פרק 8 – יעילות של אלגוריתמים

אלגוריתמים נבחנים על פי מספר קני מידה. קנה המידה החשוב ביותר הוא נכונות. כלומר השגת המטרה (מתן הפלט הנכון) עבור כל קלט חוקי. קנה מידה נוסף הוא יעילות.

יעילות של אלגוריתם (תוכנית) נמדדת על פי "משאבי המחשב" הדרושים לביצוע האלגוריתם. משאבים אלה הם **גודל המקום** (בזיכרון) וה**זמן** הדרוש לביצוע האלגוריתם.

גודל המקום נמדד בעיקר על פי מספר המשתנים של האלגוריתם.

זמן-הביצוע נקבע על פי מספר פעולות היסוד שיתבצעו במהלך הרצת האלגוריתם.

פעולות היסוד הן: פעולת קלט, פעולת פלט ופעולות חישוב.

למעשה, בצורה פשטנית נאמר שכל הוראה באלגוריתם כוללת פעולת יסוד אחת, ומכאן מדידת זמן-הביצוע של אלגוריתם נעשית על פי מספר ההוראות שיתבצעו במהלך הרצת האלגוריתם במחשב.

שימו ♥: המדד הוא מספר ההוראות ש**יתבצעו** ולא מספר ההוראות באלגוריתם! מספר ההוראות באלגוריתם אינו מעיד בהכרח על מספר ההוראות שיתבצעו במהלך הרצתו. בפרט, עבור אלגוריתם הכולל לולאה, ייתכן שמספר ההוראות שבו הוא מועט, אך מספר ההוראות שיתבצעו הוא רב, כי כל הוראה בלולאה יכולה להתבצע כמה פעמים. נפרט נקודה זו במהלך הפרק.

מדידת זמן-הביצוע **לא** נעשית על פי הזמן בפועל שאורכת ריצת תוכנית המיישמת את האלגוריתם. יש לכך כמה סיבות:

1. יש מחשבים מהירים יותר ומהירים פחות. מכיוון שפעולות היסוד נמשכות זמן שונה ממחשב למחשב, זמן ההרצה בפועל של אותה תוכנית יכול להיות שונה ממחשב למחשב.
2. גם אם נריץ את התוכנית באותו מחשב כמה פעמים ייתכן שבכל פעם ההרצה תימשך פרק זמן שונה, כתלות בעומס המחשב באותו הזמן. ניתן לדמות זאת לזמן המתנה במסעדה: זמן ההמתנה קצר יותר כאשר במסעדה פחות אורחים. בדומה, זמן ההמתנה לסיום ריצת תוכנית קצר יותר כאשר יש מעט תוכניות נוספות אשר רצות במקביל במחשב.
3. גם המהדר (הקומפיילר) שהשתמשנו בו כדי להדר את התוכנית משפיע על מהירותה של תוכנית. מהדרים שונים יוצרים תוכניות שונות (בשפת מכונה) ולכל אחת מהן זמן ריצה שונה במקצת, גם אם הן מורצות באותו המחשב ובאותה השעה.

בפרק זה נתמקד במדד זמן-הביצוע של אלגוריתם ונכיר אלגוריתמים שונים לפתרון אותה הבעיה. האלגוריתמים ייבדלו זה מזה בזמן-הביצוע, לפעמים בצורה משמעותית. למדד המקום נתייחס בפרק 10.

נמחיש את ההבדל בין זמני הביצוע של שני אלגוריתמים שונים באמצעות בעיה שאנו נתקלים בה מדי פעם בחיי יום-יום, והיא גם אחת מבעיות היסוד במדעי המחשב. הבעיה היא בעיית חיפוש ברשימה ממוינת:

קלט: רשימה של שמות ממוינים לפי סדר הא"ב (למשל מדריך טלפונים או דף קשר).
פלט: הודעה מתאימה אם שם מסוים מופיע ברשימה.

ניתן לחפש את השם המבוקש בשתי דרכים:

הדרך הראשונה, היא בחיפוש **סדרתי**, כלומר מעבר על פני הרשימה, שם אחרי שם, עד שנמצא את השם המבוקש או עד שיתברר שהשם לא מופיע ברשימה. ברשימות ארוכות מאוד, למשל כמו ספר טלפונים, חיפוש כזה עלול לקחת הרבה מאוד זמן.

