מדינת ישראל
משרד החינוך

סוג הבחינה: גמר לבתי־ספר לטכנאים ולהנדסאים
מועד הבחינה: אביב תשפ"ב, 2022
סמל השאלון: 714911
נספחים: א. דף תשובות לפרק השני
ב. מילון מונחים

מבני נתונים ויעילות אלגוריתמים

הוראות לנבחן

א. משך הבחינה: ארבע שעות.

ב. מבנה השאלון ומפתח ההערכה: בשאלון זה שני פרקים.

פרק ראשון:	70	נקודות
פרק שני:	30	נקודות
סך־הכול:	100	נקודות

ג. חומר עזר מותר בשימוש: כל חומר עזר כתוב בכתב־יד או מודפס על נייר.

ד. הוראות מיוחדות:

1. את התשובות לשאלות **שבפרק הראשון** יש לכתוב במחברת הבחינה.
   את התשובות לשאלות **שבפרק השני** יש לסמן **אך ורק** על גבי דף התשובות שבנספח א'.
2. ענו על מספר השאלות הנדרש בשאלון. המעריך יקרא ויעריך את מספר התשובות הנדרש בלבד, לפי סדר כתיבתן, ולא יתייחס לתשובות נוספות.
3. בנספח ב' לשאלון זה מובא מילון מונחים בשפות עברית, ערבית, אנגלית ורוסית. תוכלו להיעזר בו בעת הצורך.

הוראות למשגיחים:
בתום הבחינה יש לוודא שהנבחנים הדביקו את מדבקת הנבחן שלהם במקום המיועד לכך בדף התשובות שבנספח א', וצירפו אותו למחברת הבחינה.

כתבו **במחברת הבחינה בלבד**, בעמודים נפרדים, כל מה שברצונכם לכתוב **כטיוטה** (ראשי פרקים, חישובים וכדומה).
כתבו "טיוטה" בראש כל עמוד טיוטה. כתיבת טיוטות כלשהן על דפים שמחוץ למחברת הבחינה עלולה לגרום לפסילת הבחינה!

בשאלון זה 42 עמודים ו־3 עמודי נספחים.

בהצלחה!

המשך מעבר לדף

השאלות

<u>פרק ראשון</u> (70 נקודות)

ענו על השאלות 1–2 – שאלות חובה.

שאלה 1 – שאלת חובה (40 נקודות)

בשאלה זו שבעה סעיפים (א׳–ז׳). עליכם לענות על <u>כל</u> הסעיפים.
בסעיפים שבהם יש שאלות רב־ברירה נתונות ארבע תשובות, אך רק אחת מהן נכונה. בכל שאלה העתיקו למחברת את התשובה הנכונה.
בסעיפים האחרים, פעלו על־פי ההנחיות.

רוב הפרויקטים באינטרנט בנויים בשיטת MCV (Model Control View), שמכילה שלושה חלקים:

I. החלק Model מטפל בקשר בין השרת (server) ובין משאבי המערכת (כגון מסדי הנתונים ומדיה).

II. החלק Control מטפל בלוגיקה של הפרויקט (תוכניות).

III. החלק View מטפל בתצוגות (presentation) המוצגות למשתמש (כגון מסכים, הודעות וקלט).

ברצוננו לכתוב תוכנית שתהווה חלק עיקרי של Control של הפרויקט Facebook.

הדרישות:

a. משתמש יוכל ליצור לעצמו פרופיל;

b. משתמש יוכל לצרף חבר לרשימת חבריו;

c. משתמש יוכל להעלות הודעות או תכנים (posts);

d. משתמש יוכל לסמן like להודעה.

בהתאם לדרישות אלו, המערכת תתמוך בפעולות האלה:

a. יצירת פרופיל של משתמש;

b. קליטת חבר של משתמש;

c. העלאת הודעה;

d. סימון like להודעה;

e. הצגת הודעה שהעלה משתמש;

f. הצגת פרופיל של משתמש;

g. הצגת ההודעות ששלח משתמש מסוים;

h. הצגת החברים של משתמש נתון;

i. יציאה מן המערכת.

המשך בעמוד 3

שימו לב:

הנתונים שהתקבלו כבר עברו בדיקות תקינות.

המערכת תבצע, בין היתר, את הפעולות שלהלן:

I. **אתחול המערכת** – init – פעולה זו מאתחלת את המערכת ויוצרת את הטבלאות ואת עץ המשתמשים. התהליך יתואר בהמשך.

II. **יצירת פרופיל של משתמש** – insertTree – פעולה זו מוסיפה משתמש חדש לעץ המשתמשים.

III. **הכנסת חבר של משתמש** – insertFriend – פעולה זו מטפלת בהוספת חבר למשתמש נתון.

IV. **העלאת הודעה** – insertPost – פעולה זו מטפלת בהכנסה למערכת של הודעה שהעלה משתמש מסוים.

V. **סימון like להודעה** – enterLike – הפעולה מטפלת בהכנסת like להודעה מסוימת.

VI. **הצגת הודעה של משתמש** – displayPost – הפעולה מציגה הודעה מסוימת של משתמש נתון.

VII. **הצגת פרופיל של משתמש** – findUser.

VIII. **הצגת כל ההודעות שהעלה משתמש מסוים** – printPostList.

IX. **הצגת החברים של משתמש מסוים** – printFriendList.

המשך בעמוד 4

לפניכם תיאור של מבנה הנתונים התומך במימוש הפעולות הנדרשות מן המערכת הממוחשבת.

נחזיק מבנה (רשומה – head), שעליו מצביע DS, ואת המבנה נכנה בשם **"המבנה הראשי"**, המכיל את השדות האלה:

