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

## סמסטר חורף תשע"ט

## מבחן מסכם מועד א', 19 בפברואר 2019

| מספר סטודנט: | | | | | | | | | | רשום/ה לקורס: | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | 2 | 3 | 4 | 1 | 1 |

**משך המבחן: 3 שעות**.

**חומר עזר:** אין להשתמש בכל חומר עזר.

**הנחיות כלליות:**

- מלאו את הפרטים בראש דף זה ובדף השער המצורף, בעט בלבד.
- 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 תבדקנה.
- יש לכתוב באופן ברור, נקי ומסודר, **ובעט בלבד**.
- בכל השאלות, הינכם רשאים להגדיר ולממש פונקציות עזר כרצונכם. לנוחיותכם, אין חשיבות לסדר מימוש הפונקציות בשאלה, ובפרט ניתן לממש פונקציה לאחר השימוש בה.
- אלא אם כן נאמר אחרת בשאלות, **אין להשתמש בפונקציות ספריה או בפונקציות שמומשו בכיתה**, למעט פונקציות קלט/פלט והקצאת זיכרון (`malloc, free`). ניתן להשתמש בטיפוס `bool` המוגדר ב-`stdbool.h`
- אין להשתמש במשתנים סטטיים וגלובאליים אלא אם נדרשתם לכך מפורשות.
- כשאתם נדרשים לכתוב קוד באילוצי סיבוכיות זמן/מקום נתונים, אם לא תעמדו באילוצים אלה תוכלו לקבל בחזרה מקצת הנקודות אם תחשבו נכון ותציינו את הסיבוכיות שהצלחתם להשיג.
- נוהל "לא יודע": אם תכתבו בצורה ברורה "לא יודע/ת" על שאלה (או סעיף) שבה אתם נדרשים לקודד, תקבלו 20% מהניקוד. דבר זה מומלץ אם אתם יודעים שאתם לא יודעים את התשובה.
- נוסחאות שימושיות:

$$1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}=\Theta(\log n) \qquad 1+\frac{1}{4}+\frac{1}{9}+\frac{1}{16}+\frac{1}{25}+\ldots=\Theta(1)$$

$$1+2+\ldots+n=\Theta(n^2) \qquad 1+4+9+\ldots+n^2=\Theta(n^3) \qquad 1+8+27+\ldots+n^3=\Theta(n^4)$$

צוות הקורס 234114/7

**מרצים**: פרופ' תומר שלומי, ד"ר אהרון קופרשטוק, ד"ר רמי כהן, מר טל טבצ'ניק **מתרגלים:** דניאלה בר לב, עדי גרוס, עמר דהרי, ניר חכמוביץ, יארה מולא, גסוב מזאוי, נג'יב נבואני, דניאל עזוז, אלון פאר, דמיטרי רבינוביץ' (מתרגל אחראי), יאיר ריעאני, עידו רפאל, אסף שומר

**בהצלחה!**

## שאלה 1 (25 נקודות):

א. (8 נקודות) חשבו את סיבוכיות הזמן והמקום של הפונקציה f1 המוגדרת בקטע הקוד הבא, כפונקציה של n. אין צורך לפרט שיקוליכם. חובה לפשט את הביטוי ככל שניתן.

```
int f1(int n)
{
      int x = 0;

      for (int i = 1; i <= n; ++i)
            for (int j = i; j <= n; j += i)
                  ++x;

      return x;
}
```

סיבוכיות זמן: $\Theta(n \log n)$ סיבוכיות מקום: $\Theta(1)$

ב. (8 נקודות): חשבו את סיבוכיות הזמן והמקום של הפונקציה f2 המוגדרת בקטע הקוד הבא, כפונקציה של n. אין צורך לפרט שיקוליכם. חובה לפשט את הביטוי ככל שניתן.

```
int g(int arr[], int start, int end, int k)
{
      if (start > end) return 0;

      int mid = (start + end) / 2;

      if (arr[mid] < k) return 1 + g(arr, mid + 2, end, k);
      if (arr[mid] > k) return 1 + g(arr, start, mid - 1, k);

      return g(arr, start, mid - 1, k) + 1 +
            g(arr, mid + 1, end, k);
}

int f2(int arr[], int n, int k)
{
      return g(arr, 0, n - 1, k);
}
```

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

ג. (9 נקודות): חשבו את סיבוכיות הזמן והמקום של הפונקציה f3 המוגדרת בקטע הקוד הבא, כפונקציה של n. אין צורך לפרט שיקוליכם. חובה לפשט את הביטוי ככל שניתן.

```
int g(int n)
{
      if (n <= 1) return 1;
      return g(n / 2) + 1;
}

int f3(int n)
{
      int counter = 0;
      for (int i = 1; g(i) * g(i) < n; i *= 2)
            ++counter;
      return counter;
}
```

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

## שאלה 2 (25 נק')

נתון מערך ממוין של מספרים שלמים `arr` באורך `n` ומספר `x`. ממשו פונקציה

```
int count_occurences(int arr[], int n, int x)
```

שתחזיר את מספר ההופעות של מספר `x` במערך `arr`.