אם תיזכרו כיצד אתם מחפשים בספר טלפונים תיווכחו שלחיפושים ברשימות ארוכות אנו משתמשים בדרך כלל בחיפושים לא סדרתיים, כמו החיפוש **הלא-סדרתי** הבא: נתבונן בשם המופיע באמצע הרשימה, ונשווה אותו לשם המבוקש, אם השמות זהים – סיימנו את תהליך החיפוש. אם לא, נבדוק לאן עלינו להמשיך את החיפוש: אחרי השם המבוקש או לפניו (כלומר בחֲציה השני של הרשימה או בחציה הראשון). אם השם המבוקש מופיע בסדר הא״ב אחרי השם האמצעי נמשיך את תהליך החיפוש רק בחצי השני (כיוון שהרשימה ממוינת, לא ייתכן שיהיה בחצי הראשון), ואם השם המבוקש מופיע בסדר הא״ב לפני השם האמצעי נמשיך את התהליך רק בחצי הראשון. תהליך החיפוש ימשיך בדיוק באותו אופן: בדיקת השם האמצעי והשוואתו לשם המבוקש, ובהתאם לתוצאת ההשוואה המשך חיפוש במחצית התחום המתאימה, וכך הלאה עד מציאת השם או עד שהסתיים התהליך (והשם לא נמצא). חיפוש זה נקרא חיפוש בינָרי, כיוון שהוא מבוסס על הקטנת תחום החיפוש לחצי מגודלו אחרי כל השוואה של השם המבוקש לשם נבחר (האמצעי בתחום החיפוש).

נדגים זאת:
לפנינו רשימה ממוינת של מספרים:
1, 3, 5, 6, 7, 9, 12, 13, 15, 19, 20, 24, 25, 27, 31, 35, 36
וברצוננו לחפש בתוכה את המספר 27.

בשיטת **החיפוש הסדרתי**, דרושות לנו 14 פעולות השוואה עד שנמצא את המספר המבוקש (כי 27 נמצא במקום ה-14 ברשימה).

נבדוק מה קורה בשיטת **החיפוש הבינרי**:
- נתבונן במספר האמצעי ברשימה, 15. כיוון שהמספר המבוקש **גדול** ממנו, נמשיך את החיפוש מימינו.
- כעת אנו מתמקדים ברשימת המספרים: 19, 20, 24, 25, 27, 31, 35, 36.
- נתבונן במספר האמצעי ברשימה זו, 25 (למעשה, אורך הרשימה זוגי ולכן שני מספרים נמצאים באמצע הרשימה; בחרנו שרירותית את הקטן מביניהם). כיוון שהמספר המבוקש **גדול** ממנו, נמשיך את החיפוש מימינו.
- כעת אנו מתמקדים ברשימת המספרים: 27, 31, 35, 36.
- נתבונן במספר האמצעי ברשימה זו, 31 (גם הפעם, זהו המספר הקטן מבין שני המספרים שבאמצע הרשימה). כיוון שהמספר המבוקש **קטן** ממנו, נמשיך את החיפוש משמאלו.
- כעת אנו מתמקדים ברשימת המספרים: 27.
זו רשימה בת מספר אחד, והוא שווה בדיוק למספר המבוקש, ולכן סיימנו את תהליך החיפוש.

בסך הכול ביצענו ארבע השוואות (למספרים 15, 25, 31 ו-27).
בהשוואה לחיפוש הסדרתי ההבדל הוא משמעותי!

כאשר רשימת השֵמות ארוכה, יש חשיבות רבה לבחירת הדרך שתאפשר את ביצוע משימת החיפוש בזמן קצר עד כמה שניתן. זמן החיפוש נקבע בעיקר על פי מספר ההשוואות המתבצעות במהלך החיפוש (השוואות של השם המבוקש לשם נבחר). לכן נעדיף לבחור בדרך אשר בה יתבצעו פחות השוואות. עבור רשימה בת 1000 שמות, מספר ההשוואות בחיפוש הסדרתי עלול להיות

קרוב ל-1000, כיוון שייתכן שהשם המבוקש נמצא בקצה הרשימה. לעומת זאת, בחיפוש בינרי באותה רשימה מספר ההשוואות לא יעלה בכל מקרה על 10 השוואות. לכן עבור בעיית החיפוש הנתונה, חיפוש בינרי יעיל הרבה יותר. בפרקים הבאים נכיר חיפוש בינרי ביתר פירוט.

