תרגול 1
ניתוח זמני ריצה

תאריך עדכון אחרון: 9 באפריל 2010[1]

מוטיבציה. נתחיל את דיוננו בדוגמא. נתאר שני אלגוריתמים לחישוב סידרת פיבונאצ'י. כלומר, האלגוריתם מקבל כקלט מספר n, וצריך להחזיר $fib(n)$ כאשר הפונקציה fib מוגדרת בצורה הבאה:

$$fib(n) = fib(n-1) + fib(n-2) \qquad \text{and} \qquad fib(1) = fib(2) = 1$$

נתבונן בשני האלגוריתמים הבאים לפתרון הבעיה:

```
int fib2(int n) {
    int preResult = 1, result = 1,
        temp = 0;
    for (int i=2; i<n; i++) {
        temp = result;
        result += preResult;
        preResult = temp;
    }
    return result;
}
```

```
int fib1(int n) {
     if (n == 1 || n == 2) {
         return 1;
     }
     return fib1(n-1) + fib1(n-2);
 }
```

איזה אלגוריתם הינו **יעיל** יותר? איך מודדים יעילות של אלגוריתם? על זאת ננסה לענות בשיעור זה.

במבני נתונים ובאלגוריתמיקה, אנו נדרשים להציג פתרונות לבעיות המוגדרות היטב. בכדי להעריך את טיב הפתרון, או בכדי להעריך עד כמה הפתרון אופטימלי, נרצה להעריך את מספר הפעולות שהמחשב מבצע בפתרון שהצענו. נרצה לבטא את מספר הפעולות שהמחשב נדרש לבצע כפונקציה של הקלט לבעיה.

בכדי להעריך זמן ריצה, אנו נדרשים למספר הנחות מקלות. ראשית, אנו מתבוננים במכונה שבה מעבד יחיד, והפעולות אותן אנו סופרים הן פעולות בסיסיות: השמה, פעולות אריתמתיות (+, -, *, /), פעולות בוליאניות, השוואה, גישה לתא במערך וכו'. אנו מניחים שזמן הביצוע של כל פעולה שכזו הוא קבוע.

כאמור, נדבר תמיד על מספר הפעולות כפונקציה של אורך הקלט לתוכנית. נעיר כי אלגוריתם כללי לפתרון בעיה צריך לעבוד לכל קלט, ולפיכך דרישה זו הגיונית. לצרוך העניין, ברור כי מיון 5 מספרים הינו קל ומהיר יותר ממיון 1000 מספרים; אם־כן, אנו מודדים את טיב האלגוריתם כמספר הפעולות אותו הוא מבצע כפונקציה של אורך הקלט שלו, ולא כפונקציה של מספר הפעולות שביצע בפועל בריצה ספציפית.

---

[1] נכתב ע"י גלעד אשרוב. הסיכום נכתב בעיקירו על בסיס תרגוליו של דודי בן חמו, 2005, והספר "מבוא לאלגוריתמים" של קורמן, לייזרסון, ריבסט, שטיין - ותורגם לעברית ע"י האוניברסיטה הפתוחה.

נעיר כי אלגוריתם יכול לעבוד זמן שונה עבור שני קלטים מאותו האורך. לדוגמא, אם נרצה למיין את הסדרה $1,2,3,5,4$, לעומת הסידרה: $5,2,3,1,4$, סביר (אך לא הכרחי) שאותו אלגוריתם מיון יעבוד "קשה יותר" (זמן רב יותר) עבור הסדרה השנייה ־ שכן הראשונה כמעט וממויינת. לפיכך, על פי רוב, אנו נחשב את מספר הפעולות של האלגוריתם על הקלט הגרוע ביותר שהאלגוריתם יכול לקבל ($worst\ case\ analysis$). נעיר כי לפעמים עורכים ניתוחים על קלט ממוצע לבעיה ($average\ case\ analysis$), וסיבוכיות אלגוריתם יכולה להיות שונה בשני המקרים. ברוב המקרים קשה יותר לנתח מהי הסיבוכיות עבור הקלט הממוצע מאשר סיבוכיות על המקרה הגרוע ביותר.

