פרק ה: יעילות

יש יותר מדרך אחת לפצח אגוז. אפשר להניחו על הרצפה ולרקוע עליו, אפשר לפצחו בעזרת השיניים או להיעזר באגוז נוסף ובחבר חזק, ואפשר כמובן להשתמש במפצח אגוזים. בכל הדרכים נשיג את מטרתנו - אגוז מפוצח. מבחינת התוצאה, כל השיטות נכונות. השיטות השונות נבדלות ביניהן במאמץ הנדרש מאתנו לביצוע המשימה, במשך הזמן שהיא דורשת, באמצעים העומדים לרשותנו (מפצח אגוזים, חבר חזק...) ובעלות הביצוע (טיפול שיניים או נעל חדשה במקרים קיצוניים). אנו נעדיף את השיטה שתהיה היעילה ביותר עבורנו.

גם במדעי המחשב אנו רוצים שהאלגוריתמים שאנו מתכננים והתכניות שאנו בונים יהיו יעילים. כאשר מודדים יעילות של אלגוריתמים ותכניות, מתחשבים בשני קריטריונים: זמן ומקום. הזמן הוא זמן חישוב, כלומר משך הזמן שבו מתבצעת התכנית, הקריטריון השני - המקום - מתייחס לגודל זיכרון המחשב שיש להקצות לצורך ביצוע התכנית. קריטריון הזמן וקריטריון המקום מכונים **מדדי סיבוכיות** (complexity measures).

הבעיות המתעוררות בהערכת אלגוריתמים לפי שני הקריטריונים דומות במידה רבה, וחקר סיבוכיות המקום עוסק בנושאים דומים לאלו של סיבוכיות הזמן. אנו נתרכז בחישובי הסיבוכיות במדד הזמן.

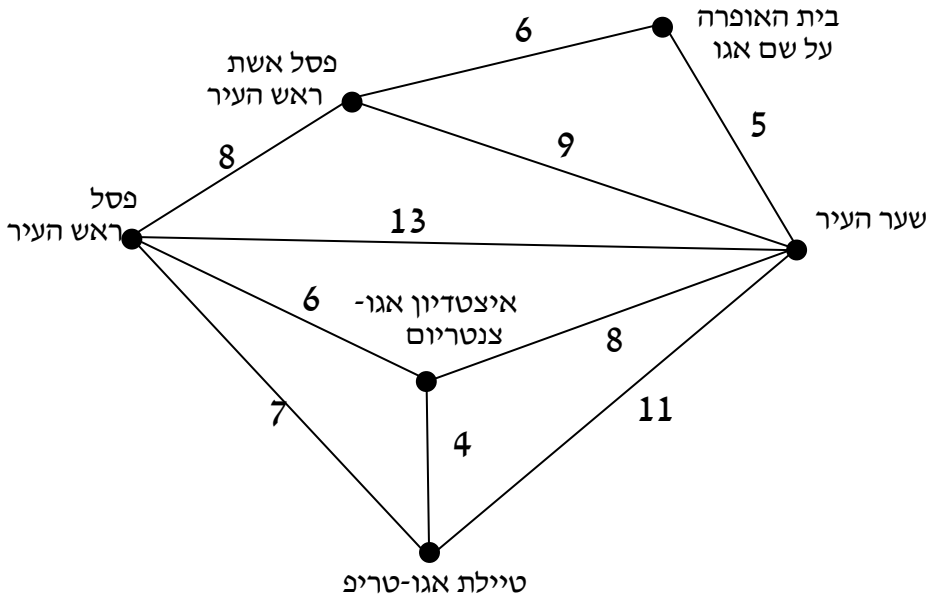
כיום נפוצים מאוד מחשבים שמבצעים מיליוני פעולות בשנייה, וכוח החישוב של מחשבים עוד הולך ומשתפר עם השנים. כל דור חדש מהיר בהרבה מהדור הקודם. נשאלת השאלה, מדוע יש להתחשב בזמן הריצה של אלגוריתמים? מדוע כדאי לדון בשאלה אם תכנית תבצע שני מיליון פעולות או רק מיליון אחד? הרי בכל מקרה משך הריצה של תכנית כזו יהיה שבריר של שנייה.

הנחה זו מוטעית מיסודה. כשאנו רוצים לחפש ספר בקטלוג הממוחשב של הספרייה או לקבל מספר טלפון של מנוי מסוים במודיעין '144', איננו רוצים לחכות יותר מכמה שניות עד שתינתן תשובה. כיוון שמדובר בחיפוש באוספי נתונים גדולים מאוד, יש חשיבות מרובה לשימוש באלגוריתם יעיל לחיפוש. במערכת חיזוי מזג אוויר, למשל, המהירות חשובה אף יותר, שהרי מה הערך של התחזית למחר אם נקבלה רק מחרתיים? במערכת כזאת דרושים אלגוריתמים יעילים לעיבוד הכמויות הגדולות של הנתונים ולביצוע החישובים הרבים והמסובכים. גם במערכות לבקרת טיסה, הנחיית טילים, בקרת מפעלי תעשייה ואפילו במשחקי מחשב, למשל, הזמן הוא גורם מכריע. מערכות אלו חייבות להגיב על גירויים חיצוניים, כגון לחיצה על כפתורים, בזמן אמת, כלומר כמעט מיד. אחרת, הדבר עלול להוביל לתוצאות שליליות ואף קטלניות.

1. בעיות "בלתי סבירות"

קיימות בעיות שפתרונן יארך, אפילו במחשבים המהירים ביותר הקיימים כיום (וגם באלו שייבנו בעתיד הנראה לעין), זמן בלתי סביר בעליל. נתבונן לדוגמה בבעיה זו:

עקב שפל חסר תקדים במספר התיירים הפוקדים את העיר שיממון, החליט ראש העירייה, מר אגו, לקדם את עסקי התיירות בעירו. לשם כך הגה תכנית לקו אוטובוס שיעבור באתרי התיירות המרכזיים בעיר. הוא זימן למשרדו את מנהל חברת האוטובוסים, הסביר לו את מבוקשו, וצייד אותו במפת העיר, שהכין מבעוד מועד, ובה צוינו האתרים החשובים, הכבישים המחברים ביניהם ואורכו של כל כביש.



איור 1 - מפת העיר שיממון (המספרים מציינים מרחקים בק"מ. אורכי הכבישים אינם לפי קנה מידה אחיד.)

מנהל חברת האוטובוסים כינס את צוות התכנון של החברה והטיל עליו את המשימה הזאת: "למצוא את המסלול הקצר ביותר היוצא משער העיר ומסתיים בו. המסלול צריך לעבור בכל אתר תיירות פעם אחת בלבד".

? *נסו למצוא שיטה אלגוריתמית לפתרון הבעיה עבור כל רשימה נתונה של אתרים ומרחקים.*

פתרון אפשרי לבעיה זו הוא מציאת כל המסלולים האפשריים, שמתחילים ומסתיימים בשער העיר ועוברים בכל אתר תיירות פעם אחת בלבד, חישוב אורכיהם ומציאת הקצר שבהם. בעיה זו נראית פשוטה, שהרי מספר המסלולים האפשריים במפת העיר שיממון לכאורה אינו גדול. אם נכתוב תכנית מחשב שתפתור את הבעיה, נראה שזמן ריצתה של התכנית - כאשר ניתן לה כקלט את המפה הנתונה - יהיה קצר למדי.

כמה מסלולים כאלה קיימים? הנקודה הראשונה במסלול היא קבועה: שער העיר. לבחירת הנקודה השנייה יש לנו חמש אפשרויות שונות, מכיוון ששער העיר מחובר לכל האתרים שבעיר. לאחר שבחרנו את הנקודה השנייה, יש לנו לכל היותר ארבע אפשרויות שונות לבחירת הנקודה השלישית, וכן הלאה. אנו מציינים שיש לנו לכל היותר x אפשרויות, כיוון שלמעשה יכול להיות שיהיו לנו פחות אפשרויות, שהרי יש מקומות שאינם מקושרים לאחרים. למשל, אם הנקודה השנייה שנבחרה היא בית האופרה, לא נוכל לבחור את פסל ראש העיר כנקודה השלישית. אם כן, מספר המסלולים האפשריים הוא לכל היותר: $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$.

מה יקרה אם נכליל את האלגוריתם לעיר שבה n אתרים? מספר המסלולים האפשריים הוא לכל היותר !(n-1). נניח שיש לנו מחשב היכול לחשב מיליון מסלולים בשנייה. לעיר שבה 11 אתרים יידרשו למחשב כ-4 שניות לעבור על כל המסלולים האפשריים ולמצוא את הפתרון. זהו משך זמן סביר. אך אם נעבור למפה גדולה רק במעט, שבה 15 אתרים, יזדקק אותו מחשב ליום שלם כדי למצוא את הפתרון. בעיר תיירותית מצויה שבה 20 אתרים, יידרשו למחשב כ-3,857 שנים למציאת הפתרון... מצב זה הוא בלתי יעיל בעליל: אף שיש בידינו אלגוריתם למציאת הפתרון, הרי שמשך החיפוש בלתי סביר.

טבלה 1 מציגה את מספר המסלולים האפשריים עבור מספרים שונים של אתרים ואת הזמנים הדרושים לחישובם:

מספר האתרים	מספר המסלולים	זמן החישוב
6	120	8 מילישניות
11	3628800	כ-3.5 שניות
13	479,001,600	כ-8 דקות
16	1,307,674,368,000	כ-15 ימים
18	~355,000,000,000,000	כ-11 שנים
21	~2,430,000,000,000,000,000	כ-77,000 אלף שנים

טבלה 1 - מספר המסלולים והזמן הדרוש לחישובם

בעיה זו נקראת "בעיית הסוכן הנוסע" על שמו של סוכן נוסע אלמוני, שניסה למצוא את המסלול הקצר ביותר העובר בין לקוחותיו. האם יש אלגוריתם אחר לפתרון בעיית הסוכן הנוסע, שזמן ריצתו קצר יותר? בעיה זו משמשת מוקד למחקר במדעי המחשב, וטרם נמצא אלגוריתם הפותר אותה בזמן סביר.

בעיית הסוכן הנוסע אינה היחידה במובן זה. יש אוסף גדול מאוד של בעיות, לרבות מהן שימושים בכלכלה ובתעשייה, כגון תכנון לוחות זמנים, ניהול מלאי והעברת משלוחים, שדינן דומה. על מנת לפתור כל אחת מהן עבור קלטים לא גדולים במיוחד באמצעות האלגוריתם הטוב ביותר הידוע לנו, ואפילו על מחשבים מהירים בהרבה מהמחשבים הקיימים כיום, ניאלץ להמתין לתשובות שנים רבות.