יש אנשים הסוברים שמחשבים הינם כה מהירים עד שאין משמעות למושג "זמן-ביצוע של אלגוריתם (או של תוכנית)" ולהשוואה בין זמני הביצוע של אלגוריתמים שונים. לדעה זו אין כל בסיס. קיימות בעיות אשר עבורן זמן הריצה של תוכנית יכול להיות דקות רבות, שעות ואף ימים (למשל תוכניות לחיזוי מזג אויר). עבור בעיית החיפוש שהצגנו, זמן הריצה של תוכנית המבצעת חיפוש סדרתי ברשימה בת 10,000,000 שמות עלול להיות מספר דקות. לעומת זאת, זמן הריצה של תוכנית המבצעת חיפוש בינרי באותה רשימה ובאותו המחשב לא יעלה על מספר שניות.

כזכור, זמן-הביצוע של אלגוריתם נמדד על פי מספר ההוראות שיתבצעו במהלך ביצוע האלגוריתם. מספר זה תלוי בדרך כלל בקלט של האלגוריתם.

למשל, בבעיית החיפוש הקלט לאלגוריתם הוא רשימת השמות הממוינת והשם שיש לחפש. מספר ההוראות שיתבצעו תלוי באורך רשימת השמות: ככל שהרשימה תהיה ארוכה יותר ייתכנו יותר השוואות, כלומר יתבצעו יותר פעולות השוואה.

בפתרון הבעיה הבאה נראה דוגמה לאלגוריתם אשר זמן-הביצוע שלו תלוי בערכו של הקלט.

בעיה 1

מטרת הבעיה ופתרונה: הצגת שלושה אלגוריתמים שונים לפתרון בעיה, הנבדלים זה מזה במידת יעילותם מבחינת זמן-ביצוע. באלגוריתמים היעילים יותר נעשה ניצול טוב של מאפייני קלט-פלט.

פתחו אלגוריתם אשר הקלט שלו הוא מספר שלם גדול מ-1, והפלט שלו הוא המחלקים של המספר הנתון. המחלקים של מספר שלם חיובי הם כל המספרים השלמים החיוביים המחלקים את המספר ללא שארית. לכן למשל עבור הקלט 6 הפלט הדרוש הוא: 6 3 2 1.

ישמו את האלגוריתם בשפת Java. תארו את זמן-ביצוע האלגוריתם על ידי הערכת מספר ההוראות שיתבצעו במהלך ביצוע האלגוריתם.

בדיקת דוגמאות קלט

שאלה 8.1

ציינו מהו הפלט המתאים עבור כל אחד מהקלטים הבאים:

א. 12

ב. 29

ג. 30

תחום המספרים השלמים שיש להציג הוא התחום שבין 1 ובין המספר הנתון כקלט. כלומר, כל מחלק גדול או שווה ל-1, והוא קטן או שווה למספר הנתון. יש להציג כל מספר בתחום אשר מחלק את המספר הנתון ללא שארית.

❓ כיצד אפשר לחשב את המספרים הדרושים לפלט?

הנה רעיון ראשון: נסרוק את תחום המספרים בין 1 לבין המספר הנתון, ונבדוק עבור כל מספר בתחום אם הוא מחלק את המספר הנתון ללא שארית.

ננסח זאת ניסוח ראשוני:

סריקת כל המספרים החיוביים בין 1 ועד למספר הנתון כקלט, ובדיקה עבור כל מספר אם הוא מחלק את המספר הנתון. אם כן, הצגתו כפלט.

❓ הרעיון המתואר מציג תת-משימה לביצוע-חוזר. מהי תת-משימה זו?

הרעיון המתואר כולל ביצוע-חוזר של התת-משימה הבאה, עבור כל מספר בתחום שבין 1 למספר הנתון:

אם המספר מחלק את המספר הנתון, הצגתו כפלט.

ננסח אלגוריתם לביטוי הרעיון המתואר.

בחירת משתנים

משתני האלגוריתם הם:

num – מטיפוס שלם, לשמירת נתון הקלט.

i – מטיפוס שלם, משתנה בקרה של ההוראה לביצוע-חוזר.