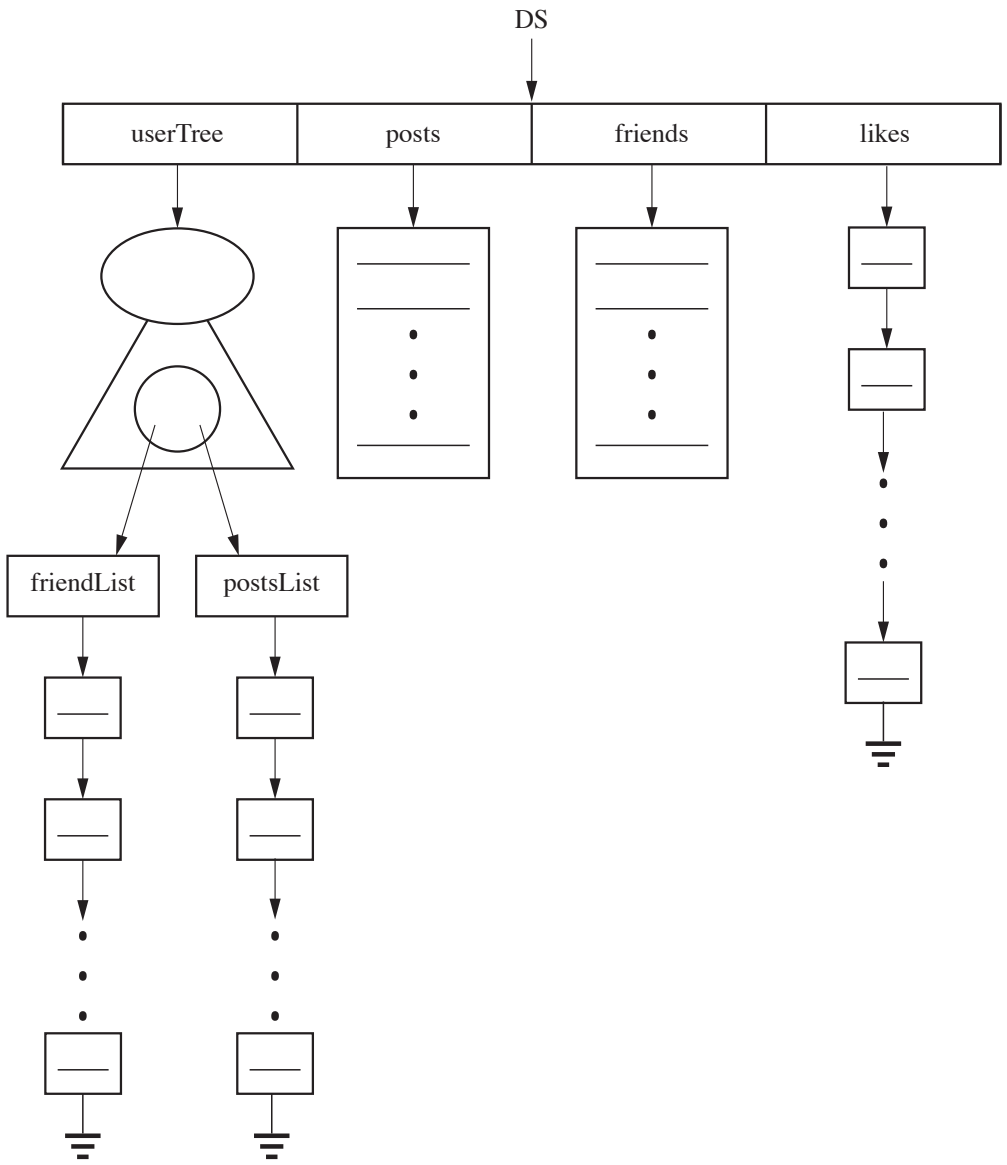
שדה 1 – **userTree** – מצביע על עץ המשתמשים

שדה 2 – **posts** – מצביע למערך דינמי של ההודעות

שדה 3 – **friends** – מצביע למערך הדינמי של החברים

שדה 4 – **likes** – מצביע לרשימה המקושרת של ה־likes

באיור לשאלה 1 מוצג תיאור סכמתי של ה"מבנה הראשי", המאגד את כל מבני הנתונים שבהם נאחסן את נתוני המערכת הממוחשבת:



איור לשאלה 1

המשך בעמוד 5

להלן הגדרות התקפות לכל הסעיפים שיבואו בהמשך (השדות יפורטו בהמשך):

```
typedef enum {FAILURE,SUCCESS,INVALID_INPUT,RECORD_NOT_FOUND,DUPLICATE_ID}
statusType;
typedef enum {FALSE,TRUE} boolean;

typedef struct friendListType // מבנה של צומת ברשימת החברים
 {
   int friendID;
   struct friendListType *next;
 } friendListRec,*friendListPtr;

 typedef struct postListType // מבנה של צומת ברשימת ההודעות
 {
   int postNum;
   struct postListType *next;
 } postListRec,*postListPtr;
```

להלן הגדרת "המבנה הראשי" בשפת C:

```
typedef struct headderType
{
   nodePtr userTree; // מצביע על עץ המשתמשים
   postPtr posts; // מצביע לטבלת ההודעות
   friendPtr friends; // מצביע לטבלת החברים
   likePtr likes; // מצביע לרשימת ה"לייקים"
 } headder, *headPtr;
```

המשך בעמוד 6

עתה נפרט את מבנה הנתונים עבור <u>שדה 1</u> של "המבנה הראשי" – userTree – עץ בינארי של המשתמשים.

להלן מבנה של תא **בעץ הבינארי של המשתמשים** בשפת C:

```
typedef struct nodeType
{
   int userID; // מספר המשתמש
   char firstName[20]; // השם הפרטי של המשתמש
   char lastName[20]; // שם המשפחה של המשתמש
   char eMail[20]; // כתובת דואר אלקטרוני
   int likes; // מספר ה"לייקים"
   struct postListType  *pList; // מצביע לרשימת ההודעות
   struct friendListType *fList; // מצביע לרשימת החברים
   struct nodeType *left;
   struct nodeType *right;
} nodeRec,*nodePtr;
```

עתה נפרט את מבנה הנתונים עבור <u>שדה 2</u> של "המבנה הראשי" – טבלת ההודעות posts.

להלן מבנה של תא **בטבלת ההודעות** בשפת C:

```
typedef struct postType
{
   int userID; //מספר המשתמש
   int postNum; //מספר ההודעה
   char datePosted[20]; // תאריך ההודעה
   char content[200]; // תוכן ההודעה
   int likes; // מספר ה"לייקים"
} postRec ,*postPtr;
```

המשך בעמוד 7

עתה נפרט את מבנה הנתונים עבור שדה 3 של "המבנה הראשי" – טבלת החברים friends.

להלן מבנה של תא **בטבלת החברים** בשפת C:

```
typedef struct friendType
{
   int friendID; //מספר החבר
   int userID; // מספר המשתמש
   char firstName[20]; //השם הפרטי של החבר
   char lastName[20]; // שם המשפחה של החבר
   char eMail[20]; // כתובת הדואר האלקטרוני של החבר
} friendRec,*friendPtr;
```

עתה נפרט את מבנה הנתונים עבור שדה 4 של "המבנה הראשי" – רשימת ה־likes.

להלן מבנה של תא **ברשימה המקושרת של ה־"לייקים"** בשפת C:

```
typedef struct likeType
{
   int postNum; //מספר ההודעה
   int userID; // like מספר המשתמש שסימן
   int senderID; // מספר שולח ה"לייק"
   struct likeType *next;
} likeRec,*likePtr;
```

המשך בעמוד 8

נתונה ספריית פונקציות המכילה, בין היתר, את הפונקציות שלהלן:

`nodePtr insertTree(nodePtr root, int key,char firstName[],char lastName[],char eMail[])`	פונקצייה זו מקבלת את נתוני המשתמש: root – מצביע על עץ המשתמשים key – מספר משתמש firstName – השם הפרטי של המשתמש lastName – שם המשפחה של המשתמש eMail – הדואר האלקטרוני של המשתמש הפונקצייה מוסיפה אותם לעץ בינארי הממוין לפי key הפונקצייה מחזירה מצביע לצומת שנוצר
`nodePtr findUser(nodePtr root, int ID)`	הפונקצייה מקבלת את הנתונים האלה: root – שורש העץ ID – מספר משתמש הפונקצייה מחזירה מצביע לתא בעץ המשתמשים עבור המשתמש המבוקש אם המשתמש לא נמצא – הפונקצייה תחזיר את הערך NULL
`int findFriend(int friendID,int userID)`	הפונקצייה מקבלת את הנתונים האלה: friendID – מספר החבר userID – מספר המשתמש אם נמצא החבר של המשתמש, הפונקצייה מחזירה את מיקומו בטבלת החברים. אחרת – היא מחזירה 1–
`void insertpostList(int userID,int postNum)`	הפונקצייה מקבלת את הנתונים האלה: userID – מספר המשתמש postNum – מספר ההודעה הפונקצייה מוסיפה הודעה לתחילת הרשימה המקושרת של ההודעות של משתמש מסוים
`int findPost(int postNum,int userID)`	הפונקצייה מקבלת את הנתונים האלה: postNum – מספר ההודעה userID – מספר המשתמש הפונקצייה מחזירה את מיקום ההודעה של המשתמש במערך ההודעות אם ההודעה או המשתמש לא נמצאו, מחזירה 1–

הניחו שהפונקציות האלה כתובות וניתן להשתמש בהן בכל הסעיפים הבאים בלי לכתוב אותן מחדש. כמו כן, בעבור כל סעיף תוכלו להשתמש בכל פונקצייה שמומשה בסעיפים שלפניו.