ראינו שיש בעיות שאפשר לפתור במחשב בעזרת אלגוריתמים פשוטים למדי שאינם יעילים. הרצת התכנית במקרים אלו תארך זמן בלתי סביר אפילו לקלטים קצרים יחסית. לקח ברור הוא שכאשר אנו מתכננים אלגוריתם לפתרון בעיה כלשהי, עלינו לבדוק את יעילותו ולוודא כי אפשר לפתור בעזרתו את הבעיה בזמן סביר. אם לא כך הדבר, רצוי למצוא אלגוריתם יעיל יותר. לפעמים לא קל למצוא אלגוריתם שכזה (כמו בבעיית הסוכן הנוסע), אך במקרים רבים הדבר אפשרי.

2. איך מודדים יעילות?

אילו הציגו בפניכם שני אלגוריתמים שונים לפתרון בעיה מסוימת, והיה עליכם להכריע איזה מבין השניים יעיל יותר, האם הייתם מסתפקים בתשובה שלפניכם?

האלגוריתמים מומשו במחשב וזמני הריצה שלהם נמדדו. להלן התוצאות:
אלגוריתם א' - פתרון הבעיה נמשך 1.25 שניות.
אלגוריתם ב' - פתרון הבעיה נמשך 0.34 שניות.
מסקנה: אלגוריתם ב' יעיל יותר!

לא ניתן להכריע איזה אלגוריתם טוב יותר, כיוון שחסרים לנו פרטים רבים. למשל: האם מדידת זמן הריצה של שני האלגוריתמים בוצעה על אותו מחשב או על שני מחשבים שונים? הרי ישנם מחשבים שונים עם מעבדים בעלי מהירויות שונות. ובכלל, האם משך הזמן הוא המדד המעניין אותנו? נראה שאנו צריכים להעריך את האלגוריתם עצמו ללא תלות במחשב שעליו הוא רץ.

משך הריצה של אלגוריתם תלוי ב"כמות העבודה" שעליו לבצע, וזו תלויה ב**אורך הקלט** שעליו פועל האלגוריתם. אלגוריתם שמסכם איברי מערך ירוץ זמן קצר יותר על מערך שבו 10 תאים בהשוואה לאותו אלגוריתם שירוץ על מערך שבו 100 תאים.

לכל בעיה יש להגדיר את אורך הקלט המתאים לה. באלגוריתם לסיכום איברי מערך, אורך הקלט הוא גודל המערך. אם אנו רוצים לחפש מילה בטקסט, אורך הקלט יכול להיות מספר האותיות או מספר המילים בטקסט. באלגוריתם המכפיל מספרים, אורך הקלט הוא מספר הספרות במספרים שעלינו להכפיל.

לאחר שהגדרנו את אורך הקלט המתאים לבעיה, נוכל לעסוק ביעילותו של הפתרון האלגוריתמי המוצע כתלות באורך הקלט. נעשה זאת על ידי ספירת הפעולות שמבצע האלגוריתם על אורכי קלט שונים.

בכל אלגוריתם אפשר למצוא פעולה או כמה פעולות המשמשות יחד **צעד בסיסי** שעליו חוזר האלגוריתם במהלך ריצתו. משך ביצועו של צעד בסיסי זה *אינו תלוי באורך הקלט*, כלומר הוא אורך זמן קבוע. כך למשל, אלגוריתם המסכם איברי מערך חוזר כמה פעמים על הצעד: "הוסף את האיבר הבא במערך לסכום המצטבר". אלגוריתם שבודק האם מספר x הוא ראשוני, יחזור על הצעד: "בדוק האם המספר y מחלק את x". אלגוריתם המכפיל שני מספרים מבצע שני סוגים של צעדים בסיסיים: הכפלת שתי ספרות וחיבור שתי ספרות. לעומת זאת הפעולה "מצא האם איבר x נמצא במערך A" אינה צעד בסיסי, שכן משך ביצועה תלוי באורך הקלט, בגודל המערך.

מספר הפעמים שהאלגוריתם מבצע את הצעד הבסיסי במהלך ריצתו תלוי באורך הקלט ומשמש מדד טוב ליעילותו. נשים לב כי מדד זה אכן אינו תלוי במחשב מסוים, אך אפשר בקלות להעריך כמה זמן תארך ריצתו של אלגוריתם על מחשב נתון. נעריך מהו משך הביצוע של צעד בסיסי על מחשב זה, ונכפיל אותו במספר הצעדים המתבצעים עבור אורך הקלט המתאים.

3. שיפור יעילותו של אלגוריתם בקבוע

נבחן לדוגמה את הבעיה הזאת: נתון לנו כקלט טקסט עברי, ועלינו לבדוק האם קיימת בו האות 'ד'. נניח כי הטקסט מאוחסן בזיכרון המחשב וכי ניתן לגשת ישירות לכל אחת מאותיות הקלט. אפשר לפתור את הבעיה בעזרת אלגוריתם 1.

מצא-'ד'

{ האלגוריתם מודיע אם נמצאה האות 'ד' בטקסט. }

(1) קרא את האות הראשונה בטקסט.

(2) כל עוד לא הגענו לסוף הטקסט וגם האות הנוכחית אינה 'ד', בצע:

(2.1) קרא את האות הבאה בטקסט.

(3) אם הגענו לסוף הטקסט, אזי הודע "אין 'ד' בקלט", אחרת הודע "יש 'ד' בקלט".

אלגוריתם 1 - מציאת האות 'ד' בטקסט

האלגוריתם עובר על תווי הטקסט לפי סדרם באמצעות לולאה, ומבצע בכל פעם שתי בדיקות:

א. האם הגענו לסוף הטקסט.

ב. האם האות הנוכחית היא 'ד'.

תשובה חיובית על אחד מן התנאים - א או ב - תגרום לאלגוריתם להפסיק את ביצוע הלולאה. אם ביצוע הלולאה הופסק עקב קיום תנאי ב, יש להודיע כי האות 'ד' נמצאה; אם הלולאה הופסקה עקב קיום תנאי א, יש להודיע כי האות 'ד' לא נמצאה.

הצעד הבסיסי שעליו חוזר אלגוריתם זה כולל שלוש פעולות: קריאת האות הבאה בטקסט וביצוע בדיקות א ו-ב. אורך הקלט בבעיה זו הוא מספר התווים בטקסט. מהו, אם כן, מספר הצעדים שיבצע האלגוריתם עבור קלט באורך n תווים? ברור כי התשובה תלויה בטקסט הנתון. במקרה הגרוע ביותר, כאשר ה-'ד' מופיעה בסוף הטקסט או כאשר אין היא מופיעה בו כלל, יהיה מספר הצעדים שיבוצעו n.

נסכם ונאמר, כי עבור קלט באורך n יבצע אלגוריתם זה לכל היותר n פעמים את הצעד הבסיסי. אם נניח כי משך ביצוע צעד בסיסי באלגוריתם זה הוא c_1, הרי שמשך ביצוע הלולאה העיקרית יהיה $c_1 \cdot n$. פעולות (1) ו-(3), המבוצעות לפני הלולאה העיקרית באלגוריתם ואחריה, אורכות זמן קבוע כלשהו, שנסמנו d_1. לכן זמן הריצה של האלגוריתם הוא $c_1 \cdot n + d_1$.

האם אפשר לשפר את זמן הריצה של האלגוריתם שהצגנו למעלה? אם נתבונן באלגוריתם 1, נשים לב כי בכל חזרה על הלולאה האלגוריתם בודק אם הגיע כבר לסוף הטקסט. אפשר לחסוך בדיקה זו. נכניס את האות 'ד' לסוף הטקסט. כעת ברור, כי ביצוע הלולאה ייפסק *תמיד* בגלל תנאי ב, שהרי קיימת לפחות אות 'ד' אחת לפני סוף הטקסט. משום כך אפשר לוותר על בדיקת ההגעה לסוף הקלט *בתוך* הלולאה. אלגוריתם משופר יכניס תחילה 'ד' לסוף הטקסט, ולא יכיל בדיקה של התנאי הראשון. כאשר ביצוע הלולאה באלגוריתם המשופר יסתיים, נבדוק אם אנו בסוף הטקסט.

אם כן, סימן שמצאנו את ה-'ד' שהוספנו, ושלא הייתה 'ד' בטקסט המקורי. אחרת, מצאנו 'ד' בטקסט. אלגוריתם 2 מציג את הגרסה המשופרת.

```
מצא-'ד'-משופר
{ האלגוריתם מודיע אם נמצאה האות 'ד' בטקסט. }
(1) הוסף 'ד' בסוף הטקסט.
(2) קרא את האות הראשונה בטקסט.
(3) כל עוד האות הנוכחית אינה 'ד', בצע:
    (3.1) קרא את האות הבאה בטקסט.
(4) אם הגעת לסוף הטקסט, הודע "אין 'ד' בקלט", אחרת הודע "יש 'ד' בקלט".
(5) הורד 'ד' מסוף הטקסט.
```

אלגוריתם 2 - מציאת 'ד' בטקסט, גרסה משופרת

הצעד הבסיסי באלגוריתם המשופר מכיל הפעם שתי פעולות בלבד: קריאת האות הבאה בטקסט וביצוע בדיקה ב. מספר הצעדים שהאלגוריתם המשופר מבצע עבור טקסט באורך n נשאר עדיין לכל היותר n. בדומה לאלגוריתם 1, זמן הריצה של אלגוריתם 2 הוא $c_2 \cdot n + d_2$, כאשר c_2 הוא משך ביצוע הצעד הבסיסי המשופר, ואילו d_2 הוא קבוע המתאר את משך ביצוע פעולות (1), (2), (4) ו-(5).

השיפור באלגוריתם זה מתבטא בחיסכון של בדיקה אחת בעת ביצוע כל צעד בסיסי, כלומר הקבוע c_2 קטן מן הקבוע c_1. אם נניח כי כל אחת מהפעולות המבוצעות בתוך הלולאה אורכת אותו משך זמן, הרי c_1 אורך 3 יחידות זמן, שכן הצעד הבסיסי באלגוריתם הראשון מכיל שלוש פעולות. c_2, לעומת זאת, אורך 2 יחידות זמן (שכן חסכנו פעולה אחת בתוך הלולאה). נצפה אם כן, כי בעזרת השיפור נפחית כשליש מזמן הריצה.

אולם אליה וקוץ בה: הקטנו את המקדם של n, אך d_2 גדל לעומת d_1; נניח כי d_1 הוא 2 יחידות זמן (מבוצעות שתי פעולות לפני גוף הלולאה ואחריה), ו-d_2 הוא 4 יחידות זמן (נוספו שתי פעולות של הוספת 'ד' לסוף הטקסט וגריעתה). גודלם של שני מקדמים אלו אינו תלוי באורך הקלט. אמנם בטקסטים קצרים האלגוריתם המשופר יעיל פחות מן האלגוריתם הפשוט בשל הגדלת d_2, אולם ככל שאורך הטקסט גדל, קטנה חשיבותם של הקבועים d_1 ו-d_2, והיחס בין זמני הריצה מתקרב ליחס המצופה: 1.5. טבלה 2 מציגה את זמני הריצה של שני האלגוריתמים ומראה, שלכל אורך קלט הגדול מ-2 תווים, האלגוריתם המשופר מהיר יותר. כאשר ברור שהקלט שעוסקים בו גדול מספיק, אין צורך להתחשב בקבועים מסוג d בעת החישוב של סיבוכיות זמן הריצה.