האלגוריתם

<u>אלגוריתם 1</u>:

1. *קלוט מספר שלם חיובי* num-*ל*
2. *עבור כל מספר שלם חיובי* i *הקטן מ-*num *או שווה לו בצע:*
   2.1. *אם* i *מחלק את* num *ללא שארית*
      2.1.1. *הצג את ערכו של* i

שימו ♥: מתאים להשתמש כאן בהוראה לביצוע-חוזר, שמספר הסיבובים בה מחושב מראש (ממומשת בשפת Java בלולאת **for**). בהוראה אנו מפרטים את תחום הערכים הנסרק באמצעות משתנה הבקרה של הלולאה (i).

באלגוריתם נעשה שימוש במשתנה הבקרה בגוף הלולאה. גוף הלולאה כולל בדיקה אם ערכו של משתנה הבקרה i הוא מחלק של num. כיוון שבמהלך ביצוע הלולאה ערכי משתנה הבקרה משתנים מ-1 עד num, הרי בדיקת החלוקה מתבצעת עבור כל מספר שלם בתחום 1 עד num.

יישום האלגוריתם

הנה התוכנית המיישמת את אלגוריתם 1:

```
/*
קלט: מספר שלם חיובי
פלט: כל המחלקים של המספר הנתון */
import java.util.Scanner;
public class Divisors1
{
    public static void main (String [] args)
    {
        int num;   // המספר הנתון
        int i;     // משתנה הבקרה
        Scanner in = new Scanner(System.in);
        System.out.print("Enter a number: ");
        num = in.nextInt();
        System.out.println( "The divisors of " + num + " are " );
        for(i = 1; i <= num ; i++)
```

```
            if (num % i == 0)
                System.out.println(i);
    } // main
} // Divisors1
```

ניגש עתה לחישוב זמן-ביצוע האלגוריתם על ידי הערכת מספר ההוראות שיתבצעו במהלך הביצוע. כדי לבצע את ההערכה נתמקד במרכיב העיקרי באלגוריתם המשפיע על מספר ההוראות שיתבצעו.

❓ מהו המרכיב העיקרי באלגוריתם המעיד על מספר ההוראות שיתבצעו?

באלגוריתם אשר אינו כולל לולאה אין בעצם מרכיב בולט. מספר ההוראות שיתבצעו במהלך ביצוע האלגוריתם הוא לכל היותר מספר ההוראות באלגוריתם. לעומת זאת, באלגוריתם הכולל לולאה, הלולאה היא המרכיב העיקרי המעיד על מספר ההוראות שיתבצעו. זאת כיוון שייתכן שהלולאה תתבצע מספר רב של פעמים, ובכל ביצוע-חוזר יתבצעו שוב כל ההוראות שבגוף הלולאה.

מכיוון שייתכן שהלולאה תתבצע מספר רב של פעמים, הרי מספר ההוראות ש**יתבצעו** במהלך ריצת האלגוריתם עשוי להיות גדול בהרבה ממספר ההוראות ש**כתובות** באלגוריתם. למשל במהלך ביצוע אלגוריתם 1 עבור הקלט 1000, תתבצע ההוראה לביצוע-בתנאי שבגוף הלולאה 1000 פעמים. מספר זה גדול בהרבה ממספר ההוראות שבאלגוריתם.

בדרך כלל הלולאה תתבצע מספר שונה של פעמים עבור קלטים שונים. לכן כדי לתאר בצורה כללית את מספר הפעמים שלולאה תתבצע, יש לבטא מספר זה על פי הקלט.

❓ מהו מספר הפעמים שתתבצע הלולאה באלגוריתם 1 עבור קלט שערכו N ?

מספר הפעמים שהלולאה באלגוריתם 1 תתבצע עבור קלט שערכו N הוא N. הלולאה תתבצע פעם אחת עבור כל אחד מהערכים 1, 2, 3, ..., N למשתנה הבקרה i .

כיוון שלולאה היא המרכיב העיקרי המעיד על מספר הפעולות שיתבצעו באלגוריתם, נהוג להתייחס למספר זה כמדד לחישוב זמן-הביצוע. לכן נאמר שתוצאת חישוב זמן-הביצוע של אלגוריתם 1 היא שלולאת האלגוריתם תתבצע N פעמים עבור קלט שערכו N.

ננסה לראות אם ביכולתנו לפתח אלגוריתם יעיל יותר מאלגוריתם 1 מבחינת זמן-הביצוע.