בכל בעיה שנדבר עליה, נציין במפורש מהו גודל הקלט; לדוגמא, במיון מספרים ־ נדבר על מספרים הקלטים, כלומר ־ n. לעומת זאת, אם נרצה לנתח אלגוריתם למכפלת שני מספרים, נדבר על מספר הסיביות הנצרכים לייצוג כל אחד מן המספרים. אם הקלט הוא גרף, נתאר את מספר הפעולות כפונקציה של מספר הקשתות בגרף, או מספר הקודקודים[2].

זמן ריצה. זמן ריצה של אלגוריתם על קלט מסויים הוא מספר פעולות היסוד המבוצעות. נרצה לחשב את סיבוכיות הזמן של אלגוריתם באופן מתמטי כך שנתעלם מאספקטים "טכנולוגיים" כגון מהירות המחשב שעליו מריצים את האלגוריתם (שכן, ברור שכאשר יוצא מחשב חדש מהיר המהיר פי שניים, האלגוריתם שלנו ירוץ מהר פי שניים). "סיבוכיות הזמן" תתאר את סדר הגודל של הפעולות הנדרשות, ותתעלם מקבועים. לצורך פשטות החישוב, נתאר את מושג ה"אסימפטוטיקה", העוזר לנו ל"הפטר" מכל הקבועים.

סיבוכיות זיכרון. לעיתים, נרצה למדוד את כמות הזיכרון שבו האלגוריתם משתמש. שוב, נרצה להתעלם מאספקטים "טכנולוגיים", ולכן גם פה נשתמש באסימפטוטיקה.

1.1 אסימפטוטיקה

אסימפטוטיקה הינה הערכה של קצב גידול של פונקציה. מה שנותר בחישוב הוא רק האיבר המשמעותי ביותר. למעשה, הדבר שקול לשאלה ־ למה שואף זמן הריצה כשגודל הקלט שואף לאינסוף.

1.1.1 חסם אסימפטוטי עליון ־ O

בד"כ, נרצה לחסום את זמן הריצה "מלמעלה". נניח תוכנית A רצה בזמן $f(n)$ עבור קלט מאורך n. אם $f(n)=O(g(n))$, אזי מספר הפעולות שהתוכנית עושה הוא סדר גודל של $g(n)$ פעולות לכל היותר. באופן מפורש יותר, שתי ההגדרות הבאות שקולות:

הגדרה 1.1 נאמר ש ־ $f(x)=O(g(x))$ אם ורק אם קיימים שני קבועים, $c>0$, $x_0\geq 0$ כך שלכל $x\geq x_0$ מתקיים:

$$|f(x)|\leq c\cdot|g(x)|$$

הגדרה 1.2 נאמר ש ־ $f(x)=O(g(x))$ אם ורק אם קיים קבוע $c>0$ כך ש:

$$\lim_{x\to\infty}\left|\frac{f(x)}{g(x)}\right|\leq c$$

[2] בקורסים מתקדמים בתיאוריה של מדעי המחשב, כגון חישוביות וסיבוכיות, אנחנו מחשבים את סיבוכיות האלגוריתם כפונקציה של מספר הביטים הנדרשים לייצוג כל הקלט. על כל פנים, בקורס זה אנו "מרמים" קצת, ומגדירים בכל פעם מהו הפרמטר בקלט שאליו אנחנו מייחסים את הסיבוכיות.

כאשר אנחנו אומרים "זמן הריצה הוא $O(n^2)$" הכוונה היא שזמן הריצה במקרה הגרוע ביותר, לקלט הגרוע ביותר, חסום ע"י $n^2 \cdot c$ כאשר c הוא איזשהו קבוע. אנחנו בעצם מסתכלים על המקרה הגרוע ביותר.

בנוסף, לעיתים רושמים ביטויים כגון: $n^2 + 5n + 2 = n^2 + O(n)$, כלומר, הופכים את $5n + 2$ ל - $O(n)$. בד"כ כשרושמים ביטויים מסוג זה, מתכוונים שישנו איזשהו $O(n)$ בביטוי שלא ממש מעניין; הגודל המעניין הוא n^2.