אורך הקלט	זמן ריצת אלגוריתם פשוט $3 \cdot n + 2$	זמן ריצת אלגוריתם משופר $2 \cdot n + 4$	היחס המקורב בין זמני הריצה
1	5	6	0.83
3	11	10	1.1

1.21	14	17	5
1.33	24	32	10
1.48	204	302	100
1.5	2,004	3,002	1,000
1.5	60,004	90,002	30,000
1.5	6,000,004	9,000,002	3,000,000

טבלה 2 - השוואת זמני הריצה של אלגוריתם 1 ואלגוריתם 2

שיפור כזה ביעילות אלגוריתם נקרא **שיפור בקבוע** (improvement by factor), שכן היחס בין זמני הריצה של שני האלגוריתמים, כאשר אורך הקלט גדול, הוא בקירוב קבוע. (טבלה 2 מדגימה כיצד, ככל שגדל אורך הקלט, הולך היחס בין זמני הריצה ומתקרב לקבוע 1.5.) אם כן, ראינו דוגמה שבה השגנו שיפור בקבוע ביעילות האלגוריתם על ידי ניתוח פרטני של צעד בסיסי והפחתת מספר הפעולות שמהן הוא מורכב.

4. מקרים טובים, גרועים וממוצעים

שימו לב לכך שכאשר ספרנו את מספר הצעדים באלגוריתם, עסקנו במצב הגרוע מכול, שבו האות 'ד' לא נמצאת כלל בטקסט, או שהיא בדיוק האות האחרונה ואנו צריכים לעבור על פני כל הטקסט כדי למצוא אותה. מה יקרה אם האות 'ד' תהיה דווקא האות הראשונה בטקסט? אז נבצע רק צעד בודד. בכל מקרה אחר נבצע בין צעד בודד ל-n צעדים.

בחרנו לחשב את זמן ריצת האלגוריתם על קלטים שמתאימים להגדרת **המקרה הגרוע ביותר** (worst case). מדוע לא נחשב את זמן הריצה של האלגוריתם דווקא על פי **המקרה הממוצע** (average case)? הרי האלגוריתם ירוץ על רוב הקלטים באורך n בזמן שהוא פחות מהזמן הגרוע ביותר! שתי סיבות לכך:

ראשית, אם נדע מהו זמן הריצה במקרה הממוצע, האם נוכל להסיק מכך מה יהיה זמן הריצה של האלגוריתם על קלט נתון? כלומר, האם נוכל לדעת אם זמן זה יהיה מהיר יותר או אטי יותר מהזמן הממוצע? הדבר דומה למדידת הגובה של תלמידים בכיתה מסוימת. אם נדע שהגובה הממוצע הוא 1.65 מ', האם נוכל להסיק מכך משהו על גובהו של דני הלומד בכיתה זו? לעומת זאת, אם נדע שהתלמיד הגבוה ביותר בכיתה מתנשא לגובה של 1.80 מ', נוכל להגיד בוודאות כי גובהו של דני לא יעלה על 1.80 מ'. כך גם בחישוב זמן הריצה. כשנחשב את מספר הפעולות לפי המקרה הגרוע ביותר ונריץ את האלגוריתם על קלט כלשהו באורך n, מספר הפעולות שהוא יבצע לעולם לא יעלה על מספר הפעולות שחישבנו.

שנית, כדי לחשב את זמן הריצה של האלגוריתם במקרה הממוצע יש למצוא מהו הממוצע של זמני הריצה על פני כל הקלטים האפשריים באורך n. חישוב הזמן במקרה זה קשה הרבה יותר מחישוב הזמן במקרה הגרוע ביותר.

יש עניין בחישוב מדד הזמן הן למקרה הגרוע ביותר והן למקרה הממוצע, אולם מהסיבות שצוינו למעלה נסתפק לרוב בניתוח היעילות של המקרה הגרוע בלבד.

? *אפשר להגדיר זמן ריצה גם למקרה הטוב ביותר. האם הוא מדד טוב ליעילות של אלגוריתם?*

מספר הפעמים, שהאלגוריתם מבצע את הצעד הבסיסי במהלך ריצתו, במקרה הגרוע ביותר, הוא פונקציה של אורך הקלט. פונקציה זו הנקראת **פונקציית זמן הריצה** של האלגוריתם היא המדד ליעילותו.

5. שיפור בסדר גודל

נחזור לדוגמת האלגוריתם לחיפוש 'ד'. שני האלגוריתמים (הפשוט והמשופר) מממשים, למעשה, אותו רעיון אלגוריתמי. הרעיון הבסיסי הוא מעבר על כל הטקסט והשוואת האותיות שבו לאות המבוקשת; רעיון זה נקרא **חיפוש סדרתי** (linear search). גם האלגוריתם המשופר נוהג כך, אלא שהוא מצליח להוריד השוואה אחת בכל ביצוע חוזר של הלולאה (בגלל הוספת ה-'ד' בסוף הטקסט). כך מושג שיפור בקבוע ביעילות האלגוריתם.

שיפור בקבוע הוא אמנם חשוב, אבל כפי שנראה מיד, לעתים גישה אחרת לפתרון בעיה - רעיון אלגוריתמי שונה - יכולה להוביל לשיפור ניכר יותר.

הבעיה שננתח היא בעיית חיפוש בספר טלפונים - מערך שכל תא בו מכיל רשומה המורכבת משני שדות: שם ומספר טלפון. כדי למצוא את מספר הטלפון של פלוני עלינו לבדוק האם שמו מופיע במערך, והיכן. לשם פשטות נניח כי השמות במערך שונים זה מזה. בעיה זו דומה במהותה לבעיית חיפוש 'ד', וניתן לפתור אותה באופן דומה.

נניח כעת כי המערך **ממוין** לפי שדה השם של הרשומות, בסדר מילוני (לקסיקוגרפי), כלומר בסדר א"ב של השמות. במקרה זה אין צורך להמשיך את החיפוש עד לסוף המערך. נוכל להפסיק את החיפוש כאשר ניתקל בשם הגדול מבחינת הסדר המילוני מהשם שאנו מחפשים.

? *מדוע אפשר להפסיק את החיפוש במצב המתואר?*

? *כיצד יהיה האלגוריתם לחיפוש סדרתי במערך ממוין שונה מהאלגוריתמים לחיפוש 'ד' שהופיעו בפרק?*

? *האם יש שינוי בניתוח זמן הריצה במקרה הגרוע ביותר, בשל העובדה שהמערך ממוין?*

את העובדה שהמערך ממוין אפשר לנצל בצורה טובה יותר באמצעות רעיון אלגוריתמי חדש: אם גודל המערך הוא אלף תאים, נתחיל לחפש דווקא בתא ה-500, ואז לפנינו שלוש אפשרויות:

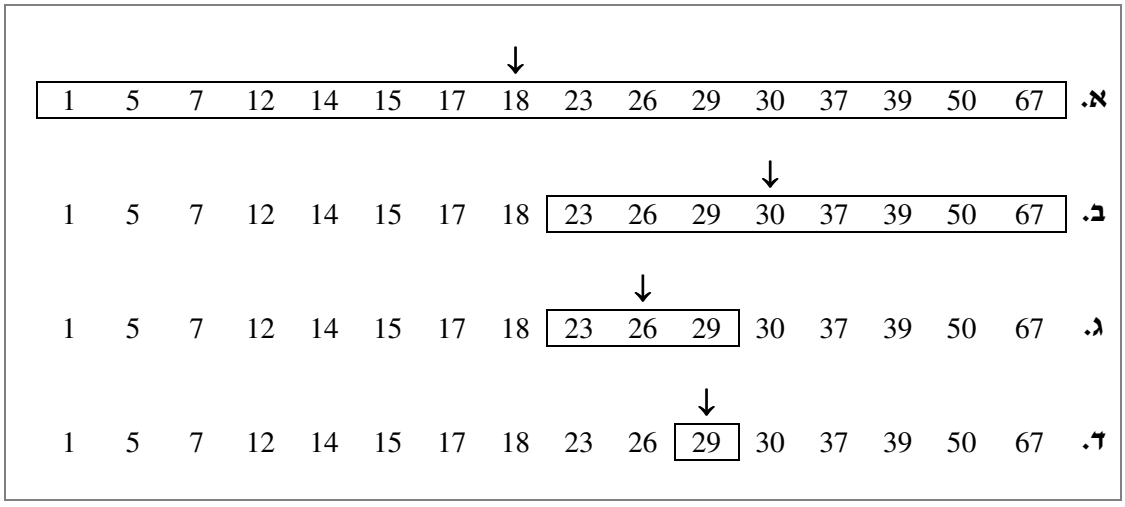
א. השם מופיע בתא שחיפשנו - הצלחנו, החיפוש הסתיים.

ב. השם שמופיע בתא זה גדול מילונית מהשם שאנו מחפשים - אנו יודעים שאין טעם לחפש בחציו השני של המערך, ונותר לנו לחפש את השם המבוקש רק בתחום הכולל את התאים מ-1 עד 499.

ג. השם שמופיע בתא קטן מילונית מהשם המבוקש - הפעם נמשיך לחפש רק בתאים 501 עד 1,000.

נניח שהתקיים תנאי ב. עכשיו, כשאנו צריכים לחפש רק בחלקו הראשון של המערך, נחפש באופן דומה בתא שמספרו 250 ושוב נעמוד בפני שלוש אפשרויות דומות. האלגוריתם ייפסק כאשר נמצא את התא המבוקש, או כאשר לא יהיה אפשר עוד לחלק את תחום החיפוש - ואז ברור כי השם המבוקש אינו נמצא במערך.

הרעיון האלגוריתמי החדש מאפשר לנו "לחתוך" חצי מתחום החיפוש בכל שלב. אלגוריתם חיפוש זה נקרא **חיפוש בינרי** (binary search). איור 2 מתאר חיפוש בינרי של המספר 29 במערך ממוין שבו 16 איברים. המסגרת מציינת את תחום החיפוש בכל שלב, והחץ מציין את התא ה"אמצעי" בתחום זה.



איור 2 - חיפוש בינרי של המספר 29 במערך ממוין

ננתח את יעילותם של שני האלגוריתמים לחיפוש. בשני המקרים הצעד הבסיסי כולל את ההחלטה אם להמשיך בחיפוש ואת חישוב מספר התא הבא שבו ייערך החיפוש.

בחיפוש סדרתי הצעד הבסיסי מכיל את הפעולות האלה:

א. בדיקה האם התא הנוכחי מכיל את השם המבוקש או שערכו המילוני של השם בתא גדול מן השם המבוקש.

ב. בדיקה האם התא הנוכחי הוא סוף המערך.

ג. אם התשובה לבדיקות א ו-ב שלילית יש לעבור לתא הבא במערך.

