מבוא למדעי המחשב

תרגול 13: שאלות חזרה ממבחנים

חורף 2014 – מועד א' – שאלה 4

בשאלה זו נממש פונקציה שמקבלת סכום כסף נתון sum, ומחזירה את מספר הצורות השונות שבהן ניתן להגיע לסכום כסף זה באמצעות צירופים של מטבעות. ערכי המטבעות הזמינים לנו נמצאים במערך coins[] שאורכו n, כאשר כל ערך של מטבע מופיע במערך פעם אחת בדיוק. ניתן להניח שהמערך coins[] מכיל ערכים שלמים, חיוביים, ושונים זה מזה. שימו לב שניתן להשתמש בכל מטבע מספר לא מוגבל של פעמים על מנת להגיע לסכום sum.

לדוגמה, עבור המערך coins[] הבא:

```
int coins[2] = {1, 2};
```

ועבור הסכום sum=4, הפונקציה תחזיר 3 כיוון שניתן להגיע לסכום 4 באמצעות צירופי המטבעות הבאים:

1 + 1 + 1 + 1 = 4
1 + 1 + 2 = 4
2 + 2 = 4

שימו לב: בשאלה זו אין משמעות לסדר המחוברים. כלומר, הסכומים (1+1+2), (1+2+1) ו-(2+1+1) נחשבים לאותו הפירוק, ויש לספור אותם פעם אחת בלבד.

יש לפתור את השאלה ברקורסיה. בשאלה זו אין דרישות סיבוכיות.

חורף 2014 – מועד א' – שאלה 4

- נאמר לנו שהפתרון הוא פונקציה רקורסיבית. ראשית, נחפש לצמצם את הבעיה לבעיה קטנה יותר, שנוכל לצרף את פתרונה לפתרון של הבעיה הגדולה בדרך פשוטה.
- כיוון שיש שני פרמטרים לבעיה (sum, coins[]), נחשוב איך אנחנו יכולים לצמצם אחד מהם (או את שניהם).

חורף 2014 – מועד א' – שאלה 4

- ניתן לחשוב, שאולי ננסה לצמצם את הבעיה על ידי כך שנחסיר כל פעם מטבע כלשהו, ונקרא רקורסיבית לפונקציה עם סכום קטן יותר.
- הבעיה בגישה הזו היא שאנחנו עלולים לספור כך את אותה אפשרות פעמיים. למשל, עבור sum=3, coins = {1, 2}, נספור את האפשרות של 1+2 פעמיים – פעם כשלקחנו את 1 קודם, פעם כשלקחנו את 2.

חורף 2014 – מועד א' – שאלה 4

- אז מה עושים?
- נשים לב שמה שהפריע לנו ברעיון הקודם היא העובדה שיכולנו לקחת כל אחד מהמטבעות בכל שלב ברקורסיה.
- הפתרון הוא לחשוב כאילו אנחנו "מסדרים" את המטבעות שלנו, ולוקחים אותם תמיד לפי הסדר.

חורף 2014 – מועד א' – שאלה 4

- הקריאה הרקורסיבית שלנו תתחלק ל-2:
  - אם אנחנו מחליטים לקחת את המטבע הראשון במערך, צמצמנו את הבעיה לאותה בעיה בדיוק, אך עם סכום קטן יותר.
  - אם אנחנו מחליטים לא לקחת את המטבע הראשון, אז אסור לנו לחזור ולקחת אותו מתישהו בעתיד, ולכן נקטין את המערך שלנו.
- מה שעשינו פה בעצם הוא **לצמצם כל אחד מהפרמטרים של הבעיה בנפרד**. גישה זו עוזרת בהרבה בעיות רקורסיביות עם מספר פרמטרים.

חורף 2014 – מועד א' – שאלה 4

- ומה הוא תנאי העצירה?
  - עבור סכום 0, קיימת בדיוק אפשרות אחת (לא לקחת אף מטבע), לכן נחזיר 1.
  - אם הסכום אינו 0, ולא נשארו לנו יותר מטבעות או שהגענו לסכום שלילי (לקחנו מטבע גדול מידי), אז אין לנו דרך לחזור אחורה. נחזיר 0 (0 אפשרויות).