יש כאלו שמגדירים את $O(g(x))$ כמשפחה של פונקציות המקיימות $f(x) = O(g(x))$. באופן מפורש יותר:

$$O(g(n)) = \{f(n) \mid \exists n_0, c \geq 0 \text{ such that } \forall n \geq n_0 : \ 0 \leq f(n) \leq c \cdot g(n)\}$$

במקרה כזה, נאמר כי $f(n) \in O(g(n))$ ולא כמו בהגדרה הקודמת $f(n) = O(g(n))$. אנו נעבוד לפי שתי ההגדרות הראשונות בלבד.

דוגמאות:

1. נניח שזמן ריצה של אלגוריתם A הוא $f(n) = 10n^2 + 5n$. אזי, ברור כי $f(n) = O(n^2)$. זאת מכיוון שקיים קבוע $c = 15$ כך שלכל n (קיים $x_0 = 0$, כך שלכל $n > x_0$) מתקיים:

$$10n^2 + 5n < 10n^2 + 5n^2 = 15n^2.$$

ולכן, לפי הגדרה 1.1 נקבל כי $f(n) = O(n^2)$. בעזרת הגדרה 1.2 נקבל את אותה התוצאה; נחשב:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{10n^2 + 5n}{n^2} = 10$$

ברור כי 10 הינו קבוע.

נציין כי אותה הפונקציה, $f(n)$ מקיימת: $f(n) = O(n^2 \log n),\ f(n) = O(n^3),\ f(n) = O(n!)$ וניתן לחשוב על עוד דוגמאות נוספות (למעשה, כל פונקציה שגדולה מ - n^2).

2. נתבונן בפונקציה $f(n) = n \log n^5 + 6n$. נראה כי $f(n) = O(n \log n)$:

$$\lim_{n\to\infty} \frac{f(n)}{n\log n} \ = \ \lim_{n\to\infty} \frac{n\log n^5 + 6n}{n \log n} = \lim_{n\to\infty} \frac{5n\log n}{n\log n} + \lim_{n\to\infty} \frac{6n}{n\log n} = 5 + \lim_{n\to\infty} \frac{6}{\log n} = 5$$

ולכן, לפי הגדרה 1.2 אנו מקבלים כי $f(n) = O(n \log n)$.

3. **טענה:** $n^2 \neq O(n)$

הוכחה: נוכיח בעזרת כל אחת מההגדרות. נתחיל עם הגדרה 1.1: נניח בשלילה שקיים c וקיים n_0 כך שלכל $n > n_0$ מתקיים:

$$n^2 \leq c \cdot n \Rightarrow n \leq c$$

בסתירה לכך שהנוסחא מתקיימת לכל $n > n_0$.

כעת, נראה הוכחה נוספת על סמך הגדרה 1.2:

$$\lim_{n\to\infty} \left|\frac{n^2}{n}\right| = \lim_{n\to\infty} |n| \longrightarrow \infty \geq c$$

לכל קבוע $c > 0$. ■

1.1.2 חסם אסימפטוטי תחתון - Ω

לעיתים נרצה לדבר על חסם תחתון לבעיה מסויימת. לדוגמא, ברור שמיון של n מספרים דורש לפחות n פעולה (סתם לבדוק אם המספרים ממויינים עולה סדר גודל של n פעולות). נרצה לחסום את זמן הריצה מלמטה. באופן מפורש יותר, שתי ההגדרות הבאות שקולות:

הגדרה 1.3 נאמר ש - $f(x) = \Omega(g(x))$ אם קיים קבוע $c > 0$, וקיים x_0 כך שלכל $x \geq x_0$ מתקיים:

$$0 \leq c \cdot g(x) \leq f(x)$$

הגדרה 1.4 נאמר ש - $f(x) = \Omega(g(x))$ אם קיים קבוע $c > 0$ כך ש:

$$\lim_{x\to\infty} \left|\frac{g(x)}{f(x)}\right| \leq c$$

דוגמאות:

1. **טענה:** $3n^2 + 5 = \Omega(n)$.

   הוכחה: נקח $c = 1$. צ.ל n_0 כך שלכל $n \geq n_0$ מתקיים:

   $$3n^2 + 5 > 1 \cdot n \Rightarrow 3n^2 - n + 5 > 0$$

   הנ"ל פרבולה "מרחפת" ולכן מספיק לקחת $n_0 = 1$. ■

2. **טענה:** $3n^2 + 5 = \Omega(n^2)$.