בחיפוש בינרי הצעד הבסיסי מכיל את הפעולות האלה:

א. בדיקה האם התא הנוכחי מכיל את השם המבוקש.

ב. בדיקה האם התחום בגודל 1 (לא ניתן להמשיך בחלוקת התחום).

ג. אם התשובה לבדיקות א ו-ב שלילית יש לחשב את מספר התא האמצעי בתחום ולעבור אליו.

נחשב את מספר הצעדים המתבצעים באלגוריתם לחיפוש בינרי בעת חיפוש איבר במערך שבו אלף איברים. כאמור, בכל שלב אנו חוצים את המערך, לכן הגדלים של קטעי המערך שבהם נבצע את החיפוש הם בקירוב: 1,000, 500, 250, 125, 63, 32, 16, 8, 4, 2, 1. במקרה הגרוע נסיים את החיפוש אחרי 10 צעדים (אז יהיה תחום החיפוש בגודל תא בודד), ונוכל להודיע אם השם נמצא או אינו נמצא במערך. מצב זה ודאי עדיף על 1,000 הצעדים המתבצעים במקרה הגרוע בחיפוש סדרתי.

טבלה 3 מסכמת את השיפור שסקרנו לעיל. טבלה זו מציגה את התלות של מספר הצעדים הבסיסיים באורכי קלט שונים. כפי שניתן להיווכח, הפעם השיפור מהותי הרבה יותר: היחס בין מספר הצעדים המתבצעים בשני האלגוריתמים הולך וגדל ככל שגדל אורך הקלט.

אורך הקלט	מספר צעדי החיפוש בחיפוש סדרתי	מספר צעדי החיפוש בחיפוש בינרי	היחס בין מספר צעדי החיפוש באלגוריתמים
10	10	4	כ-2.5
100	100	7	כ-14
1,000	1,000	10	כ-100
10,000	10,000	14	כ-714
100,000	100,000	17	כ-5,883
מיליון	מיליון	20	כ-50,000

טבלה 3 - מספר צעדי החיפוש בחיפוש סדרתי ובחיפוש בינרי

כאשר דנו בשיפור בקבוע, משך הזמן שארך צעד בסיסי עניין אותנו מאוד. כאן - חשיבותו פחותה. אף אם משך צעד באלגוריתם לחיפוש בינרי גדול בהרבה בהשוואה למשך צעד באלגוריתם לחיפוש סדרתי, הרי שיתרון זה זניח החל מאורך קלט מסוים. התבוננו בטבלה 4, שם הנחנו כי צעד בחיפוש סדרתי אורך יחידת זמן אחת, ואילו צעד בחיפוש בינרי דורש משך זמן גדול בהרבה: 100 יחידות. (ההנחות באשר למשך הזמן שאורך צעד בסיסי בכל אחד מהמקרים שרירותיות, ונועדו להדגים עד כמה מהותי השינוי בסדר הגודל. בפועל משך הצעד הבסיסי בחיפוש בינרי הוא לא יותר מפי 5 או 10 ממשך צעד בחיפוש סדרתי.) מהתבוננות בטבלה מתברר שאמנם עד לאורך קלט 1,000 החיפוש הסדרתי עדיף בשל היחס בין משך הצעדים הבסיסיים, אולם החל מ-1,000 יש עדיפות ברורה לחיפוש הבינרי. הסיבה לכך היא שהיחס בין *משך* הצעדים הבסיסיים בשני האלגוריתמים קבוע, ואילו היחס בין *מספר* הצעדים הבסיסיים הולך וגדל ככל שאורך הקלט גדל.

אורך הקלט	זמן ריצה של אלגוריתם חיפוש סדרתי (אורך צעד בסיסי - יחידת זמן אחת)	זמן ריצה של אלגוריתם חיפוש בינרי (אורך צעד בסיסי - 100 יחידות זמן)	היחס בין זמני הריצה
10	10	400	כ-0.025
100	100	700	כ-0.14
1,000	1,000	1,000	1
10,000	10,000	1,400	כ-7.14
100,000	100,000	1,700	כ-58.8
מיליון	מיליון	2,000	כ-500

טבלה 4 - השוואת זמני הריצה של חיפוש בינרי וחיפוש סדרתי

לשיפור מסוג זה נקרא **שיפור בסדר גודל** (improvement in order of magnitude): היחס בין זמני הריצה של האלגוריתמים *הולך ומשתפר* ככל שאורך הקלט גדל. (השוו זאת לשיפור בקבוע המוצג בטבלה 2, שם היחס בין זמני הריצה *קבוע*.)

5.1. סדר גודל

ראינו עד עתה שני סוגים של שיפורים בפונקציות זמני הריצה של אלגוריתמים: שיפור בקבוע ושיפור בסדר גודל. ראינו גם כי האחרון משמעותי הרבה יותר. בגלל סיבה זו, בסיווג ראשוני של זמני ריצה של אלגוריתמים נרצה "להתעלם" מהבדלים דקים בין שתי פונקציות של זמן ריצה ולקבצן יחדיו תחת קורת גג אחת. כך, למשל, האלגוריתמים הפשוט והמשופר לחיפוש 'ד' והאלגוריתם לחיפוש סדרתי - כולם דורשים במקרה הגרוע ביצוע של n צעדי חיפוש. ככל שננסה לדקדק בחישוב זמן הריצה, נקבל בסופו של דבר פונקציה מהצורה $f(n) = a \cdot n + b$, הידועה בשם פונקציה לינארית, אשר בה a ו-b קבועים כלשהם. היחס בין כל שתי פונקציות מסוג זה יהיה קבוע עבור n גדול מספיק. לכן נאמר כי פונקציות זמני הריצה של אלגוריתמים אלו הן מ**סדר גודל**

לינארי, או ש**סיבוכיות זמן הריצה** שלהן היא לינארית. מכיוון שהפונקציה הלינארית הפשוטה ביותר היא $f(n) = n$, נהוג לסמן סיבוכיות לינארית בסימון ***O(n)***, קרי: "או-גדול של n".

באופן כללי נגיד כי שתי פונקציות זמן ריצה הן מאותו סדר גודל, אם היחס בין ערכיהן מתקרב ליחס קבוע עבור n מספיק גדול. קיימות כמה משפחות שימושיות של סדרי גודל, ונעמוד עליהן בהמשך.

? *האם הפונקציה $f(n) = n^2$ היא מסדר גודל לינארי?*

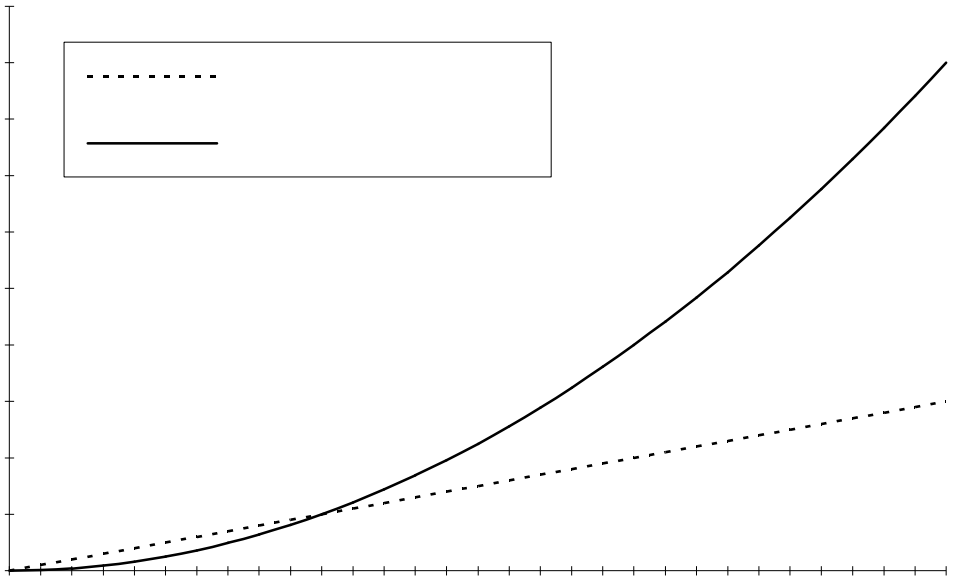
? *האם הפונקציות $f(n) = n^2/100$ ו-$g(n) = n^2/10$ הן מאותו סדר גודל?*

בדומה לסדר גודל לינארי, ניתן להגדיר גם **סדר גודל ריבועי**, המסומן ב-$O(n^2)$. זוהי משפחת כל הפונקציות שהיחס ביניהן לבין הפונקציה הריבועית הפשוטה ביותר, $f(n) = n^2$, הוא קבוע עבור n גדול מספיק. כשאנו אומרים שפונקציית זמן הריצה של אלגוריתם היא מסדר גודל ריבועי, פונקציה זו יכולה להיות n^2, $5n^2+6$, $5n^2+100n+6$ או $5{,}000n^2$. גם אם זמן ריצתו של האלגוריתם הוא רק חלק מ-n^2, למשל $n^2/6$, עדיין נאמר שזמן הריצה שלו הוא $O(n^2)$. אנו כוללים את כל הפונקציות האלה במשפחה אחת.

רעיון זה נראה אבסורדי במקצת: האם ייתכן שנשכן אלגוריתם שזמן ריצתו n^2 ואלגוריתם שזמן ריצתו $n^2/100$ תחת קורת גג אחת?

איור 3 מבהיר כי כאשר אנו משווים זמני ריצה אלו לזמני ריצה לינאריים, *סדר הגודל* הוא המכריע, וכי עבור n גדול מספיק הקבועים הופכים לחסרי משמעות. כך, אף על פי שהמקדם של הפונקציה הלינארית באיור הוא 100, ואילו המקדם של הפונקציה הריבועית הוא 1/100 (קטן פי 10,000), הרי שהחל מ-n מסוים ערך הפונקציה הריבועית גדול מערך הפונקציה הלינארית. היחס בין שתי הפונקציות הולך וגדל גם הוא, ככל שגדל ערכו של n.

אם לפתרון בעיה נתונים לנו שני אלגוריתמים בעלי סיבוכיות זמן ריצה שונה, האם נעדיף תמיד את האלגוריתם בעל הסיבוכיות הנמוכה? התבוננו באיור 3, המציג פונקציות זמני ריצה מסדר גודל לינארי וריבועי. עבור $n > 10{,}000$ אכן מתקיים: $n^2/100 > 100n$, אבל עבור $n < 10{,}000$ מתקיים ההפך: $n^2/100 < 100n$. בשל כך אפשר לומר, שאם נדע שאורכי הקלטים הצפויים הם לכל היותר 10,000, נעדיף אלגוריתם שזמן ריצתו $n^2/100$, אף על פי שסיבוכיות זמן ריצתו גדולה יותר מ-100n. אם ידוע לנו כי חלק ניכר מהקלטים הצפויים יהיו ארוכים מ-10,000, נעדיף את האלגוריתם הלינארי. מובן כי כדי להגיע להחלטות כאלו בעניין כדאיות השימוש באלגוריתמים בעלי סיבוכיות ריצה מסדרי גודל שונים, אנו נדרשים להערכות באשר למשך הזמן הדרוש לביצוע הצעד הבסיסי.