חורף 2014 – מועד א' – שאלה 4 - פתרון

```
int change(int coins[], int n, int sum) {
  if(sum==0)
    return 1;
  if(n==0 || sum<0)
    return 0;
  return
    change(coins, n, sum-coins[0])+ //pick coin 0
    change(coins+1, n-1, sum); //don't pick coin 0
}
```

אביב 2013 – מועד א' – שאלה 4

קליקה היא קבוצה של אנשים שבה כל זוג אנשים הם חברים. נתון מערך דו מימדי של חברויות `int friends[N][N]`: עבור זוג אנשים `i` ו- `j`, הערך `friends[i][j]` הוא 1 אם `i` ו-`j` חברים, ו-0 אחרת. המערך סימטרי, כלומר מתקיים `friends[i][j]==friends[j][i]` לכל `i` ו- `j`.

עליכם לכתוב פונקציה

```
int hasClique(int friends[][N], int k)
```

המקבלת מערך חברויות `friends`, ומספר `k`. הפונקציה תחזיר 1 אם קיימת קליקה המכילה לפחות `k` אנשים, ו-0 אם לא קיימת קליקה כזו.

לדוגמא, עבור מטריצת החברויות הבאה:

	0	1	2	3
0		1	1	0
1	1		1	0
2	1	1		1
3	0	0	1	

קבוצת החברים {0,1,2} היא קליקה בגודל 3, וקבוצת החברים {2,3} היא קליקה בגודל 2, אך לא קיימת קליקה בגודל 4 (כי לדוגמא 3 ו-0 לא חברים).

הערות:

- `N` מוגדר ע"י `#define`.
- יש להשתמש בשיטת backtracking כפי שנלמדה בכיתה.
- בשאלה זו אין דרישות סיבוכיות, אולם כמקובל ב-backtracking יש לוודא שלא מתבצעות קריאות רקורסיביות מיותרות עם פתרונות שאינם חוקיים.

אביב 2013 – מועד א' – שאלה 4

- כיוון שרומזים לנו לפתור את השאלה בעזרת backtracking, כנראה שאין מנוס מלנסות את כל האפשרויות **במקרה הגרוע**.
- נשים לב ראשית, שמתקיים: קיימת קליקה המכילה לפחות k אנשים אם ורק אם קיימת קליקה המכילה *בדיוק* k אנשים.
- לכן, נוכל לצמצם את הבעיה למציאת קליקה המכילה בדיוק k אנשים.

אביב 2013 – מועד א' – שאלה 4

- גם כאן, כמו בשאלה הקודמת, מתחבאים שני פרמטרים לבעיה. הראשון ברור – k. השני הינו קבוצת האנשים מתוכה אנחנו בוחרים.
- נוכל לפעול כאן באותה שיטה בדיוק. נסתכל על האנשים לפי סדר מסוים, ובכל קריאה רקורסיבית נחליט אם ננסה להכניס אדם זה לקליקה או האם נוותר עליו.
- כאשר נרצה להכניס אדם לקליקה, צריך רק לדאוג לבדוק האם הוא חבר של כל שאר האנשים שכבר בקליקה.
- לשם ייצוג בחירת האנשים, נשתמש במערך דגלים, שם יסומן עבור כל אדם האם הוא נבחר או לא.

אביב 2013 – מועד א' – שאלה 4

```
int clique_aux(int friends[N][N], int k, int chosen[], int curr)
{
 if(k==0) return 1;
 if(curr==N) return 0;

 // Without k-th person
 chosen[curr]=0;
 if (clique_aux(friends, k, chosen, curr+1))
  return 1;

 // With k-th person
 int i;
 for (i=0; i<curr; ++i){
  if (chosen[i] && friends[curr][i]==0)
   return 0;
 }
 chosen[curr]=1;
 return clique_aux(friends, k-1, chosen, curr+1);
}
```

אביב 2013 – מועד א' – שאלה 4

```
int clique(int friends [N][N], int k){
 int chosen[N];
 return clique_aux(friends, k, chosen, 0);
}
```

חורף 2013 – מועד ב' – שאלה 3

שאלה 3 (25 נקודות)

נתון מערך חד מימדי a[] באורך n המכיל ערכי 0 ו-1 בלבד. לדוגמה:

a[] = [0,1,1,1,1,1,1,0,1,1,1,1,1,0,1,1]

