מבני נתונים - תרגול 1
סדר גודל - אסימפטוטיקה*

גלעד אשרוב

27 בפברואר 2013

תקציר

בשיעור זה נלמד על קצב גידול של פונקציות, ועל סימוני $O, \Omega, \Theta, o, \omega$. נעמוד על הקשרים בינהם, וכיצד לסווג פונקציות למחלקות שקילות אסיפמטוטית. לשם מוטיבציה, נלמד כיצד לנתח זמני ריצה של קוד.

1 מוטיבציה[1]

נתחיל עם הקדמה קצרה, שהיא תהווה מוטיבציה לשיעור כולו (ואולי גם לקורס כולו).

נניח ישנה רשימה של שמות, ואנחנו מחפשים שם כלשהו בתוך הרשימה. כלומר - הקלט ל"אלגוריתם" שלנו הוא רשימה של שמות, ושם אותו אנחנו מחפשים בתוך הרשימה. הפלט שאנחנו מבקשים להחזיר הוא האם השם נמצא ברשימה, או המקום ברשימה בה הוא נמצא.

שיטה פשוטה לחיפוש השם ברשימה היא לעבור על השמות ברשימה, אחד אחרי השני, כאשר בכל פעם נבדוק האם השם הנוכחי הוא השם המבוקש, ואם לא - לפני שנמשיך לשם הבא ברשימה - נבדוק קודם האם הגענו לסוף הרשימה. נשים לב שבכל פעם אנחנו מבצעים שתי פעולות פשוטות (פעולות יסוד): השוואת האיבר לשם שאותו אנו מחפשים, ובמקרה הצורך - השוואת האיבר לסוף הרשימה. סה"כ, אם ישנם N איברים ברשימה, נשלם במקרה הכי גרוע (למשל, כשהשם המבוקש אינו ברשימה) - $2N$.

בעזרת טריקים פשוטים, ניתן לשפר את זמן הריצה בצורה שנראית לכאורה - דרסטית. אנו יכולים להוסיף את השם המבוקש לסוף הרשימה לפני תחילת החיפוש. כעת, כאשר נעבור על הרשימה - אין צורך לשאול יותר "האם הגענו לסוף הרשימה", שכן - השם המבוקש תמיד נמצא בתוך הרשימה. כאשר נמצא את האיבר המבוקש, נוכל לדעת האם הוא בעצם האיבר האחרון ברשימה (ובמקרה זה למעשה - השם לא נמצא ברשימה), או שהשם נמצא בתוך הרשימה. לאחר סיום החיפוש, נוכל להחזיר את הרשימה למצבה המקורי.

כמה זה משפר? ובכן, במקרה הכי גרוע שבו השם לא נמצא, אנו עוברים על כל הרשימה כמו מקודם. ההבדל הוא שכעת בכל פעם אנחנו מבצעים בדיקה אחת במעבר, במקום שתיים. בנוסף, נוספה לנו הכנסת השם המבוקש לסוף הרשימה והוצאתו בסיום החיפוש. לכן, העלות במקרה גרוע היא בערך $N + 2$. שימו לב שיש פה שיפור של בערך 50%, וכל זה בעזרת "תכסיס" פשוט מאוד.

מרשים ככל שיישמע, שיפור זה בזמן הריצה לא באמת דרסטי. השיפור שביצענו כאן הוא שיפור ב**קבוע** ולא ב**סדר גודל**. אצלנו, ככל שN גדל, היחס בין שני האלגוריתמים נשאר קבוע: תמיד האלגוריתם השני ירוץ בערך פי שניים יותר מהר מהאלגוריתם הראשון, וזה נכון לכל N (לפחות, לכל הNים ה"מספיק גדולים"). לעומת זאת, שיפור ב**סדר גודל** משנה לחלוטין את המאזן: אם אלגוריתם א' משפר את אלגוריתם ב' ב**סדר גודל**, אזי ככל שהקלט N יהיה גדול יותר, יחס השיפור יהיה גדול יותר. היחס בין זמני הריצה גדל ככל שהקלט גדל.

*השיעור מבוסס ברובו על הפרק המתאים בספר - "מבוא לאלגוריתמים", הוצאת האוניברסיטה הפתוחה.