לדוגמה: אם נתון מערך `arr` הבא

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -5 | 0 | 1 | 1 | 3 | 3 | 4 | 4 | 4 | 7 | 13 | 42 |

עבור `x=0`, הפונקציה תחזיר `1`

עבור `x=4`, הפונקציה תחזיר `3`

עבור `x=8`, הפונקציה תחזיר `0` (שימו לב, `8` **אינו** מופיע במערך)

**דרישות**:

- סיבוכיות זמן: $\mathbf{O(\log n)}$ .

אם לפי חישוביכם לא עמדתם בדרישות הסיבוכיות אנא ציינו כאן את הסיבוכיות שהגעתם אליה:

זמן _____ מקום נוסף _____

```
int count_occurences(int arr[], int n, int x)
{
      int prev = find_last_prev(arr, n, x),
            last = find_last_prev(arr, n, x + 1);
      return last - prev;
}
```

```
int find_last_prev(int arr[], int n, int x)
{
	int left = 0, right = n - 1;
	while (left <= right)
	{
		int mid = left + (right - left) / 2;
		if (arr[mid] >= x)
			right = mid - 1;
		if (arr[mid] < x)
			left = mid + 1;
	}

	return left - 1;
}
```

## שאלה 3 (25 נקודות) :

להלן קטע הקוד שמממש merge sort באופן איטרטיבי כפי שהוצג בהרצאה:

```
#define FAILURE -1
#define SUCCESS 0

int merge_sort(int ar[], int n)
{
	int len;
	int *temp_array, *base;

	temp_array = (int*) malloc(sizeof(int)*n);
	if (temp_array == NULL)
	{
		printf("Dynamic Allocation Error in merge_sort");
		return FAILURE;
	}

	for (len = 1; len < n; len *= 2)
	{
		for (base = ar; base < ar + n; base += 2 * len)
		{
			merge(base, len, base + len, len, temp_array);
			memcpy(base, temp_array, 2 * len * sizeof(int));
		}
	}

	free(temp_array);
	return SUCCESS;
}
```

יש לשכתב את הקוד כך שפקודת העתקת הזיכרון תתבצע לכל היותר פעם אחת לפני החזרה מביצוע הפונקציה merge_sort.

בקוד המשוכתב, אין להשתמש בפקודת memcpy או פקודת/פעולת העתקה אחרת של מערכים/תתי מערכים בתוך הלולאות. ניתן להשתמש בפונקציה merge רק לצרכי מיזוג.

| |
|---|
| int merge_sort(int ar[], int n) { |
| |
| |
| |
| |

```
int merge_sort(int arr[], int n)
{
        int *temp_array, *base;

        temp_array = (int*) malloc(sizeof(int)*n);
        if (temp_array == NULL)
        {
                printf("Dynamic Allocation Error in merge_sort");
                return FAILURE;
        }

        int *from = arr, *to = temp_array;

        for (int len = 1; len < n; len *= 2)
        {
                for (base = from; base < from + n; base += 2 * len)
                        merge(base, len, base + len, len, to + (base - from));

                swap(&from, &to);
        }

        if(from != arr)
                memcpy(arr, temp_array, n * sizeof(int));

        free(temp_array);
        return SUCCESS;
}
```

## שאלה 4 (25 נקודות) :

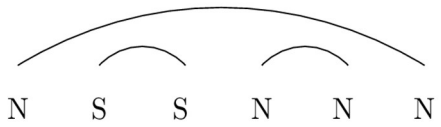
כדי לפתור את בעיית הפקקים בישראל, החליטה הממשלה לחבר כבישים מרכזיים באמצעות גשרים. הרעיון הוא לחבר זוג כבישים הפונים לאותו כיוון באמצעות גשר.

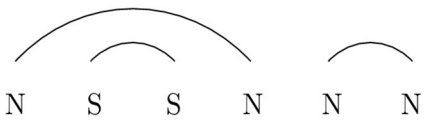
אנשי משרד התחבורה חילקו את כבישי ישראל מדרום עד צפון למערך של אותיות 'N' ו- 'S', כאשר כל אות מייצגת כביש הפונה לכיוון צפון (north) או דרום (south) בהתאמה. עלינו לבחור זוגות שונים של כבישים הפונים לאותו כיוון, כך שבין זוגות כבישים יהיה ניתן לחבר גשר, מבלי שגשרים שונים יחצו זה את זה.