עליכם לכתוב תוכנית המקבלת כקלט את המערך a ואת אורכו n, ומחליפה כל ערך במערך שאיננו 0 במרחק ממנו לנקודת האפס הקרובה ביותר (מימין או משמאל). שימו לב שב"מרחק" הכוונה פשוט למספר הצעדים במערך שיש לבצע על מנת להגיע לאפס הקרוב ביותר. לדוגמה, עבור המערך a הנ"ל, הפלט יהיה:

a[] = [0,1,2,3,3,2,1,0,1,2,3,2,1,0,1,2]

הערות:

- על הפתרון לעמוד בסיבוכיות זמן O(n) כאשר n הוא אורך המערך, וסיבוכיות מקום O(1).
- ניתן להניח שהמערך מכיל לפחות 0 אחד (הוא אינו כולו 1).

חורף 2013 – מועד ב' – שאלה 3

- פתרון נאיבי: לעבור על המערך תא אחרי תא, וכל פעם שמגיעים ל-1, לחפש את האפס הראשון משמאלו ומימינו, ולקחת את המינימום ביניהם.
- מה הבעיה?
- הסיבוכיות במקרה הגרוע היא $\Theta\left(n^2\right)$, למשל אם יש רק שני אפסים בקצוות המערך, וכל השאר 1-ים.

חורף 2013 – מועד ב' – שאלה 3

- נצטרך לחשוב על פתרון אחר, שעובר על המערך מספר קבוע של פעמים.
- ננסה לחלק את הבעיה לשני חלקים. אם נדע עבור כל 1 במערך את מרחקו לאפס הראשון משמאלו, ואת מרחקו לאפס הראשון מימינו, נוכל יחסית בקלות לפתור את הבעיה.
- איך מחשבים במעבר אחד על המערך את מרחק האפס הקרוב ביותר **משמאל** לכל אחד מה1-ים במערך?

חורף 2013 – מועד ב' – שאלה 3

- נתחיל מצד שמאל ונשמור מונה של מספר הצעדים שעשינו בלי להיתקל ב-0, ונאפס אותו כאשר ניתקל באחד. עבור כל 1 שנגיע אליו, מרחקו מהאפס הקרוב ביותר משמאל יהיה בדיוק ערך המונה! נרשום ערך זה במקום ה-1 ונמשיך הלאה.
- לאחר מעבר זה, המערך מכיל במקום ה-1-ים, את מרחק מהאפס הכי קרוב אליהם **משמאל**. נוכל לעשות בדיוק את אותו הדבר מהצד השני, רק שהפעם נצטרך לבחור במינימום בין ערך המונה ובין הערך הרשום במערך.

חורף 2013 – מועד ב' – שאלה 3

- נתחיל מצד שמאל ונשמור מונה של מספר הצעדים שעשינו בלי להיתקל ב-0, ונאפס אותו כאשר ניתקל באחד. עבור כל 1 שנגיע אליו, מרחקו מהאפס הקרוב ביותר משמאל יהיה בדיוק ערך המונה! נרשום ערך זה במקום ה-1 ונמשיך הלאה.
- לאחר מעבר זה, המערך מכיל במקום ה-1-ים, את מרחק מהאפס הכי קרוב אליהם **משמאל**. נוכל לעשות בדיוק את אותו הדבר מהצד השני, רק שהפעם נצטרך לבחור במינימום בין ערך המונה ובין הערך הרשום במערך.

חורף 2013 – מועד ב' – שאלה 3

```
void zero_dist(int a[], int n) {
 int dist = n;
 for (int i = 0; i < n; ++i) {
   if (a[i] == 0)
     dist = 0;
   a[i] = dist;
   dist++;
 }
 dist = n;
 for (int i = n-1; i >= 0; --i) {
   if (a[i] == 0)
     dist = 0;
   a[i] = (a[i] < dist) ? a[i] : dist;
   dist++;
 }
}
```

אביב 2012 – מועד ב' – שאלה 3

עליכם לכתוב פונקציית חיפוש במערך ממוין **לא-יורד** של מספרים **חיוביים**, ש"משמידה" את האברים במקומות שנמצאו בעבר. במילים אחרות, לפונקציה אסור להחזיר פעמיים את אותו המיקום. מותר לפונקציה לשנות את תוכן מערך החיפוש, אבל הדרישה הבאה חייבת להתקיים:

אם המספר x מופיע k פעמים במערך המקורי (לפני החיפוש הראשון) במקומות $i_1..i_k$, אז k החיפושים הראשונים של x יחזירו $i_1..i_k$ בסדר כלשהו, והקריאה ה- k+1 ואילך מחזירה -1.