המשך בעמוד 9

להלן הגדרות של המשתנים הגלובליים בתוכנית:

```
/******************
*   G l o b a l s *
******************/
headder head; // המבנה הראשי
headPtr DS = &head; //מצביע למבנה הראשי
int lastFriend = 0; //המיקום הפנוי במערך החברים
int lastPost = 0; // המיקום הפנוי במערך ההודעות
```

א. **(3 נק')** להלן פונקצייה שכותרתה: void **insertfriendlist**(int userID,int friendID)

הפונקצייה מקבלת את הפרמטרים האלה:

userID – מספר משתמש

friendID – מספר חבר

הפונקצייה מוסיפה צומת לתחילת הרשימה המבוקשת של החברים של משתמש מסוים.

בפונקצייה חסרים **שני** ביטויים, המסומנים במספרים בין סוגריים עגולים. כִתבו במחברת את מספרי הביטויים החסרים (1)–(2) , בסדר עולה, וכִתבו ליד כל מספר את הביטוי החסר שהוא מייצג.

```
void insertfriendlist(int userID,int friendID)
{
   static nodePtr p;
   friendListPtr t;
   p =__(1)__;
   t = malloc(sizeof(friendListRec));
   t ->friendID = friendID;
   t->next =  __(2)__;
   __(2)__= t;
}
```

המשך בעמוד 10

ב. (7 נק') להלן פונקצייה שכותרתה:

statusType **insertFriend** (int friendID ,int userID, char firstName [20], char lastName[20], char eMail [20])

פונקצייה זו מקבלת את הפרמטרים האלה:

friendID – מספר החבר

userID – מספר המשתמש

firstName – השם הפרטי של החבר

lastName – שם המשפחה של החבר

eMail – כתובת הדואר האלקטרוני של החבר

הפונקצייה מוסיפה חבר של משתמש למערך החברים וגם לרשימת החברים של המשתמש.

הפונקצייה מחזירה ערך מטיפוס statusType, כמפורט בטבלה שלהלן:

INVALID_INPUT	אם המשתמש או החבר לא נמצאו במערכת או שהחבר כבר רשום בטבלה
SUCCESS	אם הפעולה הסתיימה בהצלחה

בפונקצייה חסרים **שישה** ביטויים, המסומנים במספרים בין סוגריים עגולים. כתבו במחברת את הביטויים החסרים (1)–(6) , בסדר עולה, וכתבו ליד כל מספר את הביטוי החסר שהוא מייצג.

```
statusType insertFriend(int friendID ,int userID, char firstName[20],
char lastName[20], char eMail[20])
{
     statusType status=SUCCESS;
     int index;
     nodePtr root;
     root =___(1)___;
     index =__(2)___;
     if ((root == NULL)||( index !=-1)) status = INVALID_INPUT;
     else
```

המשך בעמוד 11

```
    {
        if (lastFriend == 0)
        {
           DS->friends =__(3)__;
        }
        else
        {
           DS->friends =__(4)__;
        }
        DS->friends[lastFriend].friendID = friendID;
        DS->friends[lastFriend].userID = userID;
        strcpy(DS->friends[lastFriend].firstName,firstName);
        strcpy(DS->friends[lastFriend].lastName,lastName);
        strcpy(DS->friends[lastFriend].eMail,eMail);
        __(5)__;
        __(6)__;
    }
    return status ;
}
```

ג. (2 נק') מהי סיבוכיות זמן הריצה של הפונקצייה המוצגת בסעיף ב', אם ידוע ש־n מציין את מספר המשתמשים ומספר החברים?

1. $O(\log n)$
2. $O(1)$
3. $O(n^2)$
4. $O(n)$

המשך בעמוד 12

ד. (8 נק') להלן פונקצייה שכותרתה:

statusType **insertPost**(int userID,int postNum,char content[200])

הפונקצייה מקבלת את הפרמטרים האלה:

userID – מספר המשתמש

postNum – מספר ההודעה

Content – תוכן ההודעה

הפונקצייה יוצרת הודעה חדשה למשתמש, שבה נוסף התאריך העכשווי (מתוך המחשב). הפונקצייה מוסיפה הודעה של המשתמש למערך ההודעות ולרשימת ההודעות של המשתמש.

הפונקצייה מחזירה ערך מטיפוס statusType, כמפורט בטבלה שלהלן:

INVALID_INPUT	אם המשתמש לא נמצא במערכת או שההודעה כבר קיימת במערכת
SUCCESS	אם הפעולה הסתיימה בהצלחה

בפונקצייה חסרים **שישה** ביטויים, המסומנים במספרים בין סוגריים עגולים. כתבו במחברת את הביטויים החסרים (1)–(6) , בסדר עולה, וכתבו ליד כל מספר את הביטוי החסר שהוא מייצג.

המשך בעמוד 13

```
statusType insertPost(int userID,int postNum,char content[200])
{
    int i;
    statusType status=SUCCESS;
    time_t current_time;
    char* c_time_string;
    nodePtr root;
    current_time = time(NULL); // קבלת התאריך הנוכחי
    i = findPost (postNum, userID);
    c_time_string = ctime(&current_time); // הפיכת התאריך למחרוזת
    root =___(1)___;
    if ((root == NULL) || (i!= -1) status = INVALID_INPUT;
    else
    {
          if ___(2)___
          {
              DS->posts = __(3)___;
          }
          else
          {
              DS->posts = ___(4)___;
          }
              DS->posts[lastPost].userID = userID;
              DS->posts[lastPost].postNum = postNum;
              strcpy(DS->posts[lastPost].datePosted,c_time_string);
              strcpy(DS->posts[lastPost].content,content);
              DS->posts[lastPost].likes = 0;
             ___(5)___;
             ___(6)___;
     }
     return status ;
}
```

המשך בעמוד 14

ה. **(8 נק')** להלן פונקצייה שכותרתה: statusType **insertLike**(likePtr *list,int userID,int postNum,int senderID)

הפונקצייה מקבלת את הפרמטרים האלה:

list – מצביע לרשימה מקושרת של "לייקים"

userID – מספר המשתמש שהעלה את ההודעה

postNum – מספר ההודעה

SenderID - מספר המשתמש ששלח את ה"לייק"

הפונקצייה מוסיפה צומת לרשימת ה"לייקים".