איור 3 - השוואה בין פונקציה לינארית לפונקציה ריבועית

לסיום הדיון במושג סדר גודל נציין, כי נהוג לסמן את סדר הגודל של פונקציה קבועה, שאינה תלויה באורך הקלט (כגון משך ביצועו של צעד בסיסי באלגוריתם), ב-$O(1)$.

5.2. זמן הריצה של חיפוש בינרי

ראינו בסעיפים הקודמים כי סיבוכיות זמן הריצה של אלגוריתמים לחיפוש סדרתי היא $O(n)$. ראינו גם כי סיבוכיות זמן הריצה של אלגוריתם לחיפוש בינרי שונה וטובה יותר, אך מהי בדיוק? בסעיף זה נחשב אותה.

לשם כך עלינו לספור כמה צעדי חיפוש מתבצעים באלגוריתם לחיפוש בינרי על מערך בגודל n תאים. חיפוש בינרי מבוסס על כך שבכל צעד אנחנו מקטינים את תחום החיפוש בחצי. במקרה הגרוע ביותר נעצור כשלא יהיה אפשר להקטין יותר את תחום החיפוש, דהיינו כשגודלו יהיה 1. כדי לדעת כמה צעדים דרושים, עלינו לחשב כמה פעמים ניתן לחלק את n ב-2 כדי לקבל 1. לשם פשטות נניח שגודל המערך הוא בדיוק חזקה של 2.

אנו מחפשים k אשר יקיים:

$$n\underbrace{/\,2\,/\,2\,/\,2\,/\,2/\ldots\ldots/2}_{k\ \text{פעמים}} = 1$$

או:

$$n/(\underbrace{2\cdot 2\cdot 2\cdot 2\cdot\ldots\ldots\cdot 2}_{k\ \text{פעמים}}) = 1$$

נעביר אגפים ונקבל:

$$1\cdot\underbrace{2\cdot 2\cdot 2\cdot 2\cdot\ldots\ldots\cdot 2}_{k\ \text{פעמים}} = n$$

כלומר:

$$2^k = n$$

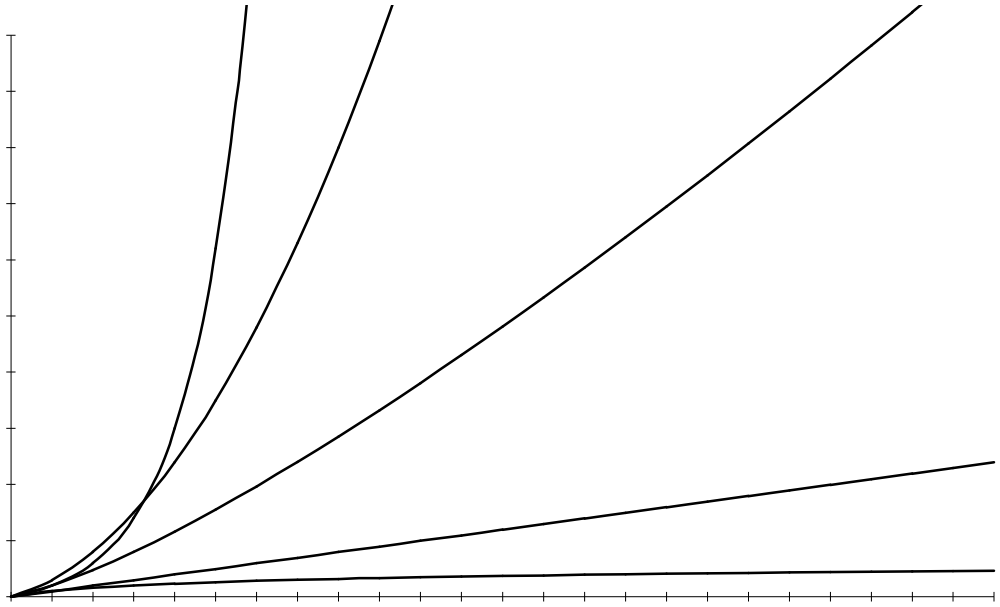
ומכאן:

$$k = \log_2 n$$

קיבלנו שזמן הריצה של חיפוש בינרי הוא מסדר גודל לוגריתמי, כלומר $O(\log n)$. לפיכך סיבוכיות זמן הריצה של חיפוש בינרי לעומת חיפוש סדרתי משתפרת מ-$O(n)$ ל-$O(\log n)$.

? *בדקו מהו מספר הצעדים המקסימלי בחיפוש בינרי ברשימות שבהן 8, 13, 16 ו-90 איברים. מה ניתן להסיק מכך על סיבוכיות החיפוש בסדרה שאורכה אינו חזקה של 2?*

לסיכום, נתבונן באיור 4 המציג פונקציות מסדרי גודל שונים. הדוגמאות מדגישות את השיפור בסדר הגודל. שימו לב להבדלים הגדלים והולכים ככל ש-n גדל.



איור 4 - פונקציות מסדרי גודל שונים

5.3. דוגמה לשיפור בסדר גודל: שכיחות של זוג אותיות

נבחן דוגמה נוספת לשיפור בסדר גודל. הפעם ברצוננו למצוא בטקסט האגור בזיכרון, מהו הזוג השכיח של אותיות עוקבות (כלומר, הזוג שמופיע בטקסט הכי הרבה פעמים). אלגוריתם אפשרי לפתרון הבעיה הוא אלגוריתם 3. נניח כי אורך הטקסט הוא n. האלגוריתם עובר זה אחר זה על כל זוגות האותיות המופיעות בטקסט (יש בדיוק n-1 זוגות כאלה), מחשב את שכיחותו של הזוג, ושומר בכל שלב את זוג האותיות בעל השכיחות המרבית עד אותו שלב.

מצא-זוג-שכיח1-

{ האלגוריתם מוצא את זוג האותיות העוקבות השכיח בטקסט ומדפיס אותו.

pair ו-max אוגרים את הנתונים על הזוג השכיח ביותר ברגע נתון.

p1 ו-p2 אוגרים זוגות של אותיות עוקבות. }

(1) max ← 0.

(2) עבור כל זוג אותיות עוקבות p1 המופיע בטקסט, בצע:

(2.1) count ← 0.

(2.2) עבור כל זוג אותיות עוקבות p2 המופיע בטקסט, בצע:

{חישוב השכיחות של p1.}

(2.2.1) אם p1 = p2 אזי count ← count + 1.

(2.3) אם count > max אזי בצע:

{ השכיחות של p1 גדולה מהשכיחות המקסימלית.}

(2.3.1) max ← count.

(2.3.2) pair ← p1.

(3) הדפס את pair.

אלגוריתם 3 - מציאת זוג אותיות שכיח בטקסט

מהי סיבוכיות זמן הריצה של אלגוריתם 3? תחילה עלינו למצוא את הצעד הבסיסי באלגוריתם. נחפש אוסף פעולות, החוזר על עצמו בעת ביצוע האלגוריתם ואינו תלוי באורך הקלט. "מועמד" להיות צעד בסיסי הוא גוף הלולאה שבשורה (2), אולם ביצועו אינו אורך זמן קבוע. בגוף הלולאה, בשורה (2.2), מקוננת לולאה שמשך ביצועה תלוי באורך הקלט. גוף הלולאה המקוננת מבוצע n-1 פעמים בכל ביצוע של הלולאה החיצונית. שאר הפעולות שבגוף הלולאה החיצונית מבוצעות רק פעם אחת במהלך בכל ביצוע חוזר, ולכן הן זניחות בחישוב הסיבוכיות. נוכל להכליל ולומר שכאשר אלגוריתם מכיל לולאות מקוננות, הצעד הבסיסי הוא אוסף הפעולות הכלולות בלולאה הפנימית ביותר, שמשך ביצוען אינו תלוי באורך הקלט.

הצעד הבסיסי באלגוריתם, גוף הלולאה המקוננת, מבוצע פעם אחת עבור כל זוג בטקסט בכל חזרה של הלולאה החיצונית. גוף הלולאה החיצונית מבוצע גם הוא פעם אחת עבור כל זוג בטקסט. מספר הזוגות הוא n-1, ולכן מספר הפעמים שמבוצע הצעד הבסיסי הוא: $(n-1)\cdot(n-1) = n^2-2n+1$, וסיבוכיות זמן הריצה של האלגוריתם היא $O(n^2)$.

נשים לב כי אלגוריתם זה מבצע לפעמים צעדים מיותרים: אם הטקסט שהאלגוריתם עובר עליו ארוך מספיק, סביר מאוד שזוג אותיות, שכבר חישבנו את שכיחותו, יופיע שוב בטקסט (כך יקרה למעשה לכל זוג ששכיחותו גדולה מ-1), לכן נראה הגיוני לחפש אלגוריתם שימנע חישובים מיותרים אלו.

נציג אלגוריתם אחר הפותר את הבעיה. אלגוריתם זה משתמש במערך דו-ממדי שהאינדקסים לשורותיו ולעמודותיו הם המספרים הסידוריים של האותיות בסדר הא"ב (בשפה העברית גודלו יהיה 22 x 22 תאים). כל תא [i, j] במערך מכיל את מספר הופעותיו של זוג האותיות המתאים. נעבור על כל זוגות האותיות בטקסט, ולכל זוג נוסיף 1 לתא המתאים במערך. פעולה זו דורשת מאתנו מעבר אחד בלבד על הטקסט. כדי למצוא את זוג האותיות השכיח, נסרוק את המערך ונמצא את המספר המקסימלי בו. אלגוריתם 4 הוא האלגוריתם המשופר.

מצא-זוג-שכיח2-

{ האלגוריתם מוצא את זוג האותיות העוקבות השכיח בטקסט ומדפיס אותו.
arr הוא מערך עזר דו-ממדי בגודל 22 x 22, האוגר את השכיחויות. }
(1) אפס את המערך arr.
(2) עבור כל זוג אותיות עוקבות בטקסט, הוסף 1 למקום המתאים במערך arr.
(3) סרוק את המערך arr ומצא את האיבר המקסימלי. האינדקסים של האיבר הם זוג האותיות השכיח.

אלגוריתם 4 - אלגוריתם משופר למציאת זוג אותיות שכיח

מהי סיבוכיות זמן הריצה של אלגוריתם זה? לולאה (2) מתבצעת n-1 פעמים, וכל ביצוע דורש זמן קבוע. בסך הכול דורשת הלולאה $O(n)$ צעדים. פעולות (1) ו-(3) דורשות מעבר על המערך arr; מספר הצעדים המתבצעים בכל אחת מהן הוא קבוע ואינו תלוי באורך הקלט. לפיכך, משך הביצוע של האלגוריתם כולו יהיה מהצורה $c \cdot n+d$, וסיבוכיותו של האלגוריתם תהיה $O(n)$. (מספר פעולות קבוע, גדול ככל שיהיה, אינו משפיע על סדר הגודל שכן אינו פונקציה של אורך הקלט.)