חתימת הפונקציה:

int seek_and_destroy(int* a, int n, int x)

a הוא מערך החיפוש, n הוא אורכו ו- x הוא המספר המבוקש. על הפונקציה להחזיר את המיקום של x אם קיים, ו- -1 אחרת.

דרישות:

סיבוכיות זמן של כל קריאה לפונקציה O(log n), סיבוכיות מקום נוסף של קריאה לפונקציה: O(1)

הצעה: ניתן להשתמש במספרים שליליים על-מנת לסמן ערכים שכבר נמצאו.

הערה: אם כתבתם פתרון בסיבוכיות זמן שהיא לא O(log n), תוכלו לקבל בחזרה חלק מהנקודות אם תכתבו כאן את הסיבוכיות של הפתרון שלכם כתלות בגודל המערך n ומספר החזרות M של האיבר שמופיע הכי הרבה פעמים.

אביב 2012 – מועד ב' – שאלה 3

- בשלב זה, אנחנו כבר אמורים להכיר טוב את מילות המפתח בשאלה שמכוונות אותנו לפתרון: "מערך ממוין" + "חיפוש" + "O(logn)" == **חיפוש בינארי**.
- ואכן זה המצב. הבסיס של הפתרון הוא מימוש חיפוש בינארי, אך בעת מציאת ערך, אנחנו צריכים לשנות אותו כך שלא נמצא אותו יותר. מצד שני אסור לנו פשוט "למחוק" אותו (למשל לשים במקומו 0, שמובטח שלא הופיע במערך המקורי), כי אז לא נוכל להשתמש בחיפוש הבינארי בקריאות הבאות.

אביב 2012 – מועד ב' – שאלה 3

- רעיון: נשתמש ברמז שניתן לנו, ופשוט נהפוך כל מספר שאנחנו מוצאים למינוס של אותו המספר.
- במבט ראשון זה נראה נכון, אבל מה נעשה בקריאה נוספת לחיפוש הבינארי?
- אם בכל קריאה נחזיר תוצאה **כלשהי**, בקריאות הבאות לחיפוש של אותו הערך לא נדע כיצד להתקדם אם mid ינחת על הערך שאנחנו מחפשים אך שכבר נמצא בעבר.

אביב 2012 – מועד ב' – שאלה 3

- הפתרון: למרות שלא נדרשנו לכך, נחזיר בכל קריאה את האינדקס **השמאלי ביותר** של הערך שמחפשים (שטרם החזרנו).
- כך, אם ננחת על הערך שאותו אנו מחפשים אך הוא שלילי, נדע שצריך להמשיך לחפש בצד ימין של המערך.

אביב 2012 – מועד ב' – שאלה 3

```
int seek_and_destroy(int* a, int n, int x) {
   int left = 0, right = n - 1, mid, val;
   while (left <= right) {
         mid = (left + right) / 2;
         val = a[mid];
         if (val < 0) {
                  if (val == -x) {
                           left = mid + 1;
                           continue;
                  } else
                           val = -val;
         }
         if (val < x)
                  left = mid + 1;
         else if (val > x)
                  right = mid - 1;
         /* This will ensure that we return leftmost. we stop if val==x AND left=right */
         else if (left == right) {
                  a[left] = -a[left]; /* found it - destroy it */
                  return left;
         } else { /* Found but MIGHT not be leftmost */
                  right = mid; /* watch out for the '-1' */
         }
   }
   return -1;
}
```

אביב 2011 – מועד א' – שאלה 3

נתון מערך a של מספרים שלמים, באורך n. ניתן להניח ש- n הוא חזקה שלמה חיובית של 2,

כלומר $n=2^k$ עבור $k>0$ כלשהו.

המערך ייקרא <u>ממוין בממוצעים</u> אם מתקיים:

1. המערך הוא באורך 2 וגם $a[0] <= a[1]$.

או:

2. המערך הוא באורך 2^k עבור $k>1$, וגם:

א. החצי השמאלי של המערך כשלעצמו הוא ממוין בממוצעים.

ב. החצי הימני של המערך כשלעצמו הוא ממוין בממוצעים.