לדוגמא, עבור המערך:

N S S N N N

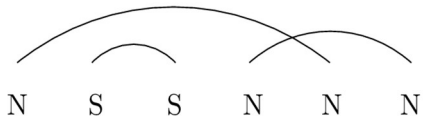
פתרונות אפשריים הם:




כאשר הגשרים מיוצגים ע"י הקשתות המחברות בין האותיות.

פתרון שגוי לבעיה הינו:

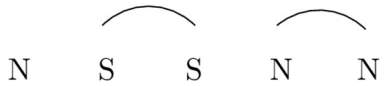



משום שבסידור זה שני גשרים נחתכים.

עבור המערך :

N S S N N

הפתרון האפשרי הינו:




**ראו המשך השאלה בעמוד 17.**

ממשו את הפונקציה build_bridges שמקבלת את מערך כיווני הכבישים roads, את גודלו n ומערך פלט bridges בגודל n:

```
int build_bridges(char roads[], int n, int bridges[])
```

ורושמת במערך הפלט bridges את הפתרון האפשרי המקסימלי (אשר מחבר את מספר זוגות הגדול ביותר). במידה והוחלט להשאיר כביש **לא מחובר** ייכתב 1- במקום המתאים. על הפונקציה להחזיר את מספר הגשרים בפתרון ולמקם במערך bridges מספרים שלמים עוקבים החל מ- 0 המייצגים פתרון כלשהו של הבעיה, כך שתאים במערך החולקים את אותו מספר ייצגו זוג כבישים המחובר בגשר.

לדוגמא, עבור המערך {N, S, S, N, N, N} הפתרון הראשון שהצגנו ניתן לכתיבה במערך bridges כ- {0, 1, 1, 2, 2, 0}, והפתרון השני ניתן לכתיבה כ- {0, 1, 1, 0, 2, 2}. בעוד עבור הדוגמה השנייה ייכתב {-1, 0, 0, 1, 1}.

במידה ולא קיים פתרון לבעיה, על הפונקציה להחזיר 0.

**הערות:**

- 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.
- ניתן ומומלץ להשתמש בפונקציות עזר (ויש לממש את כולן).

```
int build_bridges(char roads[], int n, int bridges[])
{
	for (int i = 0; i < n; ++i)
		bridges[i] = NO_BRIDGE;
	return build_bridges_aux(roads, n, bridges, 0, 0);
}

#define NO_BRIDGE -1
#define UNINITIALIZED -1

int max(int x, int y)
{
	return x > y ? x : y;
}
```

```
bool IsSolutionValid(int bridges[], int n)
{
	int * above = (int*) malloc(n * sizeof(int));
	for (int i = 0; i < n; ++i)
		above[i] = UNINITIALIZED;

	int bridges_above = 0;
	bool valid = true;

	for (int i = 0; i < n; ++i)
	{
		if (bridges[i] == NO_BRIDGE)
			continue;

		if (above[bridges[i]] != UNINITIALIZED)
		{
			if (bridges_above-- != above[bridges[i]])
			{
				valid = false;
				break;
			}
		}
		else
			above[bridges[i]] = ++bridges_above;
	}

	free(above);
	return valid;
}
```

```
int build_aux (char roads[], int n, int bridges[], int curr_road,
                                    int curr_bridge)
{
        if (curr_road == n) return curr_bridge;

        int *best_b = GetCopy(bridges, n);
        int highest_seen = build_aux(roads,n,best_b,curr_road+1,curr_bridge);
        if (bridges[curr_road] == NO_BRIDGE) {
                bridges[curr_road] = curr_bridge;
                for (int end = curr_road + 1; end < n; ++end) {
                        if ((roads[end] != roads[curr_road]) ||
                                (bridges[end]!=NO_BRIDGE))
                                continue;
                        highest_seen = TryConnect(roads,n,bridges,curr_road,
                                        curr_bridge,end,best_b,highest_seen);
                }
        }

        CopyArray(best_b, bridges, n);
        free(best_b);
        return highest_seen;
}

void CopyArray(int *src, int* dst, int n)
{
        memcpy(dst, src, n * sizeof(int));
}
```

```
int TryConnect(char roads[], int n, int bridges[], int curr_road,
              int curr_bridge, int end, int best_sol[], int highest_seen)
{
       int *curr_b = GetCopy(bridges, n);
       curr_b[end] = curr_bridge;
       if (IsSolutionValid(curr_b, n))
       {
              int num_bridges =
              build_aux(roads, n, curr_b, curr_road + 1, curr_bridge + 1);
              if (num_bridges > highest_seen)
              {
                     highest_seen = num_bridges;
                     CopyArray(curr_b, best_sol, n);
              }
       }
       free(curr_b);

       return highest_seen;
}

int* GetCopy(int *src, int n)
{
       int *dst = malloc(n * sizeof(int));
       CopyArray(src, dst, n);
       return dst;
}
```