זהו שיפור בסדר גודל לעומת האלגוריתם הקודם שראינו. אמנם לצורך כל סריקה של המערך הדו-ממדי arr יש לבצע קבוע של 484 פעולות (22^2), והאלגוריתם יבצע אותן אפילו על טקסט שאורכו 1, אבל שוו בנפשכם שופט הבודק האם המילה "חף" היא השכיחה במאגר פסקי הדין של בית המשפט העליון. מאגר זה מכיל כחמישים מיליון מילים. אם נעשה שימוש באלגוריתם הראשון, שסיבוכיותו $O(n^2)$, יתייאש השופט מהר מאוד מלמצוא את מבוקשו במשך הקדנציה שלו. האלגוריתם השני, לעומת זאת, יאפשר לסיים את החיפוש בזמן סביר. (ראו איור 3 המשווה בין סדר גודל ריבועי ללינארי.)

5.4. דוגמה נוספת לשיפור בסדר גודל: מיון מערך

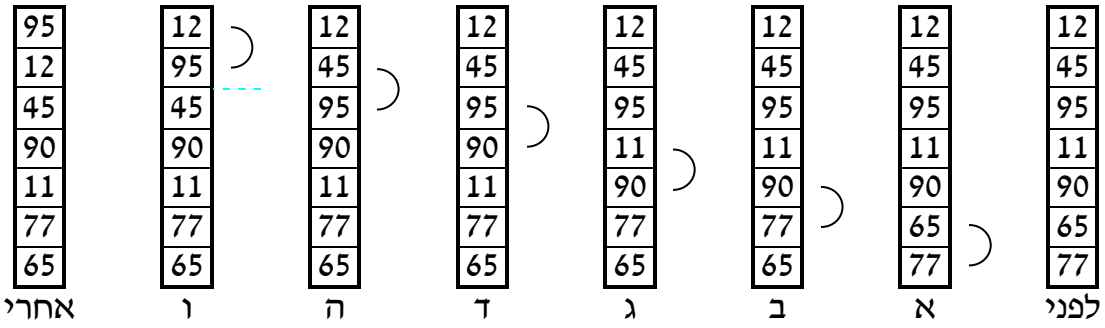
כדוגמה נוספת לשיפור בסדר הגודל של סיבוכיות זמן ריצה נבחן שני אלגוריתמים למיון מערך. הצורך במיון יעיל עולה במגוון רחב ביותר של שימושים במחשבים. כך, למשל, באחד הסעיפים הקודמים הוצג אלגוריתם לחיפוש בינרי בספר טלפונים ממוין. אם רשימת השמות ומספרי הטלפון שבידינו אינה ממוינת, לא ניתן להפעיל את האלגוריתם ולשפר בסדר גודל את זמן הריצה.

כפי שראינו, האיברים המאוחסנים במערך אינם בהכרח מספרים. כאשר ברצוננו למיין מערך של רשומות, עלינו להחליט תחילה לפי איזה **מפתח** ימוין המערך. המפתח הוא אחד השדות ברשומה

שניתן לסדר את ערכיו. בספר הטלפונים המפתח הוא השדה המכיל את שם המשתמש. היה אפשר כמובן לקבוע, כי המפתח הוא מספר הטלפון עצמו, ואז מיון ספר הטלפונים היה מניב מערך המסודר לפי מספרי הטלפון.

5.4.1. מיון-בועות

האלגוריתם הראשון שנבחן ידוע בשם **מיון-בועות** (bubble sort). המיון מתבסס על סדרת צעדים, המגדירה מעבר על המערך: נסרוק את המערך איבר אחר איבר. בכל פעם שנמצא כי האיבר הנוכחי גדול מהאיבר שבא אחריו, נחליף את זוג האיברים. בסיום המעבר האיבר הגדול במערך יהיה במקום האחרון. אם נחשוב על המערך כאילו הוא ניצב על ראשו, המעבר יגרום לאיבר הגדול ביותר "לבעבע" לסוף המערך. איור 5 מדגים מעבר על מערך שבו 7 איברים. המערך לפני הסריקה מוצג מצד ימין. בשלב א מושווים האיבר הראשון 77 והאיבר שבא אחריו 65, ומכיוון ש: $65 < 77$ הם מוחלפים. בשלב ב נבחנים האיברים 77 ו-90, ומכיוון שהאחרון הוא הגדול מבין השניים, לא מבוצעת החלפה, וכך הלאה עד לסוף המערך. עם סיום המעבר "בעבע" האיבר הגדול במערך, 95, למקומו בסוף הרשימה.



איור 5 - סריקה אחת של מערך שבו 7 איברים באמצעות מיון-בועות

לאחר המעבר שתיארנו המערך מסודר "קצת יותר", שכן האיבר הגדול נמצא בסופו, אך שאר המערך נותר לא ממוין. ניתן לבצע מעבר דומה, הפעם על n-1 האיברים הראשונים (שכן האיבר האחרון כבר נמצא במקומו). בסוף מעבר זה שני האיברים האחרונים נמצאים במקומם: האיבר הגדול במערך נשאר בסופו, והאיבר השני בגודלו "בעבע" לתא שמתחתיו. נוכל להמשיך באותו אופן על ידי ביצוע מעברים נוספים על מספר הולך וקטן של איברים. בסיומו של כל מעבר גדל באחד החלק הממוין של המערך. אלגוריתם 5 הוא האלגוריתם למיון-בועות.

מיון-בועות *(A,n)*

{ האלגוריתם ממיין בסדר עולה את המערך A שבו n איברים.}

(1) עבור k מ-(n-1) ועד 1, בצע:

(1.1) עבור i מ-1 ועד k, בצע:

(1.1.1) אם $A[i+1] < A[i]$, החלף את מקומותיהם.

אלגוריתם 5 - מיון-בועות

ננתח את זמן הריצה של האלגוריתם. הצעד הבסיסי שמבצע האלגוריתם הוא השוואת שני איברים עוקבים במערך והחלפתם, אם הראשון גדול מהשני. במעבר הראשון מבוצעים n-1 צעדים (לאיבר האחרון במערך אין איבר עוקב, ולכן אין צורך להפעיל עליו צעד נוסף). במעבר השני מבוצעים n-2 צעדים וכן הלאה, עד לסיום האלגוריתם. מספר הצעדים הכולל הוא:

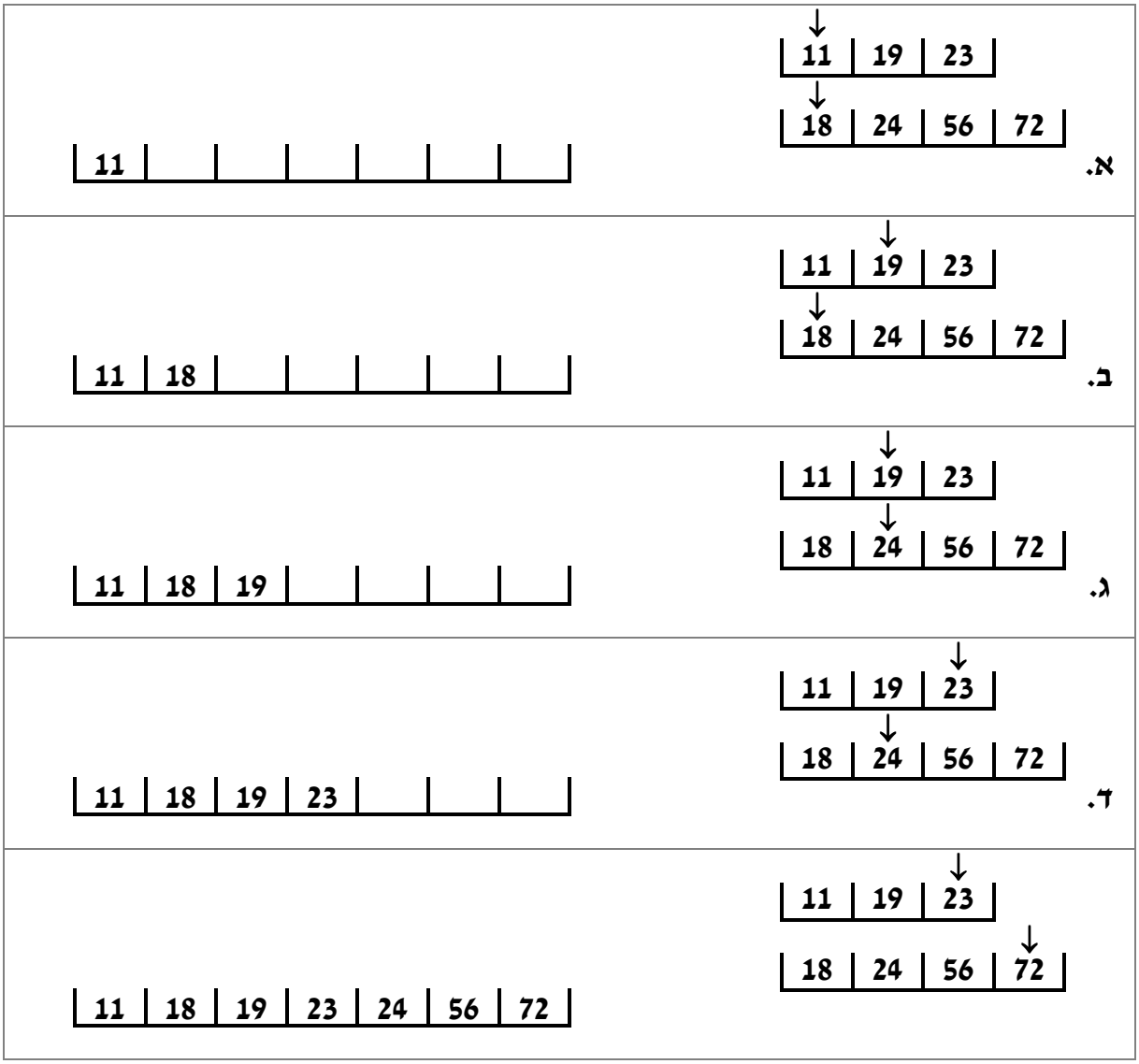
$$(n-1) + (n-2) + (n-3) + \ldots + 2 + 1 = \frac{n \cdot (n-1)}{2} = \frac{1}{2} \cdot n^2 - \frac{1}{2} \cdot n$$

אם כן, סיבוכיות זמן הריצה של מיון-בועות היא $O(n^2)$. ישנם אלגוריתמים נוספים למיון מערך שהם בעלי סיבוכיות זמן ריצה ריבועית. נשאלת השאלה, האם יש אלגוריתמים יעילים יותר?