[1]מבוססת על הספר - "המחשב אינו כל-יכול" של פרופ' דוד הראל ממכון וייצמן. הספר הוא מעין מבוא לתיאוריה של מדעי המחשב ונועד לקהל הרחב (ספר "מדע פופולארי"). מומלץ למי שרוצה להסביר לחברים מה הוא עושה בתואר :)

לדוגמא, נראה את השיפור הבא לאותה הבעייה שצוינה לעיל. נניח הפעם שהקלט שלנו הוא מערך (כלומר, ישנה גישה מהירה לכל תא), ושישנו יחס סדר בין האיברים (לדוגמא, במקרה שאכן הקלט הוא שמות, יחס סדר כזה יכול להיות סדר לקסיקוגרפי), ונניח את האלגוריתם הבא:

```
binarySearch(Array<Element> A, Element s, int startIdx, int endIdx) {
    if (endIdx < startIdx)
        return "not found"
    int med = (endIdx-startIdx)/2
    if (A[med] > s)
        return binarySearch(A, s, startIdx, med-1)
    else
        if (A[med] < s)
            return binarySearch(A, s, med+1, endIdx)
        else
            return med
}
```

האלגוריתם שבנינו מחפש את האיבר באמצע המערך. אם הוא נמצא שם - סיימנו. אם לא - הוא בודק האם הוא קטן מהאיבר שנמצא באמצע המערך או לא. אם הוא קטן - הוא מחפש את האיבר בחצי המערך ה"תחתון" (היכן שהאיברים ה"קטנים" נמצאים). אם לא - הוא מחפש אותו בחצי המערך ה"עליון" (היכן שהאיברים ה"גדולים" מופיעים). בצורה זו, עם כל השוואה יחידה - אנחנו "מוציאים מהמשחק" **חצי** מהאיברים!

מה השיפור שקיבלנו? ובכן, נשים לב שזמן הריצה של האלגוריתם הנ"ל על קלט באורך N הוא - $\log_2 N$. זהו שיפור **עצום**. בניגוד לשיפור הקודם שהיה שיפור בקבוע, פה ככל שהקלט גדל - האלגוריתם משפר הרבה יותר. לצורך ההבנה, נתבונן פשוט בטבלה הבאה, המשווה בין שני האלגוריתמים כפונקציה של אורך הרשימה, מספר ההשוואות, והיחס שבין זמני הריצה של שני האלגוריתמים:

אורך הרשימה N	מספר ההשוואות חיפוש סידרתי N	מספר ההשוואות חיפוש בינארי $\log_2 N$	יחס השיפור $N/\log_2 N$
10	10	4	$*2.5$
1000	1000	10	$*100$
1,000,000	1,000,000	20	$*50{,}000$
1,000,000,000	1,000,000,000	30	$*33{,}333{,}333$

שימו לב כמה יחס השיפור הוא **עצום**. יותר מזה - שימו לב כמה שהאלגוריתם שקיבלנו הוא מהיר. נניח שיש לנו ספר טלפונים שמכיל את כל הטלפונים בעולם (משהו כמו - 10 מליארד טלפונים (?)) - עבור קלט כזה, בכדי למצוא מספר טלפון של מישהו לפי השם שלו - אנחנו נדרשים ל34 השוואות בלבד!

ניתוח זמן ריצה. 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.

נעיר כי אלגוריתם יכול לעבוד זמן שונה עבור שני קלטים מאותו האורך. לדוגמא, אם נבצע חיפוש סידרתי שבו האיבר מופיע ברשימה (ואולי בתחילתה) לעומת המקרה שבוא האיבר אינו מופיע בה כלל - האלגוריתם יעבוד "קשה יותר" (זמן רב יותר) עבור הקלט האחרון.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.

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.

2 אסימפטוטיקה

אסימפטוטיקה הינה הערכה של קצב גידול של פונקציה. מה שנותר בחישוב הוא רק האיבר המשמעותי ביותר. עבור שתי פונקציות נתונות, $f(n), g(n)$, נרצה לומר מה היחס בינהן - האם $f(n)$ "גדולה" מ - $g(n)$, שקולה לה, או קטנה ממנה. בהינתן יחס סדר שכזה, נוכל להגדיר משפחות סטנדרטיות של פונקציות, ולמעשה לסווג פונקציות למשפחות לפי "גודלן". לאורך כל הדיון, נתבונן בפונקציות $f(n), g(n)$ חיוביות.