הפונקצייה מחזירה את הערך statusType, כמפורט בטבלה שלהלן:

DUPLICATE_ID	אם סימן ה־like ששלח משתמש מסוים להודעה שהעלה משתמש אחר כבר קיים במערכת
SUCCESS	אם הפונקצייה הסתיימה בהצלחה

בפונקצייה חסרים **שישה** ביטויים, המסומנים במספרים בין סוגריים עגולים. כְתבו במחברת את הביטויים החסרים (1)–(6) , בסדר עולה, וכְתבו ליד כל מספר את הביטוי החסר שהוא מייצג.

המשך בעמוד 15

```
statusType insertLike(likePtr *list,int userID,int postNum,int senderID)
 {
          statusType status = SUCCESS;
          likePtr p,q ;
          p = *list;
          while(p)
          {
                   if  (___(1)___)
                   {
                      status = DUPLICATE_ID;
                      break;
                   }
                   p = ___(2)___;
          }
          if(status==SUCCESS)
          {
                   ___(3)___;
                   q->userID = userID;
                   q->postNum = postNum;
                   ___(4)___;
                   q->next = ___(5)___;
                   ___(6)___;
          }
          return status;
}
```

המשך בעמוד 16

ו. **(5 נק')** להלן פונקצייה שכותרתה: statusType **enterLike**(int postNum,int userID,int senderID)

הפונקצייה מקבלת את הפרמטרים האלה:

postNum – מספר ההודעה

userID – מספר המשתמש שהעלה את ההודעה

SenderID – מספר המשתמש ששלח את ה"לייק"

הפונקצייה בודקת אם ההודעה והמשתמש קיימים במערכת. אם הם אינם קיימים – הפונקצייה תחזיר את ההודעה RECORD_NOT_FOUND. אם הם קיימים – הפונקצייה תבדוק אם ה"לייק" ששלח המשתמש כבר קיים. אם כן, תחזיר DUPLICATE_ID. אחרת – תעדכן את מספר ה"לייקים" עבור ההודעה ואת מספר ה"לייקים" של המשתמש שהעלה את ההודעה.

הפונקצייה תחזיר את הערך statusType, כמפורט בטבלה שלהלן:

RECORD_NOT_FOUND	אם ההודעה או משתמש לא קיימים במערכת
DUPLICATE_ID	אם ה־like כבר קיים
SUCCESS	אם הפעולה בוצעה בהצלחה

בפונקצייה חסרים **ארבעה** ביטויים המסומנים במספרים בין סוגריים עגולים. כתבו במחברת את הביטויים החסרים (1)–(4) , בסדר עולה, וכתבו ליד כל מספר את הביטוי החסר שהוא מייצג.

המשך בעמוד 17

```
statusType enterLike(int postNum,int userID)
{
        static nodePtr p;
        statusType status=SUCCESS;
        int i;
        p = ___(1)___;
        i = ___(2)___;
        if((p == NULL) ||( i == -1))
                status = RECORD_NOT_FOUND;
        else
           {
                status = insertLike ___(3)___;
                if(status==SUCCESS)
                {
                        ___(4)___;
                        DS->posts[i].likes++;
                }
           }
        return status;
}
```

המשך בעמוד 18

ז. **(7 נק')** להלן פונקצייה שכותרתה: statusType **printFriendList**(int userID)

הפונקצייה מקבלת את הפרמטר userID – מספר המשתמש, ומדפיסה את שמות כל החברים של המשתמש.

הפונקצייה מחזירה ערך מטיפוס statusType, כמפורט בטבלה שלהלן:

RECORD_NOT_FOUND	אם המשתמש לא קיים במערכת
SUCCESS	אם הפעולה בוצעה בהצלחה

בפונקצייה חסרים **חמישה** ביטויים, המסומנים במספרים בין סוגריים עגולים. כְתבו במחברת את הביטויים החסרים (1)–(5) , בסדר עולה, וכְתבו ליד כל מספר את הביטוי החסר שהוא מייצג.

```
statusType printFriendList(int userID)
{
     statusType status = SUCCESS;
     int f;
     nodePtr p;
     friendListPtr q;
     p = ___(1)___;
     if(p==NULL) status = RECORD_NOT_FOUND;
     else
     {
          q = ___(2)___;
          while(q)
          {
               f = ___(3)___;
               printf("\n%s \t%s", ___(4)___);
               ___(5)___;
          }
     }
     return status;
}
```

המשך בעמוד 19

שאלה 2 – שאלת חובה (30 נקודות)

עליכם לענות על <u>שניים</u> מבין הסעיפים א'–ג' (לכל סעיף – 15 נקודות).
בסעיפים שבהם יש שאלות רב־ברירה נתונות ארבע תשובות, אך רק אחת מהן נכונה. בכל שאלה העתיקו למחברת את התשובה הנכונה.
בסעיפים האחרים, פעלו על־פי ההנחיות.

(15 נק') **א.** נתונה מטריצה arr [R][C] שבה כל תא מכיל את הערכים 0 **או** 1 **או** 2 , כאשר:

0 – מסמן תא ריק

1 – מסמן תפוז טרי

2 – מסמן תפוז רקוב

לדוגמה:

1 2 0 1 2

1 1 2 0 1

1 0 0 2 1

כל תפוז רקוב בתא arr [i][j] יגרום למוחרת להרקבת כל התפוזים הטריים בתאים השכנים (השכן מלמעלה, השכן מלמטה, השכן מימין והשכן משמאל – אם הם קיימים).

הבעיה: מהו המספר המינימלי של בדיקות שיש לעשות כדי לוודא **שכלל התפוזים נרקבו** או **שלא נותרו תפוזים טריים שגובלים בתפוזים רקובים (כלומר, שלא יירקבו יותר תפוזים)**.

הפתרון הפשוט:

סורקים בכל יום את כל התאים עד שאחד משני המקרים לעיל מתגלה.

הפתרון היעיל יותר:

משתמשים בתור על־פי האלגוריתם הזה:

a. צור תור ריק Q.

b. סרוק את המטריצה כולה (משמאל לימין, למעלה ולמטה), והכנס לתור את המיקום של כל תפוז רקוב.

c. כל זמן שהתור לא ריק, בצע:

c.1 שלוף תא של תפוז מהתור.

c.2 בדוק את שכניו של התא – אם בתא השכן יש תפוז טרי, הפוך אותו לרקוב, והכנס את מקומו לתור.

d. אם נשארו תפוזים טריים החזר 1– , אחרת – החזר את מספר התפוזים הרקובים.

המשך בעמוד 20

(2 נק') 1. מהי סיבוכיות זמן הריצה במקרה זה?

I. $O(\log(R \times C))$

II. $O(R \times C)$

III. $O((R \times C)^2)$

IV. $O(R^2 \times C^2)$

להלן הגדרות של המבנים שבהם התוכנית משתמשת:

```
#define R 3
#define C 5

typedef enum {FALSE,TRUE} boolean;

typedef struct cellType //מבנה של מיקום של תא במטריצה
{
      int x;       //שורה
      int y;       //עמודה
}cellRec ,*cellPtr;

typedef struct queueType //מבנה תא בתור
{
   cellRec cell ;
   struct queueType *next;
}queueRec ,*queuePtr;

queuePtr Q = NULL;
```

המשך בעמוד 21

נתונה ספריית פונקציות המכילה, בין היתר, את הפונקציות שלהלן:

`void insertQ(queuePtr *que,cellRec cell)`	הפונקצייה מקבלת תור que ותא cell ומכניסה לתור את התא cell.
`cellRec deleteQ(queuePtr *que)`	הפונקצייה מקבלת תור que ומוציאה תא מן התור ומחזירה אותו.
`boolean isEmpty(queuePtr que)`	הפונקצייה מקבלת תור que ומחזירה TRUE אם התור ריק, אחרת – היא מחזירה FALSE.
`boolean checkall(int arr[][C])`	הפונקצייה מקבלת מטריצה arr, ומחזירה TRUE אם נשאר לפחות תפוז אחד טרי, אחרת – היא מחזירה FALSE.
`boolean isvalid(int i, int j)`	הפונקצייה מקבלת מיקום i ,j במטריצה, ובודקת שאינו חורג מתחום המטריצה. אם אינו חורג מתחום המטריצה, היא מחזירה TRUE, אחרת – היא מחזירה FALSE.