5.4.2. מיון-מיזוג

תחילה נבחן את הבעיה הזאת: בידינו שני מערכים: arr1 ו-arr2. כל אחד מהם ממוין בסדר עולה, ואנו רוצים ליצור מערך שלישי arr3, שיכיל את כל האיברים שבשני המערכים הראשונים, כשהם ממוינים בסדר עולה. נציג אלגוריתם **מיזוג** הפותר את הבעיה. נציב שני מצביעים על תחילתו של כל אחד מהמערכים arr1 ו-arr2. נשווה את ערכיהם של האיברים בתאים שעליהם מורים המצביעים (התאים הראשונים בכל אחד מהמערכים). במקום הראשון במערך arr3 נשמור את האיבר שערכו הוא הנמוך מבין השניים, ונקדם את המצביע שסימן איבר זה לתא הבא במערך. נחזור על הצעד לגבי הערכים החדשים של המצביעים. נמשיך באותו אופן עד שאחד המצביעים יגיע לסוף המערך. אז, כמובן, לא נקדם אותו, ונעתיק ל-arr3 לפי הסדר את האיברים הנותרים במערך השני.

איור 6 מדגים את תהליך המיזוג של שני מערכים ממוינים שבהם שלושה וארבעה איברים. בשלב א מושווים התאים הראשונים בשני המערכים. ערכו של האיבר במערך העליון קטן יותר, ולכן הוא מועבר למערך הממוזג. המצביע על המערך הראשון מקודם ב-1. בשלב ב מושווה התא השני במערך העליון לתא הראשון במערך התחתון. הפעם האיבר במערך התחתון קטן יותר, ולכן הוא זה שמועבר למערך הממוזג. באופן דומה מתקדם האלגוריתם עד שלב ד, שבו מגיע המצביע העליון עד קצה המערך. משלב זה מועתקים האיברים הנותרים מהמערך התחתון, לפי הסדר, למערך הממוזג.



איור 6 - שלבי המיזוג של שני מערכים באורך 3 ו4- איברים למערך אחד באורך 7

הצעד הבסיסי באלגוריתם מיזוג כולל, במקרה הגרוע, השוואה בין שני ערכים במערך, העתקת איבר למערך שלישי ועדכון הערכים של המצביעים (הצעדים המתבצעים לאחר שהגענו לסוף אחד המערכים אינם כוללים את פעולת ההשוואה). מספר הצעדים המבוצע הוא כאורכו של המערך הממוזג, וסיבוכיות זמן הריצה (כתלות באורך המערך הממוזג) היא $O(n)$.

כיצד אפשר להשתמש בשגרת המיזוג שהגדרנו כדי למיין מערך A באורך n? אם נחלק את A לשני תת-מערכים שווים פחות או יותר בגודלם, ונמיין כל אחד מהם, נוכל לאחר מכן להשתמש בשגרה שהגדרנו על מנת למזג שני חלקים אלה למערך ממוין.

נשאלת השאלה, מהי דרך המיון שנשתמש בה כדי למיין את שני חלקי A? אולי נשתמש, למשל, באלגוריתם למיון-בועות שהכרנו בסעיף הקודם? ניתן לעשות זאת, אולם ניתוח פשוט יראה כי לא נשיג שיפור ביעילות האלגוריתם ; היעילות תישאר ריבועית.

המפתח לשיפור במקרה זה הוא שימוש ברקורסיה. את שני חלקי המערך נוכל למיין רקורסיבית: נחלק כל אחד מחלקים אלה לשני חלקים, נמיין שוב כל חלק בצורה רקורסיבית, ונחזור ונמזגם לשני תת-מערכים ממוינים. בשלב כלשהו נגיע לתת-מערך בגודל 1. תת-מערך זה ממוין, ולכן תיעצר הרקורסיה. אלגוריתם 6 מציג בצורה סכמטית שיטת מיון זו המכונה **מיון-מיזוג** (merge sort).

מיון-מיזוג *(A, n)*

{ האלגוריתם ממיין בסדר עולה את המערך A שבו n איברים.

המערכים arr1 ו-arr2 הם מערכי עזר, המכילים איברים מטיפוס האיברים ב-A.}

(1) *אם $n > 1$, בצע:*

(1.1) *חלק את המערך A לשני חלקים arr1 ו-arr2,*

המכילים כל אחד מחצית מאיברי המערך.

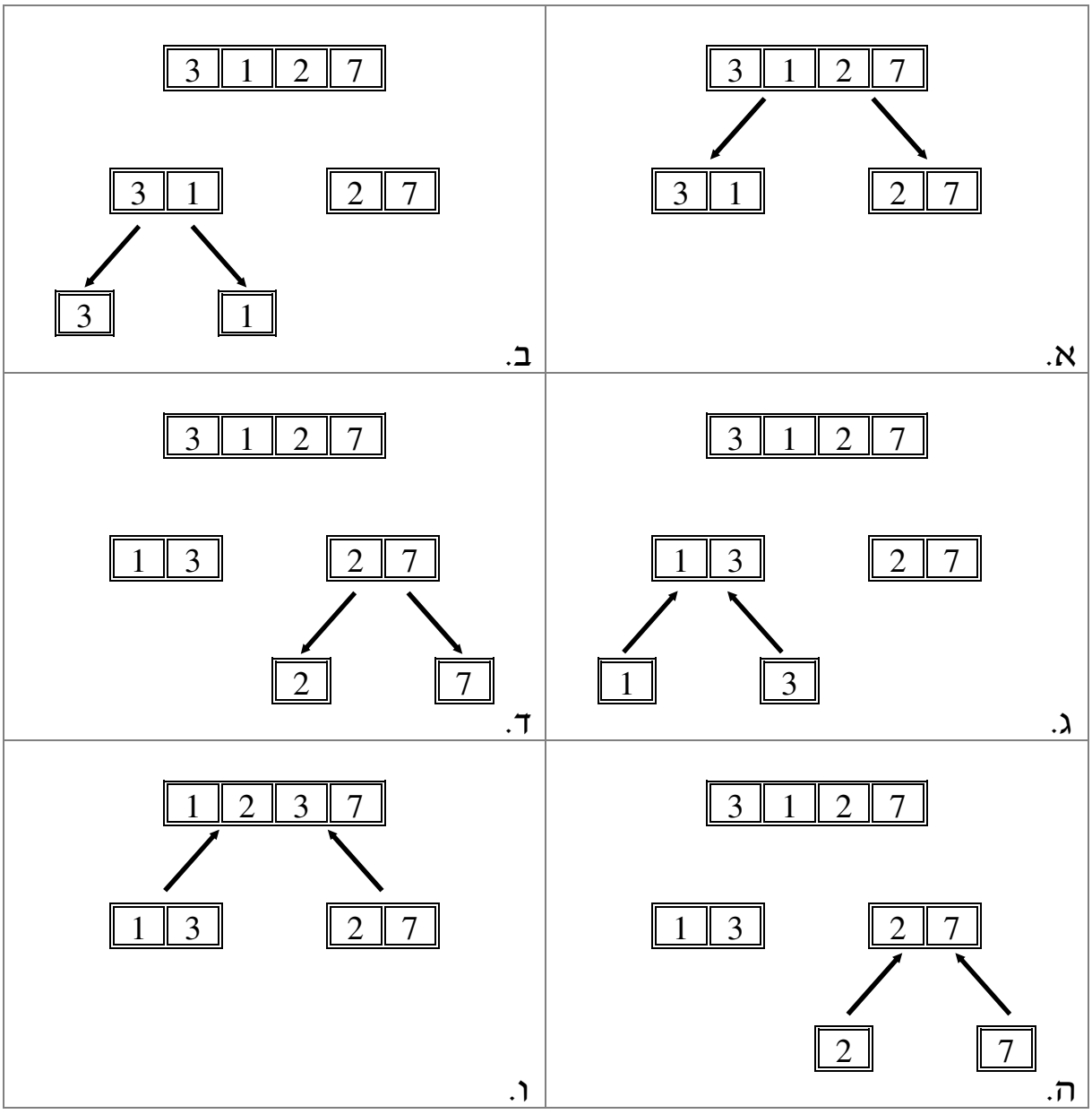
(1.2) ***מיון-מיזוג*** *$(arr1, \sim n/2)$.*

(1.3) ***מיון-מיזוג*** *$(arr2, \sim n/2)$.*

(1.4) *מזג את arr1 ואת arr2 לתוך A.*

אלגוריתם 6 - אלגוריתם רקורסיבי למיון-מיזוג

איור 7 מתאר את ריצת האלגוריתם על מערך שבו ארבעה איברים. בשלב א המערך הראשוני, שבו ארבעה איברים, מחולק ל-2 תת-מערכים בגודל 2 (שורה 1.1). בשלב ב קוראים ברקורסיה לאלגוריתם המיון על התת-מערך השמאלי (שורה 1.2). קריאה רקורסיבית זו יוצרת שני תת-מערכים בגודל 1, ובשלב זה נעצרת הרקורסיה. בשלב ג ממוזגים שני תת-מערכים אלו לתוך המערך המקורי בגודל 2, וכעת הוא ממוין. בשלבים ד ו-ה מבוצעת אותה סדרה של פעולות על התת-מערך הימני. סדרת פעולות זו מופעלת בשל הקריאה הרקורסיבית השנייה (שורה 1.3). לבסוף - בשלב ו - ממוזגים שני מערכים אלו למערך ממוין בגודל 4 (שורה 1.4).



איור 7 - מיון-מיזוג של מערך שבו ארבעה איברים

נבדוק מהו מספר הצעדים הבסיסיים שמבצע האלגוריתם מיון-מיזוג, הפועל על מערך שבו n איברים. נניח לשם פשטות ש-n הוא חזקה של 2. ברמת הרקורסיה הראשונה מחולק המערך לשני מערכים, שבכל אחד מהם n/2 איברים. ברמת הרקורסיה השנייה מחולקים מערכים אלו לארבעה מערכים, שבכל אחד מהם n/4 איברים וכך הלאה. הרקורסיה נעצרת כאשר המערך חולק ל-n מערכים, שבכל אחד מהם איבר בודד.

נבחן מהו מספר הצעדים הבסיסיים המבוצע בכל אחת מרמות הרקורסיה. ברמה הראשונה אורך המערך הבודד הוא n. שורה (1.4) ממזגת את שני התת-מערכים למערך אחד באורך n. כפי שראינו כאשר ניתחנו את יעילותו של אלגוריתם המיזוג, מספר הצעדים הבסיסיים שמבוצע במקרה כזה הוא n. מהו מספר הצעדים הדרוש לצורך ביצוע השורה (1.1)? ובכן, הדבר תלוי במימוש המדויק

של האלגוריתם, אך בכל מקרה מספר צעדים זה הוא *O(n)*. נוכל לסכם כי מספר הצעדים הבסיסיים שמבצע **מיון-מיזוג** ברמת הרקורסיה הראשונה הוא c·n, כאשר c הוא קבוע המתאר את מספר הצעדים הכולל הדרוש לצורך חלוקת המערך לשניים בשלב הראשון ומיזוגו בשלב השני.