❓ האם אפשר לפתח אלגוריתם שבו תהיה לולאה שתתבצע פחות מ-N פעמים עבור קלט שערכו N ?

כן. ניתן לפתח אלגוריתם ובו לולאה "יעילה" יותר. נראה זאת כעת.

אחד המאפיינים של חלוקה של מספרים שלמים היא שהמחלקים של מספר שלם חיובי num אינם גדולים מ-num/2, מלבד המספר num עצמו. ניתן לראות זאת בדוגמאות הקלט שנבדקו בשאלה 8.1. לכן בעצם אפשר "לחסוך" ולהריץ את משתנה הבקרה בלולאה שבאלגוריתם רק עד ל-num/2. נבטא רעיון זה באלגוריתם 2, והוא אלגוריתם יעיל יותר לפתרון הבעיה.

<u>אלגוריתם 2</u>:

1. קלוט מספר שלם חיובי ב-num
2. עבור כל מספר שלם חיובי i הקטן מ-num/2 או שווה לו בצע:
   2.1. אם i מחלק את num ללא שארית
      2.1.1. הדפס את ערכו של i
3. הדפס את ערכו של num

אמנם הוצאנו את הצגתו של num אל מחוץ ללולאה, אך עיקר העבודה נעשית בלולאה. באלגוריתם 2, תתבצע הלולאה רק $N/2$ פעמים עבור קלט שערכו N. זהו שיפור משמעותי, במיוחד עבור קלטים שערכם גדול (למשל 10,000).

יישום האלגוריתם

הנה התוכנית המיישמת את אלגוריתם 2:

```
/*
קלט: מספר שלם חיובי
פלט: כל המחלקים של המספר הנתון
*/
import java.util.Scanner;
public class  Divisors2
{
    public static void main (String [] args)
    {
        int num;   // המספר הנתון
        int i;     // משתנה הבקרה
        Scanner in = new Scanner(System.in);
        System.out.print("Enter a number: ");
        num = in.nextInt();
        System.out.println( "The divisors of " + num + " are " );
        for(i = 1; i <= num / 2 ; i++)
            if (num % i == 0)
               System.out.println(i);
        System.out.println(num);
    } // main
} // Divisors2
```

שיפרנו את הפתרון הראשון. האם נוכל להמשיך ולשפר גם את הפתרון השני?

❓ האם אפשר לפתח אלגוריתם שבו תהיה לולאה שתתבצע **פחות** מ-$N/2$ פעמים עבור קלט שערכו N?

כן. אפשר לפתח אלגוריתם ובו לולאה "יעילה" יותר.

לכל מספר שלם k שהוא מחלק של num, יש "בן-זוג" שלם num/k, שגם מחלק את num. (שהרי $num = k \cdot num/k$). אם באלגוריתם, עבור כל מחלק k שנמצא, נציג כפלט גם את num/k, לא נצטרך לעבור על כל המחלקים של num, אלא רק עד מחציתם (שהרי הצגנו כבר את "בני-הזוג").

❓ היכן נמצא קו המחצית של המחלקים? כלומר, מתי עלינו להפסיק את החיפוש כדי לא להציג מחלקים כפולים?

התשובה היא $\sqrt{num}$. כך נובע מהאבחנה הבאה: עבור כל זוג מחלקים k ו-num/k, לפחות אחד מהם אינו גדול מ- $\sqrt{num}$. מדוע? משום שאם שניהם גדולים מ- $\sqrt{num}$, אז מכפלתם גדולה מ-num! אם כך, אם נסרוק את כל המספרים מ-1 עד $\sqrt{num}$, ונציג עבור כל מחלק גם את "בן-זוגו" המחלק, למעשה נציג את כל המחלקים של num.

❓ מהו בן זוגו של $\sqrt{num}$?

זהו $\sqrt{num}$ בעצמו, משום שמכפלתו של $\sqrt{num}$ בעצמו שווה ל-num.

לכן משום שבן-זוגו של $\sqrt{num}$ הוא $\sqrt{num}$ עצמו, עלינו לשים לב לא להציג את $\sqrt{num}$ פעמיים כפלט.

לדוגמה: אם num=100 אז $\sqrt{num} = 10$. כל מחלקיו של 100 הקטנים מ-10 הם: 2, 4, ו-5. "בני זוגם" הם: 50, 25 ו-20, בהתאמה. אם נוסיף להם את $\sqrt{num} = 10$ נקבל את הרשימה המלאה של כל המחלקים של 100.