2. לפניכם הפונקצייה: int **rotOranges**(int arr[][C]) (13 נק')

הפונקצייה מקבלת מטריצת תפוזים, ובהתאם לאלגוריתם, מחזירה 1– אם נשארו תפוזים טריים; אחרת – הפונקצייה מחזירה את מספר התפוזים הרקובים בזמן הבדיקה (המשתנה ans).

בפונקצייה חסרים שבעה ביטויים המסומנים במספרים בין סוגריים עגולים. כתבו במחברת את הביטויים החסרים (1)–(7) , בסדר עולה, וכתבו ליד כל מספר את הביטוי החסר שהוא מייצג.

```
int rotOranges(int arr[][C])
{
    cellRec  temp;
    int ans = 0;
    int i,j;
    for (i=0; i<R; i++)
    {
       for (j=0; j<C; j++)
       {
            if (arr[i][j] == 2)
            {
                temp.x = i;
                temp.y = j;
                ans++;
```

המשך בעמוד 22

```
                    ____(1)____;
                }
            }
        }
        while(!isEmpty(Q))
           {
                temp = ____(2)____ ;

                if (isvalid(temp.x+1, temp.y) && ____(3)____)
                {
                    ____(4)____;
                    ans++;
                    temp.x++;
                    ____(5)____;

                    temp.x--;
                }
                if (insvalid(temp.x-1, temp.y) && ____(6)____)
                {
                    ans++;
                    arr[temp.x-1][temp.y] = 2;
                    temp.x--;
                    ____(5)____);
                    temp.x++;
                }
                if (isvalid(temp.x, temp.y+1) && arr[temp.x][temp.y+1] == 1)
                {
                    ans++;
                    arr[temp.x][temp.y+1] = 2;
                    temp.y++;
                    ____(5)____;
                    temp.y--;
                }
                if (isvalid(temp.x, temp.y-1) && arr[temp.x][temp.y-1] == 1)
                {
```

המשך בעמוד 23

```
                ans++;
                arr[temp.x][temp.y-1] = 2;
                temp.y--;
                ____(5)____;
            }
        }
        if (____(7)____)
            return -1;
        else
            return ans;
}
```

המשך בעמוד 24

ב. **(15 נק')** נתון מערך arr[n] של N מספרים טבעיים. לכל איבר arr[i] במערך יש למצוא את האיבר הראשון שאחריו arr[j], כך ש- arr[i] < arr[j] עבור $j = i + 1 \ldots n-1$.

הפתרון הפשוט: לכל איבר arr[i] במערך בודקים את האיברים שאחריו. האיבר הראשון שגדול ממנו הוא האיבר המבוקש.

1. (2 נק') מהי סיבוכיות זמן הריצה במקרה זה?

   I. $O(n)$

   II. $O(\log n)$

   III. $O(n^2)$

   IV. $O(n \log n)$

2. (13 נק') ניתן לכתוב פתרון יעיל יותר על-ידי שימוש **במחסנית**. שימו לב, על-ידי פתרון זה הפונקצייה תדפיס לכל איבר במערך את המספר הראשון אחריו שגדול ממנו.

   אם אין מספר גדול ממנו הפונקצייה תדפיס 1- .

   בפתרון זה סדר הדפסת האיברים אינו בהכרח הסדר שבו הם מופיעים במערך עצמו.

   הרעיון הכללי הוא:

   לכל איבר במערך arr[i] בודקים את האיבר **העוקב** במערך.

   אם האיבר העוקב קטן מ- arr[i], מכניסים את arr[i] ואת האיבר העוקב למחסנית.

   אם האיבר העוקב גדול מ- arr[i], מדפיסים את שניהם לפי הסדר, שולפים איבר מן המחסנית, ומשווים מחדש את האיבר שהוצאנו עם האיבר העוקב לו.

   אם נשארים בסוף איברים במחסנית, שולפים אותם ומדפיסים אותם בצירוף 1- (כלומר לא נמצא איבר גדול ממנו).

   להלן הפונקצייה: void **printNextGreater**(int arr[], int n)

   הפונקצייה מקבלת מערך וגודלו (כלומר: מספר האיברים), ומדפיסה לכל איבר את האיבר עצמו ואת האיבר הראשון אחריו שגדול ממנו, או 1- אם לא קיים איבר כזה.

   הפונקצייה משתמשת במחסנית S ובפונקציות הנלוות לה:

```
stackPtr S; // הגדרת מחסנית
void push(stackPtr *S, int x);
int pop (stackPtr *S);
Boolean isEmpty(stackPtr S);
```

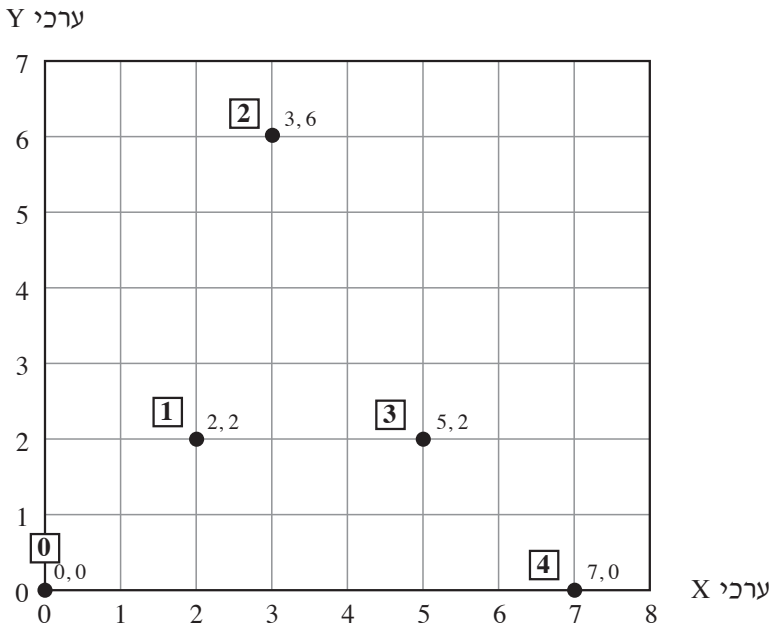
המשך בעמוד 25

בפונקצייה חסרים **חמישה** ביטויים המסומנים במספרים בין סוגריים עגולים. כְתבו במחברת את הביטויים החסרים (1)–(5), בסדר עולה, וכְתבו ליד כל מספר את הביטוי החסר שהוא מייצג.

```
void printNextGreater(int arr[], int n)
{
      int i = 0;
      int next,element;
      push(&S, arr[0]);
      for (i=1; i<n; i++)
      {
           next = arr[i];
           if (!isEmpty(S))
           {
               element = ____(1)____;
               while (____(2)____)
               {
                    printf("\n %d -> %d", element, next);
                    if(isEmpty(S))
                      break;
                      ____(3)____;
               }
               if(element >=next)
               ____(4)____;
           }
           ____(5)____;
      }
      while(!isEmpty(S))
      {
               element = pop(&S);
               next = -1;
               printf("\n %d -> %d", element, next);
      }
}
```

המשך בעמוד 26

ג. **(15 נק')** נתונות חמש ערים, p0…p4, הממוקמות בקואורדינטות: (7,0), (5,2), (3,6), (2,2), (0,0).



