סעיף א (10 נקודות)

חשבו את הפרמטרים השונים של הלסטד. לארבעה הראשונים יש לפרט איך הגעתם למספר זה

η_1= 12 // public, static, int, <=, =, ;, *=, ++, (), {}, return, for נשים לב שאופרטור => הוא אחד ולא שניים. זה נכון גם לגבי אופרטור ++ ו =*. אמנם אפשר היה לפרק אותם לשניים ואז זה גם היה מגדיל את כמות השימוש במשתנים, אבל זה מנוגד לחשיבה של הלסטד שהסיבוכיות והסיכוי לבאגים נובעים מגודל התוכנה. זה כמובן גם מתקשר לחסרונות של השיטה.
η_2= 5 // n, result, 1, i, factorial
N_1= 22 // public:1, static:1, int:4, <=:1, =:2, ;:5, *=:1, ++:1, ():2, {}:2, return:1, for:1
N_2= 12 // n:2, result:3, 1:2, i:4, factorial: 1
η = 17 // η_1 + η_2
N= 34 // N_1 + N_2
V = 34*log2(17) = 138.973736603
Ň = 12*log2(12) + 5*log2(5) = 54.6291904831
D = (12/2)*(12/5) = 14.4
E = 14.4*138.973736603 = 2001.22180708
T = 2001.22180708/18 = 111.178989282
B = 138.973736603/3000 = 0.04632457886

סעיף ב (6 נקודות)

ציינו והסבירו בקצרה שתי חסרונות של שיטה זו

- No standards for a line of code
- Developed in the context of assembly languages and too fine grained for modern programming languages.
    - Highly biased by software language, developer's skills and experience
- The treatment of basic and derived measures is somehow confusing.

- The notions of time to develop and remaining bugs are arguable.
- Programming language and programmer skills are not incorporated.

שאלה 4 (30 נקודות) — Control-flow & Domain Testing

להלן מתודה:

```
1.  public void foo(int a, int b) {
2.    int z = a + b;
3.    if (a <= b + 5) {
4.      System.out.println("1");
5.      if (b > 2 * b) {
6.        System.out.println("2");
7.      } else if (b < 0) {
8.        System.out.println("3");
9.      } else {
10.       System.out.println("4");
11.     }
12.     if (z >= 2 * a) {
13.       System.out.println("5");
14.     }
15.     System.out.println("6");
16.   }
17.   System.out.println("7");
18. }
```