לכן בעצם ניתן להריץ את משתנה הבקרה בלולאה שבאלגוריתם עד ל- $\sqrt{num}$ בלבד. נבטא רעיון זה באלגוריתם 3, שהוא אלגוריתם יעיל יותר משני האלגוריתמים האחרים.

<u>אלגוריתם 3</u>:

1. *קלוט מספר שלם חיובי* num-ב
2. *עבור כל מספר שלם חיובי* i *הקטן מ-* $\sqrt{num}$ *בצע*:
   - 2.1. *אם* i *מחלק את* num *ללא שארית*
     - 2.1.1. *הדפס את ערכו של* i
     - 2.1.2. *הדפס את ערכו של* num/i
3. *אם* $\sqrt{num}$ *מחלק את* num *ללא שארית*
   - 3.1. *הדפס את ערכו של* $\sqrt{num}$

כאמור, הוצאנו אל מחוץ ללולאה את הבדיקה של $\sqrt{num}$ כדי שלא נציג את ערכו פעמיים במקרה שהוא מחלק את num. לעומת זאת, אין צורך לטפל מחוץ בלולאה ב-num עצמו, בניגוד לאלגוריתם הקודם, משום שהוא בן הזוג של 1 ולכן יוצג כפלט בסיבוב הראשון בלולאה.

כעת שיפרנו את אלגוריתם 2 באופן משמעותי. עבור קלטים גדולים מאוד המספר $\sqrt{N}$ קטן משמעותית מהמספר N/2. למשל, אם N הוא 10,000 אז באלגוריתם 2 יהיו 5000 סיבובים בלולאה, בעוד שבאלגוריתם 3 יהיו 100 סיבובים בלבד!

ישמו בעצמכם את האלגוריתם בשפת Java.

סוף פתרון בעיה 1

נסכם את הנלמד מפתרון בעיה 1:

- באלגוריתם שאינו כולל לולאה מספר הפעולות שיתבצעו הוא לכל היותר מספר ההוראות באלגוריתם.
- באלגוריתם שכולל לולאה, מספר ההוראות שיתבצעו עשוי להיות גדול בהרבה ממספר הוראות האלגוריתם. מספר ההוראות שיתבצעו תלוי במספר הפעמים שהלולאה תתבצע. לכן מספר הפעמים שהלולאה תתבצע משמש כמדד לזמן-ביצוע האלגוריתם.
- מספר הפעמים שהלולאה תתבצע תלוי בדרך כלל בערכו של הקלט, ומבוטא באמצעות ערכו.

בפתרון בעיה 1 פיתחנו תחילה אלגוריתם אחד, ואחר כך פיתחנו אלגוריתם שני אשר היה יעיל יותר מבחינת מספר הפעמים של ביצוע הלולאה, כלומר מבחינת זמן-הביצוע. ולבסוף מצאנו אלגוריתם יעיל יותר גם ממנו. בפיתוח האלגוריתם השני השתמשנו בעובדה שאיברי הפלט, הלוא הם כל המחלקים של נתון הקלט N, אינם גדולים מ-N/2 מלבד N עצמו. בפיתוח האלגוריתם השלישי, היעיל מבין השלושה, השתמשנו בעובדה שאיברי הפלט, ניתנים לחלוקה לזוגות כך שמכפלת איברי כל זוג שווה ל-N ובכל זוג יש איבר אחד קטן מ- $\sqrt{N}$. העובדה האחרונה אפשרה לנו לכתוב לולאה אשר תתבצע $\sqrt{N}$ פעמים בלבד, במקום הלולאה המקורית שמתבצעת N פעמים.

- ♦ פיתוח אלגוריתם יעיל יותר נעשה תוך ניתוח וניצול טוב של מאפייני קלט-פלט. שימוש טוב במאפייני קלט-פלט חשוב לפיתוח לולאה אשר תתבצע מספר מועט של פעמים עד כמה שאפשר. כלומר השקעת מאמץ בניתוח מעמיק של הבעיה יכולה להתבטא אחר כך באלגוריתם שהוא משמעותית יעיל יותר.
- ♦ במהלך פתרון בעיה אלגוריתמית נשתדל לפתח אלגוריתם יעיל ככל האפשר מבחינת זמן-הביצוע. כלומר אלגוריתם שהלולאות שבו יתבצעו מספר פעמים מועט (לולאות "יעילות") עד כמה שניתן.