יש לסלול כבישים בין הערים כך שאפשר יהיה להגיע מכל עיר לכל אחת מן הערים האחרות.

עלות הסלילה של כביש בין עיר i לעיר j מחושבת לפי הנוסחה:

cost (i,j)=abs(p[i].x-p[j].x) + abs(p[i].y-p[j].y) כאשר: x , y הם קואורדינטות של הערים האלה.

לדוגמה: עלות הסלילה של כביש מעיר 1 לעיר 2 (לפי הסרטוט) תחושב באופן הזה: cost(1,2)=|2-3|+|2-6|=5

יש להציג את הכבישים שיש לסלול כדי שעלות הסלילה תהיה מינימלית.

ניתן לפתור את הבעיה בעזרת **גרף ומטריצת צמידות**: cost[N][N] שבה עלות סלילת הכביש מ־i ל־j תהיה cost[i][j] ולבנות את מערך הפריסה המינימלי.

מהלך התוכנית:

1. יצירת מטריצת הצמידות והכנסת הכבישים **לתור עם עדיפויות**.
2. כל זמן שהתור לא ריק:
   2.1 הוצא כביש מתור העדיפויות.
   2.2 בדוק אם הכביש אינו יוצר מעגל במערך הפריסה המינימלי (מעגל נוצר כאשר העיר ההתחלתית של שתי ערים היא אותה עיר).
   2.3 אם הכביש אינו יוצר מעגל, צרף אותו **למערך הפריסה המינימלי**.

המשך בעמוד 27

להלן הגדרת מבני הנתונים שבהם התוכנית משתמשת:

```
#define N 5 //מספר הערים
#define NIL -1

typedef struct pointType
{
      int x; // קואורדינטות x של עיר
      int y; // קואורדינטות y של עיר
}point,*pointPtr;

struct edge //הגדרת כביש
{
        int u; //u מעיר
        int v; //v לעיר
        int weight; //מחיר סלילת הכביש
        struct edge *link;
}*front = NULL;
```

נתונה ספריית פונקציות המכילה, בין היתר, את הפונקציות לטיפול בתור העדיפויות כמפורט להלן:

void **insert_pque**(int i,int j,int wt)	הפונקצייה מקבלת את מספרי הערים i ו- j, ואת wt, שהוא עלות הכביש שמחבר ביניהן, ומכניסה את הכביש לתור העדיפויות.
struct edge ***del_pque**()	הפונקצייה מוציאה כביש מתור העדיפויות ומחזירה אותו.
int **isEmpty_pque**( )	הפונקצייה מחזירה 1 אם התור ריק, אחרת – היא מחזירה 0 .
int **dist**(point a,point b)	הפונקצייה מקבלת את הקואורדינטות של הערים a ו- b, ומחזירה את עלות הסלילה של הכביש בין a ל-b.

המשך בעמוד 28

להלן המשתנים הגלובליים של התוכנית:

```
point points[N] = {{0,0},{2,2},{3,6},{5,2},{7,0}};
int cost[N][N];
struct edge tree [N]; // מערך פריסת הכבישים
```

1. לפניכם פונקצייה שכותרתה: void **create_graph**() (6 נק')

הפונקצייה בונה את **מטריצת הצמידות** ואת **תור העדיפויות**.

בפונקצייה חסרים **שני** ביטויים, המסומנים במספרים בין סוגריים עגולים. כִתבו במחברת את הביטויים החסרים (1)–(2) , בסדר עולה, וכִתבו ליד כל מספר את הביטוי החסר שהוא מייצג.

```
void create_graph()
{
        int i,j;
        printf(" \nAdjacent Matrix\n");
        for (i= 0;i<N;i++)
        {
              for (j=0; j< i; j++)
              {
                    cost[i][j] = _____(1)_____;
                    _____(2)_____;
              {
        }
}
```

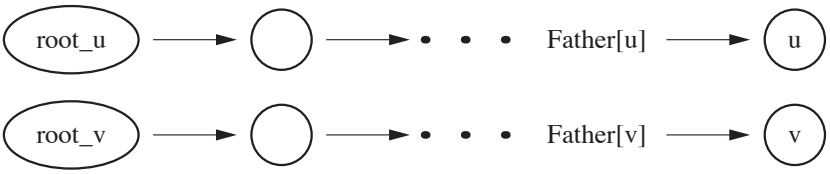
2. לפניכם פונקצייה שכותרתה: void **make_tree**(struct edge tree[]) (9 נק')

הפונקצייה מקבלת מערך (tree) שמייצג את מערך הפריסה המינימלי שלתוכו מוכנסים הכבישים.

המשך בעמוד 29

הפונקצייה מוציאה כביש **מתור העדיפויות**, ובודקת אם ניתן לצרפו למערך tree.

הפונקצייה משחזרת את המסלולים לשני קצותיו, u ו־v, של הכביש, כפי שמוצג באיור שלפניכם:



כאשר:

Father[u] מסמן את העיר הקודמת לעיר u.

root_u מסמן את העיר שממנה מתחיל המסלול ל־u.

אם המסלול ל־u אינו מתחיל כמו המסלול ל־v, מוסיפים למערך הפריסה את הכביש (u,v).

בפונקצייה חסרים **ארבעה** ביטויים, המסומנים במספרים בין סוגריים עגולים. כִתבו במחברת את הביטויים החסרים (1)–(4), בסדר עולה, וכִתבו ליד כל מספר את הביטוי החסר שהוא מייצג.

```
void make_tree(struct edge tree[])
{
        struct edge *tmp;
        int v1,v2,root_v1,root_v2;
        int father[N];
        int i;
        int count = 0; // מספר הכבישים

        for(i=0; i<N; i++)
                father[i] = NIL;

          while( !isEmpty_pque( ) && count < N-1 )
               {
                      ____(1)____;
                      v1 = tmp->u;
                      v2 = tmp->v;
```

המשך בעמוד 30

```
                while( v1 !=NIL )
                {
                        root_v1 = v1;
                        v1 = father[v1];
                }
                while( v2 != NIL  )
                {
                        root_v2 = v2;
                        v2 = father[v2];
                }

                if (____(2)____)
                {
                        tree[count].u = tmp->u;
                        tree[count].v = tmp->v;
                        tree[count].weight = tmp->weight;
                        father[root_v2]=root_v1;
                       ____(3)____;
                }
            }

        if (____(4)____)
        {
             printf("\nGraph is not connected, no spanning tree possible\n");
             exit(1);
        }
}
```

המשך בעמוד 31

פרק שני (30 נקודות)

ענו על אחת מבין השאלות 3–4 (לכל שאלה – 30 נקודות).