רמת הרקורסיה השנייה מבוצעת על שני מערכים באורך n/2. על כל אחד ממערכים אלה מבוצעים c·n/2. בסך הכול מבוצעים ברמת הרקורסיה השנייה 2·(c·n/2) = c·n צעדים. באופן דומה ניתן לוודא כי בכל אחת מרמות הרקורסיה מבוצעים c·n צעדים.

איור 8 מתאר סכמתית את צעדי אלגוריתם המיון על מערך שבו 16 איברים. בשלב הראשון מבוצעים על מערך, אחד שבו 16 איברים, 16·c צעדים. בשלב השני יש שני מערכים שבכל אחד מהם 8 איברים, ועל כל אחד מהם מבוצעים 8·c צעדים - בסך הכול 16·c צעדים. גם בשלב השלישי והרביעי מבוצעים בסך הכול 16·c צעדים (שימו לב כי על מערכים באורך 1 לא מבוצע אף צעד).

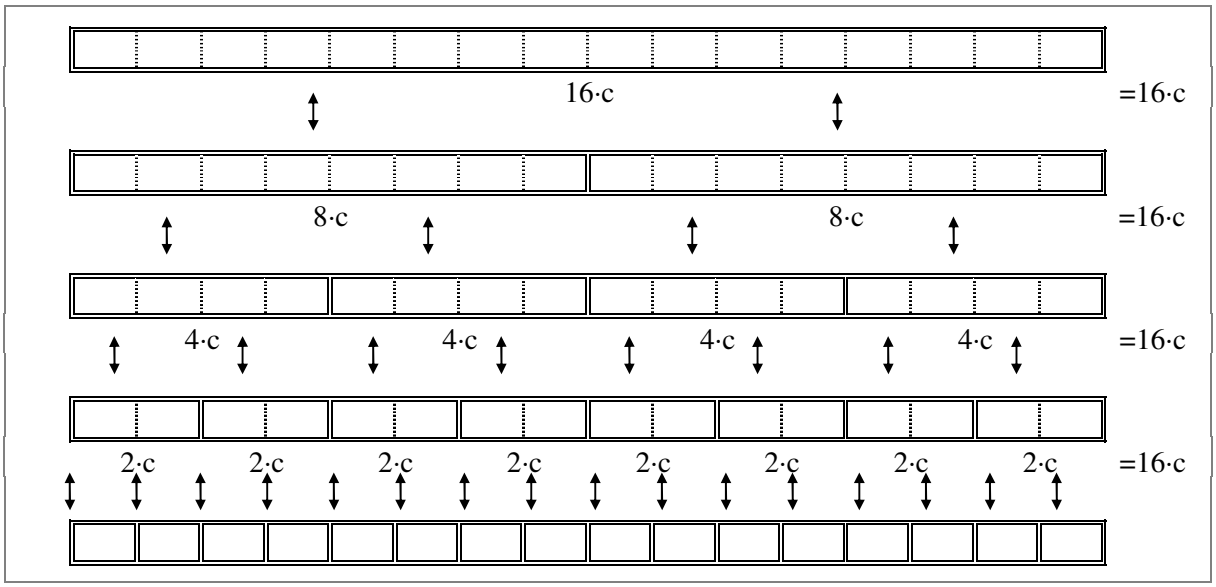
ומה באשר למספר הצעדים הכולל? ובכן, יש לחשב מהו מספר רמות הרקורסיה ולהכפילו ב-c·n. את מספר הרמות ניתן למצוא באותו אופן שבו מצאנו את מספר החלוקות באלגוריתם לחיפוש בינרי.

אנו מחפשים k, מספר שלבים, אשר יקיים: $= 1 /2 4 2 4 4 2 2 / 2 / . 4 . 4 3 / 2\ n$

k

כלומר: $2^k = n$

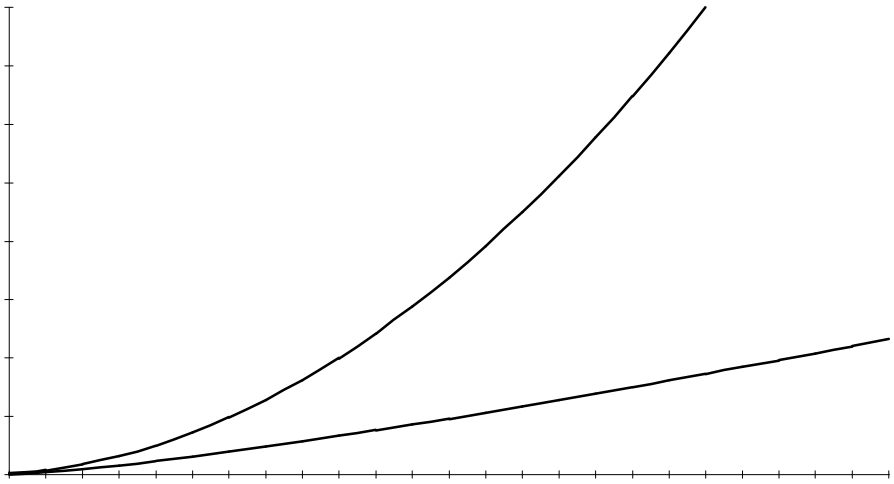
ומכאן: $k = \log_2 n$



איור 8 - מספר הצעדים הבסיסיים בעת מיונו של מערך שבו 16 איברים

מספר הצעדים הכולל שיבוצע על ידי האלגוריתם מיון-מיזוג על מערך באורך n הוא: $c \cdot n \cdot \log_2 n$, לכן סיבוכיות האלגוריתם היא *O(n·log n)*. זהו שיפור של ממש בהשוואה לאלגוריתם מיון-בועות,

שהוא בעל סיבוכיות ריצה ריבועית $O(n^2)$. איור 9 משווה בין שני סדרי הגודל ומראה כי קיים הבדל ניכר ביעילות האלגוריתמים כבר במערכים קצרים, הבדל שהולך וגדל ככל שגדל אורך המערך.



איור 9 - השוואה בין זמני הריצה של האלגוריתמים מיון-בועות ומיון-מיזוג

6. סיבוכיות זמן ריצה מעריכית

לסיום נחזור לבעיית הסוכן הנוסע ולדומותיה, הנחשבות בעיות בלתי סבירות. באלגוריתמים אלו פונקציות זמן הריצה גדלות בקצב מהיר כל כך, שאפילו בקלטים קצרים אי-אפשר להשתמש בהם לפתרון הבעיות.

אילו פונקציות נחשבות כמתארות זמן ריצה בלתי סביר? ובכן, משפחת הפונקציות החשובה ביותר המתארת זמני ריצה כאלו היא משפחת ה**פונקציות המעריכיות** (exponential functions). בפונקציות אלו מופיע אורך הקלט כמעריך של חזקה.

הפונקציה 2^n היא פונקציה מעריכית. אלגוריתם שסיבוכיות זמן ריצתו 2^n לא יהיה כדאי לממש על מחשב, אלא במקרים שבהם ברור שהקלטים שיוצגו קצרים במיוחד. כך, למשל, קלט שאורכו 10 ידרוש $2^{10} = 1,024$ צעדים בסיסיים, ולכן במקרה זה יהיה אפשר להיעזר במחשב. אולם קלט שאורכו 100 ידרוש $2^{100} = 1.26 \cdot 10^{30}$ צעדים בסיסיים, וביצוע האלגוריתם כבר לא יהיה אפשרי.

מובן כי קיימות פונקציות זמן ריצה שגדלות בקצב מהיר אף יותר מפונקציות מעריכיות פשוטות. הפונקציות $n \cdot 2^n$, 2^{n^2}, n^n ו-$n!$ כולן גדלות בקצב מהיר מפונקציות מעריכיות, וכולן נחשבות בלתי סבירות מבחינת זמן הריצה שלהן.

7. סיכום

- לבעיות נתונות קיימים פתרונות אלגוריתמיים שונים. יש לקבוע מדדים שיאפשרו להשוות בין הפתרונות השונים. יעילותו של פתרון אלגוריתמי נקבעת לפי זמן ביצועו ולפי מרחב הזיכרון שהוא צורך.

- צעד בסיסי הוא אוסף פעולות החוזר על עצמו בעת ביצוע אלגוריתם ומשך ביצועו אינו תלוי באורך הקלט.

- על מנת להגדיר מדד זמן שאינו תלוי בחמרה, נהוג להתייחס לתלות שבין מספר הפעמים שהאלגוריתם מבצע את הצעד הבסיסי לבין אורך הקלט.

- פונקציית זמן הריצה של אלגוריתם מתארת את מספר הצעדים הבסיסיים שמבצע האלגוריתם על קלט באורך n במקרה הגרוע ביותר.

- אפשר לשפר את יעילותו של אלגוריתם על ידי קיצור משך הזמן הדרוש לביצוע צעד בסיסי בודד. שיפור כזה מכונה **שיפור בקבוע**, והוא מאופיין ביחס קבוע בין זמני הריצה של האלגוריתם הראשוני והאלגוריתם המשופר עבור אורכי קלט גדולים מספיק.

- רעיון אלגוריתמי שונה לפתרון בעיה נתונה יכול להוביל ל**שיפור בסדר גודל**, המאופיין ביחס הולך וגדל בין זמני הריצה של האלגוריתם המקורי ושל האלגוריתם המשופר, ככל שגדל אורך הקלט.

- שיפור בסדר גודל הוא שיפור של ממש ביעילות אלגוריתם, ולכן לשם הנוחות נהוג להגדיר משפחות של סדרי גודל, המאגדות פונקציות זמני ריצה שהיחס ביניהן מתקרב לקבוע באורכי קלט גדולים מספיק.

- הכרנו שני אלגוריתמים לחיפוש במערך ממוין:
  - חיפוש סדרתי שסיבוכיות זמן הריצה שלו לינארית.
  - חיפוש בינרי שסיבוכיות זמן הריצה שלו לוגריתמית.

- הכרנו שני אלגוריתמים למיון מערך:
  - מיון-בועות שסיבוכיות זמן הריצה שלו ריבועית.
  - מיון-מיזוג שסיבוכיות זמן הריצה שלו $O(n \cdot \log n)$.

- קיימות בעיות שכל האלגוריתמים הידועים לפתרונן דורשים זמן בלתי סביר.

- אלגוריתמים שסיבוכיות זמן ריצתם מעריכית או גרועה ממעריכית אינם ישימים במחשב אלא לאורכי קלט קצרים במיוחד.

מושגים ומילות מפתח

input size	אורך קלט
binary search	חיפוש בינרי
linear search	חיפוש סדרתי
efficiency	יעילות
complexity measure	מדד סיבוכיות
merge sort	מיון-מיזוג
worst case	מקרה גרוע ביותר
best case	מקרה טוב ביותר
average case	מקרה ממוצע
order of magnitude	סדר גודל
run time complexity	סיבוכיות זמן ריצה
elementary step	צעד בסיסי
improvement by a factor	שיפור בקבוע

תרגילים