   הוכחה:

   $$\lim_{n\to\infty} \frac{n^2}{3n^2 + 5} = \frac{1}{3}$$

   ■

3. **טענה:** $3n^2 + 5n \neq \Omega(n^3)$.

   הוכחה:

   $$\lim_{n\to\infty} \left|\frac{n^3}{3n^2 + 5}\right| = \infty > c$$

   לכל קבוע $c > 0$. ■

1.1.3 חסם הדוק אסימפטוטית - Θ

באופן אינטואיטיבי, נאמר ש - $f(n) = \Theta(g(n))$ אם מתקיים: $f(n) = O(g(n))$ וגם $f(n) = \Omega(g(n))$. באופן פורמלי:

הגדרה 1.5 נאמר ש - $f(n) = \Theta(g(n))$ אם קיימים קבועים $c_1, c_2 > 0$, וקיים קבוע n_0 כך שלכל $n \geq n_0$ מתקיים:

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

הגדרה שקולה:

הגדרה 1.6 נאמר ש ־ $f(n) = \Theta(g(n))$ אם קיים קבוע $c > 0$ כך ש:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

דוגמא. נראה ש ־ $3n^2 + 5 = \Omega(n^2)$ לפי שתי ההגדרות.
לפי הגדרה 1.5, נקח $c_1 = 1$, $c_2 = 8$, ו ־ $n_0 = 0$. לכל $n > n_0$ מתקיים:

$$c_1 \cdot n^2 = n^2 \leq 3n^2 + 5 \leq 8n^2 = c_2 n^2.$$

בכדי להראות זאת לפי הגדרה 1.6, נקבל:

$$\lim_{n\to\infty} \frac{3n^2+5}{n^2} = 3.$$

טענה 1.7 לכל שתי פונקציות $f(n)$, $g(n)$: $f(n) = \Omega(g(n))$ וגם $f(n) = O(g(n))$ אם ורק אם: $f(n) = \Theta(g(n))$.

ההוכחה נשארת לקורא כתרגיל (לא שיש יותר מדי מה להוכיח פה..).

1.1.4 הסימון ־ o

כפי שציינו, הסימון O מציין חסם עליון לפונקציה ־ לאו דווקא הדוק. כאשר אנו יודעים בוודאות שהחסם אינו הדוק, ניתן להשתמש ב ־ o (o "קטן"). לדוגמא, נתבונן בפונקציה $5n$. נקבל כי $5n = O(n)$ וגם $5n = O(n^2)$, אך החסם האחרון אינו הדוק אסימפטוטית. במקרה זה נרשום: $n = o(n^2)$. נדגיש כי $5n \neq o(n)$. באופן פורמלי, נקבל:

הגדרה 1.8 נאמר ש ־ $f(n) = o(g(n))$ אם לכל קבוע חיובי $c > 0$ קיים קבוע $n_0 > 0$ כך שלכל $n \geq n_0$ מתקיים:

$$0 \leq f(n) < cg(n)$$

הגדרה שקולה:

הגדרה 1.9 נאמר ש$f(n) = o(g(n))$ אם מתקיים:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

1.1.5 הסימון - ω

כפי שהגדרנו יחס שבין o קטן לבין O גדול, מציג עכשיו את היחס שבין Ω לבין ω. אנו משתמשים ב - ω בכדי לציין חסם תחתון שאינו הדוק אסימפטוטית. לדוגמא, אם הפונקציה היא n^2 אז היא שייכת גם ל - $\Omega(n^2)$ וגם ל - $\Omega(n)$. החסם $\Omega(n)$ אינו הדוק, ולכן ניתן לרשום $n^2 = \omega(n)$, אך: $n \neq \omega(n)$.

הגדרה 1.10 נאמר כי $f(n) = \omega(g(n))$ אם לכל קבוע חיובי $c > 0$ קיים קבוע $n_0 > 0$ כך שלכל $n \geq n_0$ מתקיים:

$$0 \leq cg(n) < f(n)$$

הגדרה 1.11 נאמר ש$f(n) = \omega(g(n))$ אם מתקיים:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$

