימו את סיבוכיות מספר הקריאות ל-f במקום המתאים למטה. פתרון בעל מספר קריאות לא אופטימאלי יזכה לנקוד חלקי בלבד.

מספר קריאות ל f: $\Theta(\log(n))$

```
int find_maxB(int n) {
  int low=1, high=n-1;
  if (n==1 || f(1)<f(0)) return 0;
  if (f(n-1)>f(n-2)) return n-1;
  while (low<=high) {
    int mid = (low+high)/2;
    d_fr = f(mid+1)-f(mid);
    d_fl = f(mid)-f(mid-1);
    if (d_fr<0 && d_fl>0)
      return mid;
    else if (d_fr<0)
      high = mid-1;
    else if (d_fl>0)
      low = mid+1;
  }
  return low;
}
```

חיפוש בינארי, כאשר מחשבים נגזרת לפי ערכים של שני נקודות סמוכות.
הבדיקה בהתלחה נחוצה כדי לא לחרוג מהטווח [0..n-1]

שאלה 3 (25 נקודות)

שאלה זאת עוסקת במיון מערכים.

סעיף א

בסעיף זה נתון מערך a באורך n, המאוחסן בזיכרון **שהכתיבה אליו מאוד איטית** (בניגוד לקריאה, שהיא מהירה מאוד). במילים אחרות, **בדיקת** התוכן של תא כלשהו במערך הינה פעולה מהירה, אולם **שינוי** ערכו של תא כלשהו הינה פעולה איטית. עליכם לממש פונקציה שתמיין את המערך, תוך שימוש במספר קטן ככל האפשר של כתיבות אליו.

דרישות סיבוכיות: על הפתרון לעבוד ב- O(1) סיבוכיות מקום נוסף (פתרון שלא עומד בדרישה זו לא יתקבל). כאמור, יש לבצע מספר קטן ככל האפשר של **כתיבות** למערך, וכן להשלים את סיבוכיות מספר הכתיבות במקום המתאים למטה. פרט לכך, **אין הגבלה** על סיבוכיות הזמן של הפתרון או על מספר הקריאות מהמערך.

מספר של הכתיבות למערך: $\Theta(n)$

<u>הערה: בסעיף זה ניתן להשתמש בפונקציות שנלמדו בכיתה.</u>

```
void sort_slow(int a[], int n) {
  max_sort(a, n);
}
```

maxsort אומנם עובד בסיבוכיות זמן $\Theta(n^2)$
אבל עושה רק $\Theta(n)$ כתיבות למערך.

סעיף ב

בסעיף זה עליכם שוב למיין מערך של מספרים שלמים. הפעם גודל המערך הוא k+m, כאשר k האיברים הראשונים, במקומות ה-0 עד ה-k-1, מאוחסנים בזיכרון איטי לכתיבה, ואילו m האיברים האחרונים, במקומות ה-k עד ה-k+m-1, מאוחסנים בזיכרון רגיל:

רגיל	רגיל	רגיל	רגיל	איטי	איטי	איטי
k+m-1	**...**	**k+1**	**k**	**k-1**	**...**	**0**

הפונקציה מופיעה בעמוד הבא.

דרישות סיבוכיות:

- עליכם לבצע מספר מועט ככל האפשר של כתיבות לזיכרון האיטי a[0]…a[k-1].
- כמו כן במידת האפשר, על הפתרון גם לעבוד בסיבוכיות זמן טובה ככל האפשר (עם זאת בכל מקרה העדיפות היא למספר כתיבות קטן ככל הניתן ל a[0]…a[k-1]).
- על הפתרון לעמוד בסיבוכיות מקום O(m). פתרון פתרון בסיבוכיות גרועה מזו יקבל ניקוד חלקי בלבד.

השלימו את מדדי הסיבוכיות של הפתרון שלכם במקום המתאים:

מספר הכתיבות לזיכרון האיטי: Θ(k)

סיבוכיות הזמן של הפונקציה: Θ( )

לשאלה זאת כמה פתרונות אפשריים

```
void sort_mixed(int a[], int k, int m) {
  int i;
  for (i=0; i<k; i++){          //this loop puts k smallest
    int min_ind = find_min(a+i,k+m-i); //members in place.
    swap(&a[i], &a[min_ind]);
  }
  mergesort(a+k, m);
}
```

פתרון זה עובד בסיבוכיות זמן $\Theta(k(m+k)+m\log m)$
(לא הכי יעיל, קיבל את רוב הניקוד)

```
void sort_mixed(int a[], int k, int m) {
  int i;
  mergesort(a+k,m);
  for (i=0; i<k; i++) {
    int len = min(k+1, k+m-i);
    ind min_ind = find_min(a+i,len);
    swap(&a[i], &a[min_ind+i]);
  }
  mergesort(a+k, m);
}
```

פתרון זה דומה לקודם, אבל קודם ממיינים את החלק השני של המערך, ולכן לולאת חיפוש האיבר הקטן מצתמצמת לגודל k במקום k+m.
פתרון זה עובד בסיבוכיות זמן $\Theta(k^2+m\log m)$

```
void sort_mixed(int a[], int k, int m) {
  int i;
  if (m>=k){
    int* a_copy = (int*)malloc(sizeof(int)*(k+m));
    for (i=0; i<k+m; i++)
      a_copy[i] = a[i];
    mergesort(a_copy, m+k);
    for (i=0; i<k+m; i++)
      a[i] = a_copy[i];
    free(a_copy);
  } else { // k>m
    max_sort(a, k+m);
  }
}
```

פתרון זה בודק מה יותר גדול, m או k.