שאלה 3 (30 נקודות)

בשאלה זו שמונה סעיפים. ענו על כל הסעיפים. בכל סעיף נתונות ארבע תשובות, אך רק אחת מהן נכונה. בכל סעיף בחרו את התשובה הנכונה והקיפו בעיגול את הספרה המייצגת אותה בדף התשובות שבנספח א'.

(3 נק') **א.** נתון הקטע הזה:

```
int i, j, k = 0;
for (i = n / 2; i <= n; i++)
{
    for (j = 2; j <= n; j = j * 2)
    {
        k = k + n / 2;
    }
}
```

מהי סיבוכיות זמן הריצה של הקטע הזה?

1. $O(n)$
2. $O(n \log n)$
3. $O(n^2)$
4. $O(n^2 \log n)$

המשך בעמוד 32

ב. **(4 נק')** נתונה הפונקצייה הזו:

```
void function(int n)
{
   int i,j,k;
    int count = 0;
    for (i=n/2; i<=n; i++)
        for (j=1; j+n/2<=n; j++)
            for (k=1; k<=n; k = k * 2)
                count++;
}
```

מהי סיבוכיות זמן הריצה של הפונקצייה הזו?

1. $O(n^3)$
2. $0(n \log n)$
3. $O(n^2)$
4. $O(n^2 \log n)$

ג. **(3 נק')** $T(n) = 16T(n/4) + n$ היא פונקציית זמן הריצה של אלגוריתם מסוים הפועל על קלט שגודלו n.
מהי סיבוכיות זמן הריצה של האלגוריתם?

1. $\Theta(n)$
2. $\Theta(n^2 \log n)$
3. $\Theta(n \log n)$
4. $\Theta(n^2)$

ד. **(4 נק')** $T(n) = T(n/2) + 2^n$ היא פונקציית זמן הריצה של אלגוריתם מסוים הפועל על קלט שגודלו n.
מהי סיבוכיות זמן הריצה של האלגוריתם?

1. $\Theta(2^n)$
2. $\Theta(n^2 \log n)$
3. $\Theta(2^n \log n)$
4. $\Theta(n^2)$

המשך בעמוד 33

(4 נק') **ה.** נתונה פונקצייה **רקורסיבית** שמקבלת מצביע לתחילת הרשימה המקושרת הזו:

```
1->2->3->4->5->6
```

להלן מבנה של צומת ברשימה:

```
struct node
{
    int data;
    struct node* next;
};
void fun(struct node* start)
{
    if (start == NULL)
        return;
    printf("%d ", start->data);
    if (start->next != NULL)
        fun(start->next->next);
    printf("%d ", start->data);
}
```

מה תדפיס הפונקצייה fun?

1. 1 4 6 6 4 1
2. 1 3 5 1 3 5
3. 1 2 3 4
4. 1 3 5 5 3 1

המשך בעמוד 34

ו. **(4 נק')** לפניכם שלוש סריקות של עץ חיפוש בינארי (לא ממוין), ולא ידוע אילו מהן היא סריקה תקינה של אותו העץ.

I. M B C A F H P Y K

II. K A M C B Y P F H

III. M A B C K Y F P H

על־פי שיטות הסריקה preorder, inorder ו־postorder, מהו ההיגד הנכון?

1. סריקות I ו־II הן סריקות preorder ו־inorder, בהתאמה
2. סריקות I ו־III הן סריקות preorder ו־postorder, בהתאמה
3. סריקות II ו־III הן סריקות preorder ו־inorder, בהתאמה
4. סריקה II היא סריקת inorder, ולא ניתן לקבוע מהן הסריקות I ו־III

ז. **(4 נק')** לפניכם המערך: 20,18,15,13,12 , המייצג ערימת מקסימום.

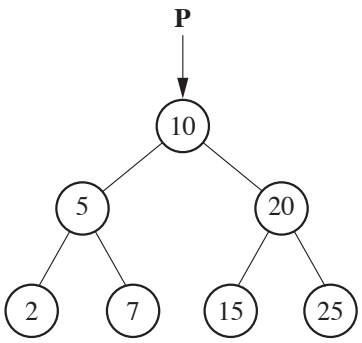
רוצים להוסיף לערימה את האיברים 10 ו־25 (לפי הסדר). הניחו שהמערך מספיק גדול.

מה יהיה סדר האיברים במערך אחרי הוספת שני האיברים?

1. 25,20,18,13,10,12,15
2. 25,18,20,13,12,10,15
3. 25,18,13,20,12,10,15
4. 25,18,20,15,12,13,10

המשך בעמוד 35

(4 נק') **ח.** נתון העץ הזה:



להלן המבנה של צומת בעץ:

```
typedef struct nodeType
{
    int data;
    struct nodeType *left;
    struct nodeType *right;
    struct nodeType *nextRight;
}node, *nodePtr;
```

המשך בעמוד 36

הניחו שערך המצביע nextRight בכל הצמתים הוא NULL.

לפניכם פונקצייה **רקורסיבית** המקבלת מצביע p לשורש העץ.

בצעו מעקב אחרי הפונקצייה על העץ המתואר לעיל.

```
void recur(nodePtr p)
{
    if (!p)
        return;
    if (p->left)
        p->left->nextRight = p->right;
    if(p->right)
         if(p->nextRight)
                p->right->nextRight  = p->nextRight->left;
         else
                p->right->nextRight = NULL;
    recur(p->left);
    recur(p->right);
}
```

מה מבצעת הפונקצייה?

1. קושרת את הצמתים הנמצאים באותה רמה
2. קושרת את כל עלי העץ
3. קושרת כל צומת לאביו עד השורש
4. הופכת בין הבן הימני לבן השמאלי

המשך בעמוד 37

שאלה 4 (30 נקודות)

בשאלה זו שמונה סעיפים. ענו על כל הסעיפים. בכל סעיף נתונות ארבע תשובות, אך רק אחת מהן נכונה. בכל סעיף בחרו את התשובה הנכונה והקיפו בעיגול את הספרה המייצגת אותה בדף התשובות שבנספח א'.

(3 נק') **א.** נתונה הפונקצייה הזו:

```
void function(int n)
{
    int i, j, k;
    int count = 0;
    for (i=n/2; i<=n; i++)
        for (j=1; j<=n; j = 2 * j)
            for (k=1; k<=n; k = k * 2)
                count++;
}
```

מהי סיבוכיות זמן הריצה של הפונקצייה?

1. $O(n \log n)$
2. $O(n^2 \log n)$
3. $O(n(\log n)^2)$
4. $O(n^3)$

(4 נק') **ב.** נתונה הפונקצייה הרקורסיבית הזו:

```
void silly(int(n))
{
   if (n<=0) return;
    printf("n = %d",++n);
    silly(n-2);
}
```

מהי סיבוכיות זמן הריצה של הפונקצייה?

1. $O(\log n)$
2. $O(n)$
3. $O(n \log n)$
4. $O(n^2)$

המשך בעמוד 38

ג. **(3 נק')** $T(n) = 3T(n/3) + n/2$ היא פונקציית זמן הריצה של אלגוריתם מסוים הפועל על קלט שגודלו n.
מהי סיבוכיות זמן הריצה של האלגוריתם?