1.1.6 השוואת פונקציות

למעשה, פונקציות $O, \Omega, \Theta, o, \omega$ הן מעין יחס גדול / קטן:

$$\begin{aligned}
f(n) = O(g(n)) &\quad \approx \quad f(n) \leq g(n) \\
f(n) = \Omega(g(n)) &\quad \approx \quad f(n) \geq g(n) \\
f(n) = \Theta(g(n)) &\quad \approx \quad f(n) = g(n) \\
f(n) = o(g(n)) &\quad \approx \quad f(n) < g(n) \\
f(n) = \omega(g(n)) &\quad \approx \quad f(n) > g(n)
\end{aligned}$$

כמובן שכל היחסים - $=, \leq, \geq, <, >$ הם אסימפטוטיים, ולא שיוויונים אמיתיים.
רבים מן היחסים המתקיימים בין מספרים ממשיים מתקימים גם בהשוואות אסימפטוטיות.

טרנזיטיביות.

- אם $f(n) = \Theta(g(n))$ וגם $g(n) = \Theta(h(n))$ אזי $f(n) = \Theta(h(n))$.
- אם $f(n) = O(g(n))$ וגם $g(n) = O(h(n))$ אזי $f(n) = O(h(n))$.
- אם $f(n) = \Omega(g(n))$ וגם $g(n) = \Omega(h(n))$ אזי $f(n) = \Omega(h(n))$.
- אם $f(n) = o(g(n))$ וגם $g(n) = o(h(n))$ אזי $f(n) = o(h(n))$.
- אם $f(n) = \omega(g(n))$ וגם $g(n) = \omega(h(n))$ אזי $f(n) = \omega(h(n))$.

רפלקסיביות.

- $f(n) = \Theta(f(n))$
- $f(n) = O(f(n))$
- $f(n) = \Omega(f(n))$

סימטריה. $f(n)=\Theta(g(n))$ אם ורק אם $g(n)=\Theta(f(n))$.

בנוסף, נשים לב:

$$\begin{aligned}
\text{constants}: &\quad n^{\frac{1}{c+1}} < \frac{1}{n^c} < \frac{1}{\log n} < 1 < \ldots \\
\text{logarithms}: &\quad < \log n < (\log n)^k < \ldots \\
\text{``polynomials''}: &\quad < \sqrt{n} < n < n\log n < n^k < n^{k+1} < \ldots \\
\text{exponentials}: &\quad < 2^n < n! < n^n
\end{aligned}$$

שאלה: בהינתן $f(x)$, $g(x)$, האם תמיד מתקיים־ $g(x)=O(f(x))$ או $f(x)=O(g(x))$?

תשובה: לא. נתבונן ב - $f(n)=n^{1+\sin n}$, וב - $g(n)=n$. לא ניתן להשוות בינהם אסימפטוטית.

שאלה: האם לכל פונקציה $f(n)$ קיימת פונקציה "סטנדרטית" ($n, n\log n, n^2, \ldots$) $g(n)$ כך ש - $f(n)=\Theta(g(n))$?

תשובה: לא. נתבונן ב - $f(n)$ המוגדרת באופן הבא:

$$f(n)=\begin{cases} n & n \text{ odd} \\ n^3 & \text{o.w} \end{cases}$$

תרגיל 1.12 הוכח: $\log(n!)=\Theta(n\log n)$

הוכחה:

מתקיים:

$$\log(n!) = \log(1\cdot 2\cdot\ldots\cdot n) = \log\left(\prod_{i=1}^{n} i\right) < \log\left(\prod_{i=1}^{n} n\right) = \log n^n = n\log n$$

ולכן $\log(n!)=O(n\log n)$.

נותר להראות כי קיים c עבורו: $\log(n!) > cn\log n$. נקבל:

$$\begin{aligned}
\log(n!) &= \log(1\cdot 2\cdot\ldots\cdot n) = \sum_{i=1}^{n}\log i = \sum_{i=1}^{\frac{n}{2}-1}\log i + \sum_{i=\frac{n}{2}}^{n}\log i \\
&> \sum_{i=\frac{n}{2}}^{n}\log i > \sum_{i=\frac{n}{2}}^{n}\log\frac{n}{2} = \frac{n}{2}\log\frac{n}{2} = \frac{n}{2}(\log n - \log 2) \\
&= \frac{n}{2}(\log n - 1) = \frac{n}{2}\log n - \frac{n}{2}
\end{aligned}$$

