

**סיבוכיות של אלגוריתמים בקורס
"מבוא למדעי המחשב" באוניברסיטה:
כיצד נוכל לשפר את הוראתה?**

פרופ' יהודית גל-עזר, תמר וילנר, דר' אלה צור

האוניברסיטה הפתוחה

galezertamiela@openu.ac.il

מקומו של המושג יעילות בתכניות הלימודים השונות

- המושג "סיבוכיות של אלגוריתמים" תופס מקום מרכזי בתכניות לימוד באוניברסיטאות
- בתכנית הלימודים החדשה של התיכון מושג זה נלמד, בעיקר לתלמידי 5 יח"ל.

חשיבות הוראת יעילות

- המחשב אינו כל יכול – בעיות בזמן בלתי סביר
- השוואה בין אלגוריתמים שונים לאותה בעיה
- פיתוח יכולת תכנון אלגוריתמים
- הפיכת המתכנת המתחיל (novice) למתכנת מומחה (expert)
- העלאת הרמה הקוגניטיבית של התלמיד

בעייתיות בהוראת המושג

- נטיית סטודנטים להסתפק בנכונות האלגוריתם
- מתן דוגמאות פשוטות יחסית
- ראייה מתמטית עמוקה

זרעים – כבר בקורס המבוא

שאלת שלושת החלקים

- סעיף א – הצגת פונקציה; הסטודנט צריך לזהות מה הפונקציה עושה
- סעיף ב – הסטודנט צריך לחשב את סיבוכיות הפונקציה שניתנה
- סעיף ג – הסטודנט צריך לכתוב פונקציה שמבצעת את מה שביצעה הפונקציה מסעיף א בסיבוכיות קטנה יותר

נתון מערך a בגודל $n-1$ שכל איבר בו הוא שלם בין 1 ל- n .
כל האיברים במערך שונים זה מזה

```
int something (int a[n-1]) {
    int i, j, flag;
    for (j=1; j<=n; j++) {
        flag = 0;
        for (i=0; i<n-1; i++) {
            if (a[i]==j) {
                flag = 1;
                break;
            }
        }
        if (!flag) return j;
    }
    return -1;
}
```

א. מה מבצעת הפונקציה *something*? הסבר בקצרה מה מבצעת הפונקציה באופן כללי, ולא כיצד היא מבצעת זאת.

ב. מהו סדר הגודל של זמן הריצה של הפונקציה *something*?

ג. כתוב את הפונקציה *something* כך שתפתור את הבעיה בסיבוכיות זמן ריצה קטנה מסיבוכיות הזמן של הפונקציה לעיל (קטנה בסדר גודל ולא רק בקבוע). פונקציה שתהיה בסיבוכיות גדולה יותר (מבחינת זמן ו/או מבחינת זיכרון) מזו הנדרשת לפתרון הבעיה לא תקבל את מלוא הנקודות.

פתרונות אפשריים

1. מיון המערך בעזרת מיון-מיזוג (merge-sort),
ואח"כ מעבר על כל המספרים מ-1 עד n
ובדיקה בעזרת חיפוש בינרי איזה מספר לא
נמצא.

$$O(n \log_2 n)$$

2. מיון המערך בעזרת מיון-מיזוג (merge-sort),
ואח"כ מעבר סדרתי על המערך למצוא איזה
מספר חסר.

$$O(n \log_2 n)$$

3. שימוש במערך עזר בגודל n , מעבר על המערך המקורי וסימון במערך העזר של כל המספרים שנמצאים במערך המקורי. אח"כ מעבר על מערך העזר – התא שאינו מסומן, הוא המספר החסר.