1. $\Theta(n^2 \log n)$
2. $\Theta(n^2)$
3. $\Theta(n \log n)$
4. $\Theta(\log n)$

ד. **(4 נק')** $T(n) = 7T(n/3) + n^2$ היא פונקציית זמן הריצה של אלגוריתם מסוים הפועל על קלט שגודלו n.
מהי סיבוכיות זמן הריצה של האלגוריתם?

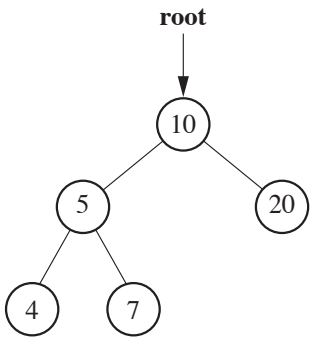
1. $\Theta(n^2)$
2. $\Theta(n \log n)$
3. $\Theta(n^2 \log n)$
4. $\Theta(n)$

ה. **(4 נק')** לפניכם ארבעה מערכים של מספרים טבעיים. איזה מהם מייצג ערימה?

1. 23 , 17 , 14 , 6 , 13 , 10 , 1 , 12 , 7 , 5
2. 23 , 17 , 14 , 6 , 13 , 10 , 1 , 5 , 7 , 12
3. 23 , 17 , 14 , 7 , 13 , 10 , 1 , 5 , 6 , 12
4. 23 , 17 , 14 , 7 , 13 , 10 , 1 , 12 , 5 , 7

המשך בעמוד 39

(4 נק') **ו.** נתון העץ הזה:



להלן מבנה של צומת בעץ:

```
struct node
{
    int data;
    struct node* left;
    struct node* right;
};
```

כמו כן, נתונים:

התור queue

פונקציית createQueue – יצירת תור ריק

פונקציית enQueue – הכנסת איבר לתור

פונקציית deQueue – הוצאת איבר מן התור

המשך בעמוד 40

לפניכם השגרה הזו:

```
void P(struct node* root)
{
    int rear, front;
    struct node** queue = createQueue(&front, &rear);
    struct node* temp_node = root;

    while (temp_node)
    {
        printf("%d ", temp_node->data);

        if (temp_node->left)
            enQueue(queue, &rear, temp_node->left);

        if (temp_node->right)
            enQueue(queue, &rear, temp_node->right);

        temp_node = deQueue(queue, &front);
    }
}
```

בצעו מעקב אחרי השגרה P עם העץ הזה.

מה מדפיסה השגרה?

1. 4 5 7 10 20
2. 10 5 4 7 20
3. 4 7 5 20 10
4. 10 5 20 4 7

המשך בעמוד 41

(4 נק') **ז.** לפניכם מבנה נתונים של רשימה מקושרת:

```
struct node
{
  int value;
  struct node *next;
};
```

להלן פונקצייה שמקבלת את הרשימה המקושרת הזו: 1->2->3->4->5->6->7

```
void rearange(struct node *list)
{
  struct node *p, * q;
  int temp;
  if ((!list) || !list->next)
      return;
  p = list;
  q = list->next;
  while(q)
  {
     temp = p->value;
     p->value = q->value;
     q->value = temp;
     p = q->next;
     q = p?p->next:0;
  }
}
```

בצעו מעקב אחר הפונקצייה.

איך תיראה הרשימה המקושרת לאחר ביצוע הפונקצייה?

1. 1->2->3->4->5->6->7
2. 2->1->4->3->6->5->7
3. 1->3->2->5->4->7->6
4. 2->3->4->5->6->7->1

המשך בעמוד 42

ח. (4 נק') נתונים שלושה מספרים שונים. כמה עצי חיפוש בינארי שונים זה מזה ניתן לבנות מהם?

1. שלושה עצים
2. ארבעה עצים
3. חמישה עצים
4. שישה עצים

בהצלחה!

זכות היוצרים שמורה למדינת ישראל.
אין להעתיק או לפרסם אלא ברשות משרד החינוך.

מקום למדבקת נבחן

נספח א': דף תשובות לפרק השני – שאלות 3 ו־4,
לשאלון 714911, אביב תשפ"ב

הקיפו בעיגול את הספרה המייצגת את התשובה הנכונה לכל סעיף.

הדביקו את מדבקת הנבחן במקום המיועד לכך, והדקו את הדף הזה למחברת הבחינה.

שאלה 3

סעיף א	1	2	3	4
סעיף ב	1	2	3	4
סעיף ג	1	2	3	4
סעיף ד	1	2	3	4
סעיף ה	1	2	3	4
סעיף ו	1	2	3	4
סעיף ז	1	2	3	4
סעיף ח	1	2	3	4

שאלה 4

סעיף א	1	2	3	4
סעיף ב	1	2	3	4
סעיף ג	1	2	3	4
סעיף ד	1	2	3	4
סעיף ה	1	2	3	4
סעיף ו	1	2	3	4
סעיף ז	1	2	3	4
סעיף ח	1	2	3	4

נספח ב': מילון מונחים (2 עמודים)

לשאלון 714911, אביב תשפ"ב

המונח	תרגום המונח		
	ערבית	רוסית	אנגלית
אחזור	استرجاع	Возврат, извлечение	retrieval
איבר	مُتَغيِّر / عضو	Элемент	item
אקראי	عشوائي	Случайный	random
אתחול	قيمة بدائيّة	Инициализация	initialization
זמן ריצה	مدّة التنفيذ	Время работы	run time
טבלת ערבול (גיבוב)	جدول الخلط	Хеш-таблица	hash table
טיפוס	نوع	Тип	type
מחסנית	باغة	Стек	stack
מטריצת סמיכויות	جدول الحدود الزميلة	Матрица смежности	adjacency matrix
מיון טופולוגי	تصنيف	Топологическая сортировка	topological sorting
מסלול	مسار	Путь	path
מערך דינמי	مصفوفة غير ثابتة	Динамический массив	dynamic array
מצביע	مُؤَشِّر	Указатель	pointer
משתנה גלובלי	مُتَغيِّر عامّ	Глобальная переменная	global variable
סדרה	سلسلة	Последовательность	series
סיבוכיות	تعقيد	Сложность (вычислений)	complexity
עדיפות	أولويّة	Приоритет	preference
עץ חיפוש בינארי מאוזן	شجرة بحث ثنائيّ متوازنة	сбалансированное двоичное дерево поиска	balanced binary search tree
ערך מוחלט	قيمة مُطْلَقة	модуль	absolute value
ערימה	كومة	Куча	heap
ערימה בינארית	كومة ثنائيّة	Двоичная куча	binary heap
פונקצייה רקורסיבית	دالّة أو عمليّة تراجعيّة	Рекурсивная функция	recursive function
צומת	مُفْتَرَق	Узел	node

המשך בעמוד 2

המונח	תרגום המונח		
	ערבית	רוסית	אנגלית
קודקוד	رأس	вершина	vertex
קשת	وصلة	Дуга	arc
רשומה	سِجِلّ	Запись (элемент структуры данных)	record
שדה קישור	حقل رابط	Поле, содержащее ссылку	linking field
שורש	جذر	Корень	root
תת־עץ	شجرة فرعيّة	Поддерево	sub-tree