ג. הממוצע של החצי השמאלי אינו גדול מהממוצע של החצי הימני.

אביב 2011 – מועד א' – שאלה 3

דוגמא: הסדרה הבאה:

14	21	20	30

ממוינת **בממוצעים**. לחצי השמאלי יש ממוצע 17.5. לחצי הימני יש ממוצע 25. מתקיים 25 <= 17.5, כדרוש. בנוסף, החצי השמאלי כשלעצמו ממוין בממוצעים כי מתקיים 21 <= 14, החצי הימני כשלעצמו ממוין בממוצעים כי 30 <= 20.

לעומת זאת, הסדרה הבאה:

90	110	40	50

אינה ממוינת בממוצעים כי הממוצע של החצי השמאלי הוא 100 בעוד הממוצע של הימני 45, ומתקיים 100>45.

כתבו פונקציה שחתימתה:

```
int check_sorted_in_averages(int a[], int n)
```

הפונקציה מקבלת מערך a באורך n, כאשר n חזקה שלמה חיובית של 2, ומחזירה 1 אם הוא ממוין בממוצעים, אחרת 0. אין לשנות את אברי מערך הקלט בחזרה מהפונקציה.

דרישות: סיבוכיות זמן O(n), מקום O(log n).

הערה: פתרון נכון בסיבוכיות זמן O(n log n) ומקום O(log n) **או** בסיבוכיות זמן O(n) ומקום O(n) יזכה ברוב הנקודות המוקצות לשאלה זו.

אביב 2011 – מועד א' – שאלה 3

- נשים לב שהבעיה עצמה מוגדרת באופן רקורסיבי, לכן סביר מאוד להניח שהפתרון שלה יהיה רקורסיבי.
- תנאי העצירה והצעד ברקורסיה רשומים לנו, לכן נוכל פשוט לגשת ולממש אותם.
- אבל רגע – הפונקציה שביקשנו לממש מחזירה רק האם מערך הוא **ממוין בממוצעים**, אבל בצעד הרקורסיה נדרש לנו מידע נוסף – **הממוצע עצמו של אותו מערך**.

אביב 2011 – מועד א' – שאלה 3

- מה נעשה?
- נוכל לחשב בעצמו את הממוצע של כל חצי. אם נעשה זאת, נעשה O(k) פעולות בכל קריאה רקורסיבית המקבלת מערך באורך O(k). זה מזכיר את מה שאנחנו עושים בmerge-sort, ולכן סיבוכיות זמן הריצה תהיה O(nlogn).
- אבל אנחנו יכולים לעשות יותר טוב מזה.

אביב 2011 – מועד א' – שאלה 3

- נשים לב, שאם היו מגלים לנו את הממוצע של כל אחד מחצאי המערך, היינו יכולים בקלות לחשב את הממוצע של המערך כולו, וכך להחזיר במעלה הרקורסיה את ערך הממוצע בלי לעבור על האיברים כלל.
- בשביל להחזיר **גם** את הממוצע של המערך, נצטרך להוסיף לפונקציה ארגומנט שיהיה מצביע לערך שאותו תעדכן הפונקציה. כיוון ששינינו את חתימת הפונקציה, נהפוך אותה לפונקציית עזר.

אביב 2011 – מועד א' – שאלה 3

```
int check_sorted_in_averages_helper(int a[], int n, double* average) {
  int result_left, result_right;
  double av_left, av_right;

  if (n == 2) {
       *average = (a[0] + a[1]) / 2.0;
       return a[0] <= a[1];
  }

  result_left = check_sorted_in_averages_helper(a, n / 2, &av_left);
  result_right = check_sorted_in_averages_helper(a + n / 2, n / 2,
  &av_right);
  *average = (av_left + av_right) / 2.0;
  return result_left && result_right && (av_left <= av_right);
}

int check_sorted_in_averages(int a[], int n) {
  double average;
  return check_sorted_in_averages_helper(a, n, &average);
}
```

אביב 2009 – מועד א' – שאלה 1 (ב')

```
void g(int n);

void h(int n){
     int i;
     if (n < 1) return;
      for (i = 0; i < n; i++)
          printf("?");
      h(n/2);
}

void f(int n){
   int i;
   if (n < 2) return;
   for (i=0; i<n/2; i+=5)
      printf("!");
   g(n/3);
   g(n/3);
}

void g(int n){
   int i;
   for (i = 0; i < n; i++)
      printf("?");
   f(3*n/2);
}
```

- מה סיבוכיות הזמן והמקום של הפונקציה h כתלות בn?
- מה סיבוכיות הזמן והמקום של הפונקציה f כתלות בn?