- אם m יותר גדול אז $O(m)$ מקום לא מגביל אותנו וניצור עותק של מערך, נמיין ונעתיק למערך המקורי. (העותק הכרחי כי mergesort עושה הרבה כתיבות.
- אם k יותר גדול, אז הרי אין מנוס מלעשות k^2 פעולות כדי למיין את החלק האיטי, אז אפשר לעשות זאת גם עבור החלק המהיר (כי במקרה זה הוא אינו גדול מהחלק האיטי)

פתרון זה עובד בסיבוכיות זמן טובה יותר מהפתרון הקודם.

שאלה 4 (25 נקודות)

חברת התעופה Fly-PC פועלת ב-N ערים ברחבי עולם. החברה מפעילה קווי טיסה סדירים בין **חלק** מן הערים הללו, כאשר לכל טיסה נתון מחיר חיובי ממש. ערים שאין טיסה ישירה ביניהן, ניתן לעבור ביניהן באמצעות ביצוע מספר טיסות בזו אחר זו.

המטרה בשאלה זו הינה לכתוב פונקציה שבהינתן עיר מוצא ועיר יעד, מחשבות את המחיר המינימאלי הדרוש על מנת להגיע מעיר המוצא לעיר היעד. במידה ולא קיים מסלול מעיר המוצא לעיר היעד, על הפונקציה להחזיר 1-.

מחירי הטיסות נתונים במטריצה דו-ממדית E, כאשר התא E[i][j] מכיל את מחיר הטיסה מהעיר i לעיר j. אם אין טיסה בין i ל-j אז במטריצה יופיע 1- בתא המתאים. שימו לב שהמטריצה אינה בהכרח סימטרית: במילים אחרות, אם יש טיסה מ-i ל-j זה לא אומר שיש גם טיסה מ-j ל-i, וכן גם שאם יש טיסה כזאת, מחירה עשוי להיות שונה.

שימו לב:
- אנו מחפשים את המסלול **הזול ביותר** בין שתי הערים, ולא הקצר ביותר במחינת מספר הטיסות.
- בשאלה זו אין דרישות סיבוכיות.
- אין טיסות מעיר כלשהיא לעצמה, כלומר E[i][i] שווה 1- לכל i.

```
double cheapest_rate(double E[N][N], int from, int to) {
  return cheapest_rate_aux(E, from, to, N-1);
}

double cheapest_rate_aux(double E[N][N], int from, int to,
                         int maxlen) {

  double minprice = -1, route_price;
  int i;

  if (from == to) return 0;
  if (maxlen == 0) return -1;

  for (i=0; i<N; i++) {

    if (E[from][i]<0)
      continue;  // no direct flight to i

    route_price = cheapest_rate_aux(E, i, to, maxlen-1);
    if (route_price<0)
      continue;  // can't continue from i to destination

    if (minprice < 0 || minprice > E[from][i]+route_price)
      minprice = E[from][i]+route_price;
  }
  return minprice;
}
```

המסלול הזול ביותר יוצא מ- from בטיסה ישירה לעיר ביניים כלשהי (בלתי ידועה) p, וממשיך ממנה ל-to במסלול הזול ביותר. על מנת למצוא את p, אנו עוברים על כל האפשרויות לטוס מ-from לעיר כלשהי i, מחשבים רקורסיבית את המחיר הזול ביותר מ-i ל-to, וסוכמים את שני המחירים. העיר p היא זו שמביאה את מחיר המסע הכולל למינימום. שימו לב שייתכן שהמסלול הזול ביותר הוא פשוט טיסה ישירה מ-from ל-to, ומקרה זה מתואר על ידי הבחירה p==to.

יש לזכור שרקורסיה רק עובדת כאשר הבעיה הרקורסיבית קטנה ממש מהבעיה המקורית – אחרת יש לנו מצב של רקורסיה אינסופית ללא הקטנת הבעיה. לפיכך, הוסף לפונקציה פרמטר maxlen שקובע את מספר הטיסות המקסימאלי שאנו מרשים על מנת להגיע מ-from ל-to. בכל קריאה רקורסיבית אנו מקטינים מספר זה באחת כיוון שהשתמשנו בטיסה אחת על מנת להגיע מ-from ל-i, ובכך הבעיה הוקטנה. אם הגענו ל-0, אין פתרון למצב זה ולכן מחזירים 1-.