סיבוכיות זמן $O(n)$
סיבוכיות מקום $O(n)$

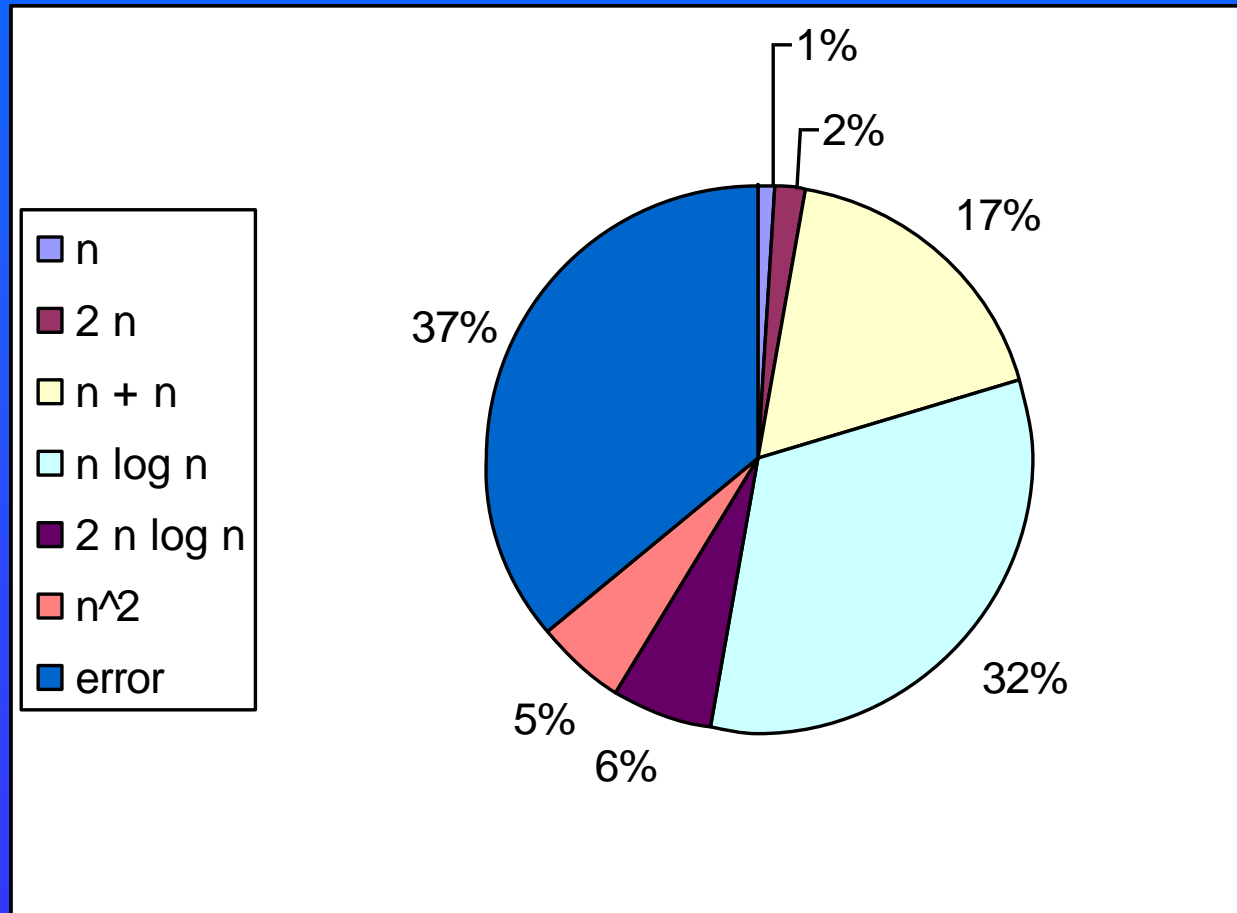
4. חישוב סכום המספרים מ-1 עד n (בעזרת לולאה), סיכום איברי המערך, וחישוב ההפרש בין שני הסכומים – זהו המספר החסר.

סיבוכיות זמן $O(n)$

5. הפתרון האופטימלי - חישוב סכום המספרים
מ-1 עד n (בעזרת נוסחה לחישוב סכון טור
חשבוני), סיכום איברי המערך, וחישוב ההפרש
בין שני הסכומים – זהו המספר החסר.

סיבוכיות זמן $O(n)$

התפלגות הפתרונות



חשוב לדעת!

- אימון הסטודנטים בתכנון אלגוריתמים שונים לאותה בעיה
- הבעיות הנבחרות צריכות להיות בעלות אלגוריתמים שונים לפתרון (מסיבוכיות שונה)
- חשוב שהפתרונות יהיו מסוגי תבניות מגוונים

Suppose a is a given array of length n . Consider the following function:

```
int something (int a[n])
{
    int temp = 0, i, j;
    for (i=0; i<n; i++)
        for (j=i+1; j< n; j++)
            if (abs (a[j]-a[i]) > temp)
                temp = abs (a[j]-a[i]);
    return temp;
}
```

Solution to example 1

- a) The function finds the maximum difference between two values in the array.
- b) $O(n^2)$
- c) Finding the maximum and minimum values in the array and calculating the difference between them. Therefore the problem will be solved in $O(n)$.

Suppose a and b are given arrays of length n . a is sorted in increasing order, and b is not sorted. Consider the following function:

```
int what(int a[],int b[],int &i,int &j)
{
    for (j=0; j<n; j++)
        for (i=0; i<n-1; i++)
            if (b[j] == a[i]+a[i+1])
                return 1;
    return 0;
}
```

Solution to example 2

- a) The function checks whether a value in array b equals the sum of two consecutive values in array a .
- b) $O(n^2)$
- c) For each value in array b , a variation of a binary search needs to be carried out in array a , therefore the problem will be solved in $O(n \log n)$

Suppose a is a given array of length n .

Consider the following function:

```
int something (int a[n])
{
    for (i=0; i<n; i++)
        for (j=i+1; j<n; j++)
            if (a[i] == a[j])
                return 0;
    return 1;
}
```

Solution to example 3

- a) The function checks whether all the values stored in the array a are different .
- b) $O(n^2)$
- c) First, sort the array a , and then pass for finding two adjacent cells with the same value. The sort takes $O(n \log n)$, and the passing $O(n)$. Therefore the problem will be solved in $O(n \log n)$

Suppose a is a given array of length n . Consider the following function:

```
int something (int a[n])
{
    int i,j,temp;
    for (i=0; i< n; i++)
        if (a[i]%2==0)
        {
            temp = a[i];
            for (j=i; j>0; j--)
                a[j] = a[j-1];
            a[0] = temp;
        }
}
```

Solution to example 4

- a) The function rearranges the array a that all the even values will be at the first cells, and the odd values will be at the last cells.
- b) $O(n^2)$
- c) Pass on the array from both sides, and make the necessary exchanges. The following function makes that in $O(n)$:

```
int something (int a[n]) {
    int head=0, tail=n-1, temp;
    while (head<tail) {
        if (a[head]%2==0)
            head++;
        else if (a[tail]%2!=0)
            tail--;
        else {
            temp = a[head];
            a[head] = a[tail];
            a[tail] = temp;
            head++;
            tail--;
        }
    }
}
```