אביב 2009 – מועד א' – שאלה 1 (ב')

```
void g(int n);

void h(int n){
     int i;
     if (n < 1) return;
      for (i = 0; i < n; i++)
          printf("?");
      h(n/2);
}

void f(int n){
   int i;
   if (n < 2) return;
   for (i=0; i<n/2; i+=5)
      printf("!");
   g(n/3);
   g(n/3);
}

void g(int n){
   int i;
   for (i = 0; i < n; i++)
      printf("?");
   f(3*n/2);
}
```

הפונקציה h הינה פונקציה רקורסיבית. היא קוראת לעצמה פעם אחת בלבד בגוף הפונקציה, ובכל קריאה היא מבצעת O(k) עבודה, כאשר k הוא הארגומנט לפונקציה. נשים לב כי הקריאה בעומק i מקבלת כארגומנט את $n/2^i$.

יהיו סדר גודל של log(n) קריאות רקורסיביות, ולכן סיבוכיות הזמן:

$$\sum_{i=0}^{\log_2(n)} \frac{n}{2^i} = n \cdot \underbrace{\sum_{i=0}^{\log_2(n)} \frac{1}{2^i}}_{converging\ \ sum} = \theta(n)$$

אביב 2009 – מועד א' – שאלה 1 (ב')

```
void g(int n);

void h(int n){
     int i;
     if (n < 1) return;
      for (i = 0; i < n; i++)
          printf("?");
      h(n/2);
}

void f(int n){
   int i;
   if (n < 2) return;
   for (i=0; i<n/2; i+=5)
      printf("!");
   g(n/3);
   g(n/3);
}

void g(int n){
   int i;
   for (i = 0; i < n; i++)
      printf("?");
   f(3*n/2);
}
```

יהיו סדר גודל של log(n) קריאות רקורסיביות, ולכן סיבוכיות הזמן:

$$\sum_{i=0}^{\log_2(n)} \frac{n}{2^i} = n \cdot \underbrace{\sum_{i=0}^{\log_2(n)} \frac{1}{2^i}}_{converging\ \ sum} = \theta(n)$$

והמקום: $\theta(\log n)$

אביב 2009 – מועד א' – שאלה 1 (ב')

```
void g(int n);

void h(int n){
     int i;
     if (n < 1) return;
      for (i = 0; i < n; i++)
          printf("?");
      h(n/2);
}

void f(int n){
   int i;
   if (n < 2) return;
   for (i=0; i<n/2; i+=5)
      printf("!");
   g(n/3);
   g(n/3);
}

void g(int n){
   int i;
   for (i = 0; i < n; i++)
      printf("?");
   f(3*n/2);
}
```

הפונקציות f ו-g קוראות אחת לשנייה. נסמן את זמני הריצה שלהן: $T_g(n), T_f(n)$.

מתקיים: $T_f(n) = 2T_g\left(\frac{n}{3}\right) + \frac{n}{2\cdot 5}c_1 + c_2$

$$T_g(n) = T_f\left(\frac{3n}{2}\right) + n\cdot c_3 + c_4$$

נציב את $T_g(n)$, ונקבל:

$$T_f(n) = 2\left(T_f\left(\frac{3\left(\frac{n}{3}\right)}{2}\right) + \frac{n}{3}\cdot c_3 + c_4\right) + \frac{n}{2\cdot 5}c_1 + c_2$$

אביב 2009 – מועד א' – שאלה 1 (ב')

$$T_f(n) = 2\left(T_f\left(\frac{3\left(\frac{n}{3}\right)}{2}\right) + \frac{n}{3}\cdot c_3 + c_4\right) + \frac{n}{2\cdot 5}c_1 + c_2$$

$$= 2\left(T_f\left(\frac{n}{2}\right) + \frac{n}{3}\cdot c_3 + c_4\right) + \frac{n}{10}c_1 + c_2 =$$

$$2T_f\left(\frac{n}{2}\right) + \frac{2}{3}c_3\cdot n + 2c_4 + \frac{c_1}{10}n + c_2 =$$

$$2T_f\left(\frac{n}{2}\right) + c_5\cdot n + c_6$$

אביב 2009 – מועד א' – שאלה 1 (ב')