שאלה 8.2

ציינו עבור כל אחד מהקלטים הבאים את מספר הפעמים שתתבצע הלולאה בכל אחד משלושת האלגוריתמים שפיתחנו:

א. 1000

ב. 2000

שאלה 8.3

בקטע התוכנית הבא מחושבת המכפלה של שני נתוני קלט חיוביים שלמים בשימוש בפעולת חיבור בלבד:

```
x = in.nextInt();
y = in.nextInt();
sum = 0;
for(i = 1; i <= x ; i++)
    sum = sum + y;
System.out.println( "The product is " + sum);
```

ידוע שאחד מנתוני הקלט גדול באופן משמעותי מהשני, אך לא ידוע אם זה הנתון הראשון או השני. השתמשו במאפיין זה של הקלט כדי לשנות את קטע התוכנית הנתון כך שיהיה יעיל ככל שניתן.

הדרכה: שימו לב שיש חשיבות לבחירת המשתנה אשר על פיו נקבע מספר הפעמים שתתבצע הלולאה.

מהו מספר הפעמים שתתבצע הלולאה של קטע התוכנית הנתון, ומהו מספר הפעמים שתתבצע הלולאה של קטע התוכנית החדש?

שאלה 8.4

פתחו אלגוריתם מבלי ליישמו שיהיה יעיל ככל האפשר, והקלט שלו הוא שני מספרים שלמים גדולים מ-1 שאינם מחלקים זה את זה, והפלט שלו הוא כל המספרים השלמים החיוביים המחלקים את שני מספרי הקלט. תארו לפי ערכו של הקלט את מספר הפעמים שתתבצע לולאת האלגוריתם.

שאלה 8.5 (מתקדמת)

נניח שבבעיה 1 הפלט הדרוש הוא כל המספרים השלמים החיובים הקטנים מ-N ש**אינם** מחלקים את נתון הקלט N. מה יהיה מספר הפעמים שתתבצע הלולאה באלגוריתם לפתרון הבעיה החדשה?

שאלה 8.6

יש לפתח אלגוריתם אשר הקלט שלו הוא מספר שלם חיובי N, והפלט שלו הוא כל המספרים השלמים החיוביים אשר קטנים מ-N והשורש שלהם הוא מספר שלם.
למשל עבור הקלט 50 הפלט יהיה 49 36 25 16 9 4 1.

האלגוריתם הבא, הכולל משתנים מטיפוס שלם הוא פתרון אפשרי:

1. קלוט מספר שלם חיובי ל-num
2. עבור כל מספר שלם חיובי i שקטן מ-num בצע:
   2.1. אם השורש של i הוא מספר שלם
      2.1.1. הדפס את ערכו של i

א. מהו מספר הפעמים שתתבצע הלולאה של האלגוריתם הנתון?
ב. ישמו בביטוי בוליאני בשפת Java את התנאי "השורש של i הוא מספר שלם"
ג. אפשר לכתוב אלגוריתם אשר זמן-הביצוע שלו יהיה קצר בהרבה מהאלגוריתם הנתון, וזאת ב"ייצור" מספרי הפלט, באמצעות העלאה בריבוע של כל המספרים השלמים אשר ריבועם קטן מן הקלט.
האלגוריתם החלקי הבא מבוסס על הרעיון המתואר:

1. קלוט מספר שלם חיובי ל-num
   1.1. עבור כל מספר שלם חיובי i הקטן מ-_____ בצע:
      1.1.1. הדפס את ערכו של i^2

השלימו את האלגוריתם ותארו את מספר הפעמים שתתבצע הלולאה של אלגוריתם זה.

כל האלגוריתמים שניתחנו עד עתה בפרק כללו לולאה שמספר הביצועים שלה מחושב מראש ובה הורץ משתנה הבקרה מ-1 עד ערך כלשהו בקפיצות של 1. לכן קל היה לחשב את מספר הפעמים של ביצוע הלולאה.

בלולאות מורכבות יותר, בהן ערכו ההתחלתי של משתנה הבקרה אינו 1, והשינוי בערכו בין סיבוב לסיבוב הוא לאו דווקא 1, חישוב מספר הסיבובים עלול להיות מסובך יותר. בלולאת while שבה אין משתנה בקרה, חישוב זמן-הביצוע יכול להיות אף מורכב יותר. יש לספור את מספר הפעמים של ביצוע הלולאה במעקב אחר שינוי ערכי משתנים המתעדכנים בגוף הלולאה ומופיעים בתנאי הכניסה ללולאה.