כאשר $n>4$ נקבל כי: $\log n > 2$, ולכן $\frac{n}{4}\log n > \frac{n}{2}$. נקבל כי לכל ה־$n$ים המספיק גדולים:

$$\log(n!) > \frac{n}{2}\log n - \frac{n}{2} > \frac{n}{2}\log n - \frac{n}{4}\log n = \frac{n}{4}\log n$$

ולכן, כאשר $c=\frac{1}{4}$, ו - $n_0=4$, נקבל כי לכל $n>n_0$ מתקיים: $\log(n!) > c\cdot n\log n$, כלומר - $\log(n!)=\Omega(n\log n)$. נסיק אם כן כי $\log(n!)=\Theta(n\log n)$ ∎

1.2 סיבוכיות קוד

ניתן ללמוד את סיבוכיות האלגוריתם מתוך מבט כללי על מבנה הקוד. לפי מספר הלולאות שהקוד מבצע, לפי קריאות רקורסיביות, וכו'.

דוגמא: נתבונן בקוד הבא:

```
for (unsigned u=0; u<n; ++u) {
    basic_step1;
    basic_step2;
}
```

ישנה לולאה שרצים עליה n פעמים, בכל פעם מבצעים 2 פעולות, ולכן סיבוכיות הקוד הינה - $2n$, כלומר - $O(n)$ (למעשה $\Theta(n)$).

דוגמא נוספת: נתבונן בקוד הבא:

```
for (unsigned u=0; u<10; ++u) {
    basic_step1;
    basic_step2;
}
```

מספר הפעולות שהאלגוריתם מבצע הוא:

$$\sum_{u=0}^{9} 2 = 20$$

ולכן, נקבל $O(1)$.

עוד דוגמא: נתבונן בקוד הבא:

```
for (int i = n; i > 0; --i) {
    for (unsigned j=0; j<n; ++j) {
        basic_step;
    }
}
```

$$\sum_{i=1}^{n} \sum_{j=0}^{n-1} 1 = \sum_{i=1}^{n} n = n^2$$

כלומר, $O(n^2)$.

ועוד אחת: נתבונן בקוד הבא:

```
for (int i = 1; i <= n; i*=2) {
    basic_step;
}
```

בכל איטרציה מבצעים בדיוק פעולה אחת. כמה איטרציות יש לנו? נעקוב אחרי i: הערכים אותם הוא מקבל: $1, 2, 4, 8, \ldots, n$. יש לנו למעשה סידרה הנדסית עם הפרמטרים: $a_1 = 1$, $a_k = n$, $q = 2$. לפי נוסחא לסידרה הנדסית: $a_k = a_1 \cdot q^{k-1}$, כלומר: $n = 1 \cdot 2^k$, ולכן $k = \log n$. כלומר, מספר האיטרציות הוא $\log n$. מכיוון שבכל איטרציה מבצעים פעולה אחת, נקבל כי הסיבוכיות היא: $O(\log n)$.

מה קורה כאשר הלולאות תלויות אחת בשניה? לדוגמא, נתבונן בקוד הבא:

```
for (int i = 1; i <= n; i *= 2) {
    for (j = 1; j <= i; j++) {
        basic_step;
    }
}
```

במקרה זה לא נוכל סתם לכפול את הלולאה החיצונית בלולאה הפנימית; נתבונן קודם בניתוח **לא מדויק**: הלולאה החיצונית מתבצעת $\log n$ פעמים, הלולאה הפנימית מתבצעת במקרה הגרוע ביותר n פעמים, ולכן נקבל בסה"כ ־ $O(n \log n)$. **הנ"ל אינו חסם הדוק!**

בכדי לקבל חסם הדוק, נחשב לפי סיגמאות. נשים לב כי i מקבל ערכים $1, 2, 4, 8, \ldots, n$, לפיכך, נגדיר משתנה k שירוץ מ ־ 0 ועד ל ־ $\log n$, ונגדיר את i להיות 2^k. נקבל:

$$\sum_{k=0}^{\log n} \sum_{j=1}^{i} 1 = \sum_{k=0}^{\log n} \sum_{j=1}^{2^k} 1 = \sum_{k=0}^{\log n} 2^k = 2^{\log n+1} - 1 = 2 \cdot 2^{\log n} - 1 = 2n - 1 = \Theta(n)$$

כלומר, הסיבוכיות היא לינארית.