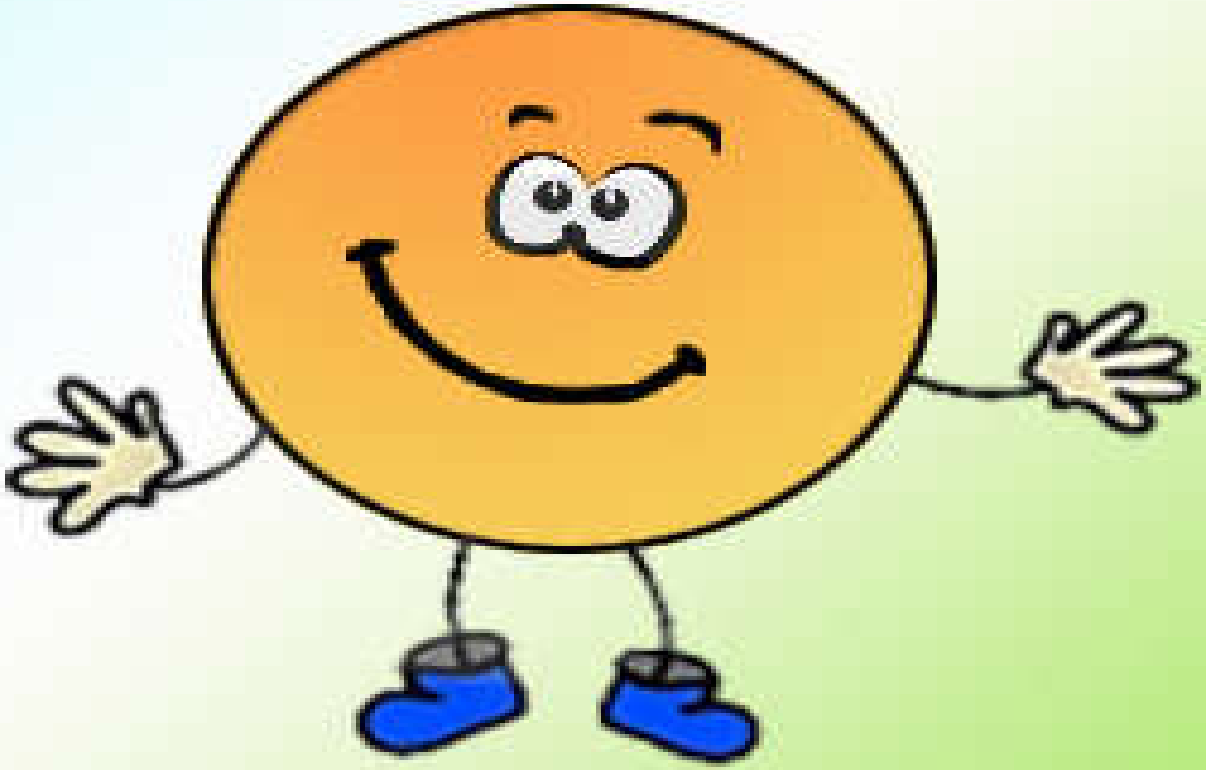
```
void g(int n);

void h(int n){
     int i;
     if (n < 1) return;
      for (i = 0; i < n; i++)
          printf("?");
      h(n/2);
}

void f(int n){
   int i;
   if (n < 2) return;
   for (i=0; i<n/2; i+=5)
      printf("!");
   g(n/3);
   g(n/3);
}

void g(int n){
   int i;
   for (i = 0; i < n; i++)
      printf("?");
   f(3*n/2);
}
```

קיבלנו כי $T_f(n) = 2T_f\left(\frac{n}{2}\right) + c_5 n + c_6$

ניתן לפתח פיתוח טלסקופי, או להיזכר שזה בדיוק הביטוי שקיבלנו עבור ניתוח הסיבוכיות של merge-sort, ולכן סיבוכיות זמן הריצה היא $\theta(n \log n)$

בכל קריאה רקורסיבית אנחנו מקטינים את n בפקטור כפלי, ולכן עומק הרקורסיה לא יעלה על סדר גודל של $\theta(\log n)$ - וזו סיבוכיות המקום.

Good Luck



All The Best