שאלה 8.7

יש לפתח וליישם אלגוריתם אשר הקלט שלו הוא שני מספרים שלמים חיוביים, כך שהמספר השני גדול מהראשון. הפלט הדרוש הוא הכפולות של המספר הראשון אשר קטנות מהמספר השני או שוות לו. למשל עבור הקלט 1000 200 הפלט יהיה 1000 800 600 400 200.
משפטי התוכנית הבאים הם יישום של אלגוריתם לפתרון הבעיה:

```
x = in.nextInt();
y = in.nextInt();
i = x;
for (i = x; i <= y; i++)
    if (i % x == 0)
        System.out.println(i);
```

א. כמה פעמים תתבצע הלולאה עבור הקלט 1000 200?
ב. תארו בצורה כללית את מספר הפעמים שתתבצע הלולאה על פי ערכי נתוני הקלט x ו-y.

ג. בפתרון המוצג `i` גדל בקפיצות של 1. ניתן לשפר את יעילות הפתרון הנתון בשינוי הקפיצות של `i` לקפיצות גדולות מ-1, קפיצות אשר מתאימות למרחק בין זוג מספרי פלט עוקבים (שימו לב שהמרחק בין כל זוג מספרי פלט עוקבים הוא אחיד). כתבו פתרון יעיל יותר המבוסס על הרעיון המתואר, ותארו את מספר הפעמים שתתבצע לולאת הפתרון החדש.

שאלה 8.8

נתונה לולאת ה-**for** הבאה:

```
for (i = 1 ; i <= 30000 ; i++)
    if (i % 500 == 0)
        System.out.println(i);
```

א. מהו מספר הפעמים שתתבצע הלולאה?

ב. מהי מטרת הלולאה?

ג. כתבו לולאה יעילה הרבה יותר להשגת אותה המטרה. מהו מספר הפעמים שתתבצע הלולאה היעילה שכתבתם? פי כמה מספר זה קטן מתשובתכם בסעיף א?

סיכום

בפרקים הקודמים בחנו אלגוריתמים על פי קנה המידה **נכונות**. בפרק זה הכרנו קנה מידה חדש – **יעילות**.

יעילות של אלגוריתם נמדדת על פי "משאבי המחשב" הדרושים לביצוע האלגוריתם. משאבים אלה הם גודל המקום בזיכרון והזמן הדרוש לביצוע.

גודל המקום נמדד בעיקר על פי מספר המשתנים באלגוריתם.

זמן-הביצוע נקבע לפי מספר פעולות היסוד שיתבצעו במהלך ביצוע האלגוריתם.

פעולות היסוד הן: פעולות קלט, פעולות פלט ופעולות חישוב. בצורה פשטנית ניתן לומר, שכל הוראה באלגוריתם כוללת פעולת יסוד אחת, ומכאן – **מדידת זמן-הביצוע** של אלגוריתם נעשית על פי מספר ההוראות שיתבצעו במהלך ריצת האלגוריתם.

באלגוריתם אשר אינו כולל לולאה מספר ההוראות שיתבצעו במהלך הביצוע הוא לכל היותר מספר ההוראות באלגוריתם.

לעומת זאת באלגוריתם הכולל לולאה, מספר ההוראות שיתבצעו במהלך הביצוע אינו נקבע על פי מספר ההוראות באלגוריתם, אלא על פי מספר הפעמים שהלולאה תתבצע. מספר זה יכול להיות תלוי בקלט של האלגוריתם ואז הוא מבוטא באמצעות מאפייני הקלט.

כאשר נתונים שני אלגוריתמים שונים לפתרון בעיה, משווים את יעילותם מבחינת זמן-הביצוע לפי מספר הפעמים שהלולאות שבהם יתבצעו במהלך הרצת כל אלגוריתם.

בפיתוח אלגוריתם נשתדל לבחור לולאות שיתבצעו מספר פעמים מועט עד כמה שניתן, כלומר לולאות "יעילות" ככל האפשר. בחירת לולאה יעילה נעשית בניתוח ובניצול טוב של מאפייני הקשר בין הקלט לפלט.