2.1 חסם אסימפטוטי עליון - O

בד"כ, נרצה לחסום את זמן הריצה "מלמעלה". נניח תוכנית A רצה בזמן $f(n)$ עבור קלט מאורך n. אם $f(n) = O(g(n))$, אנו בעצם אומרים שהפונקציה $f(n)$ היא "קטנה שווה" לפונקציה $g(n)$. נתן כעת שתי הגדרות שקולות:

הגדרה 1. נאמר ש - $f(n) \in O(g(n))$ אם ורק אם קיימים שני קבועים, $c > 0$, $n_0 \geq 0$ כך שלכל $n \geq n_0$ מתקיים:

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

הגדרה 2. נאמר ש - $f(x) \in O(g(x))$ אם ורק אם קיים קבוע $c > 0$ כך ש:

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

נשים לב שכאשר אנחנו אומרים כי $f(n) \in O(g(n))$, אין זו אומר שתמיד $f(n) \leq g(n)$; בהחלט ייתכן כי עבור nים מסויימים - $f(n) > g(n)$. מה שההגדרה כן אומרת היא כי החל מאיזשהו n מסויים, תמיד $f(n) \leq g(n)$ (או $f(n) \leq c \cdot g(n)$ עבור איזשהו קבוע c).

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.

דוגמאות:

1. נניח שזמן ריצה של אלגוריתם A הוא $f(n) = 10n^2 + 5n$. אזי, ברור כי $f(n) \in O(n^2)$. זאת מכיוון שקיים קבוע $c = 15$ כך שלכל n (קיים $n_0 = 0$, כך שלכל $n > n_0$) מתקיים:

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

ולכן, לפי הגדרה 1. נקבל כי $f(n) = O(n^2)$. בעזרת הגדרה 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: $f(n) \in O(n^2 \log n),\ f(n) \in O(n^3),\ f(n) \in O(n!)$ 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 $f(n) \in O(n \log n)$:

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

ולכן, לפי הגדרה 2. אנו מקבלים כי $f(n) \in O(n \log n)$.

3. **טענה:** $n^2 \neq O(n)$

הוכחה: נוכיח בעזרת כל אחת מההגדרות. נתחיל עם הגדרה 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 2.:

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

לכל קבוע $c > 0$. ■

2.2 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 3. נאמר ש - $f(n) \in \Omega(g(n))$ אם קיים קבוע $c > 0$, וקיים n_0 כך שלכל $n \geq n_0$ מתקיים:

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

הגדרה 4. נאמר ש - $f(n) \in \Omega(g(n))$ אם קיים קבוע $c > 0$ כך ש:

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

דוגמאות:

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

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

הוכחה:

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

■

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

הוכחה:

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

לכל קבוע $c > 0$. ■

2.3 חסם הדוק אסימפטוטית - Θ

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

הגדרה 5. נאמר ש- $f(n) \in \Theta(g(n))$ 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 6. נאמר ש- $f(n) \in \Theta(g(n))$ 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 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 6., נקבל:

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

טענה 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..).

2.4 הסימון - o

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

הגדרה 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:

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

כלומר, אנחנו לוקחים את הפונקציה $g(n)$ ומנסים להקטין אותה בעזרת איזשהו קבוע - c (כאן, c קבוע קטן ממש, נניח $c = 0.000001$). לכל קבוע קטן כזה - נוכל למצוא n_0 שהחל ממנו $f(n) < cg(n)$. הגדרה שקולה:

הגדרה 9. נאמר ש$f(n) = o(g(n))$ אם מתקיים:

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

2.5 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 10. נאמר כי $f(n) \in \omega(g(n))$ 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 11. נאמר ש$f(n) \in \omega(g(n))$ אם מתקיים:

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

2.6 השוואת פונקציות

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

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

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 $g(n) = \Theta(f(n))$.

מחלקות של פונקציות. בהינתן פונקציה $f(n)$, ננסה לשייך אותה למחלקה $g(n)$ "סטנדרטית". הפונקציות $g(n)$ שעליהן בד"כ מסתכלים, והיחס בינהם, הם כדלהלן:

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

3 תרגילים

שאלה: בהינתן $f(n)$, $g(n)$, האם תמיד מתקיים $g(n) = O(f(n))$ או $f(n) = O(g(n))$?
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" ($g(n)$ כגון $n, n\log n, n^2, \dots$) 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 12. הוכח: $\log(n!) = \Theta(n\log n)$

הוכחה:
מתקיים:

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

ולכן $\log(n!) \in O(n \log n)$.
בנוסף, מתקיים:

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

ולכן - $\log(n!) \in \Omega(n \log n)$. נסיק אם כן כי $\log(n!) = \Theta(n \log n)$ ■

4 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.