# מיני פרויקט בנושאים בגרפים רנדומיים

**נושא:** רכיבי קשירות בגרפים שבהם כל קשת נבחרת לתוך הגרף בהסתברות אוניפורמית וב״ת באחרות.

**מגישים:** אשחר מיכאל, ת"ז: 203841184.

גיל נגרין, ת"ז: 205730070.

<u>פתיח:</u> בספר The Probalistic Method, פרק 11 - The Erdos-Rényi Phase Transition,

מוצגות מספר טענות על גודל רכיבי קשירות בגרפים רנדומיים בעלי n קודקודים באשר לכל צלע הסתברות של p להופיע. הטענות אותם נבדוק מתייחסות ל5 תחומים שונים של p עבור n-ים מספיק גדולים.

בכדי לבחון טענות אלו רשמנו תכנית ב-Java אשר בודקת את גודל רכיבי הקשירות עבור ערכי ה-p השונים – כל טענה עבור מספר גרפים בגדלים שונים.

במקום לעבור על כל $\binom{n}{2}$ צלעות אפשריות, נבחר בצורה אחידה 2 קודקודים ונוסיף את הקשת ביניהם לגרף, נחזור על תהליך זה $\binom{n}{2} * p$ פעמים שזו התוחלת של הצלעות בגרף.

לכן ההסתברות שצלע תופיע בגרף היא $p \approx$, וההסתברות שאותה צלע תיבחר פעמיים זניחה ולכן ניתן להתעלם ממנה.

נגדיר: 1. $p = \frac{c}{n}$ – the coarse parametrization

2. $p = \frac{1}{n} + \lambda n^{-\frac{4}{3}}$ - the fine parametrization

חמשת תחומי ה-p אותם בחנו הם:

א. **Very Subcritical**, נשתמש ב – 1, ונניח כי c הוא קבוע כך ש – c < 1 לדוגמה: $p = \frac{1}{2n}$.

ב. **Barely Subcritical**, נשתמש ב – 2, $p = \frac{1-\varepsilon}{n}$ כך ש $\varepsilon = \lambda n^{-\frac{1}{3}}$ נניח כי $\epsilon =$ $O(1)$ וכי $\lambda \to \infty$, לדוגמה: $p = \frac{1}{n} - n^{-\frac{4}{3}} * n^{0.01}$.

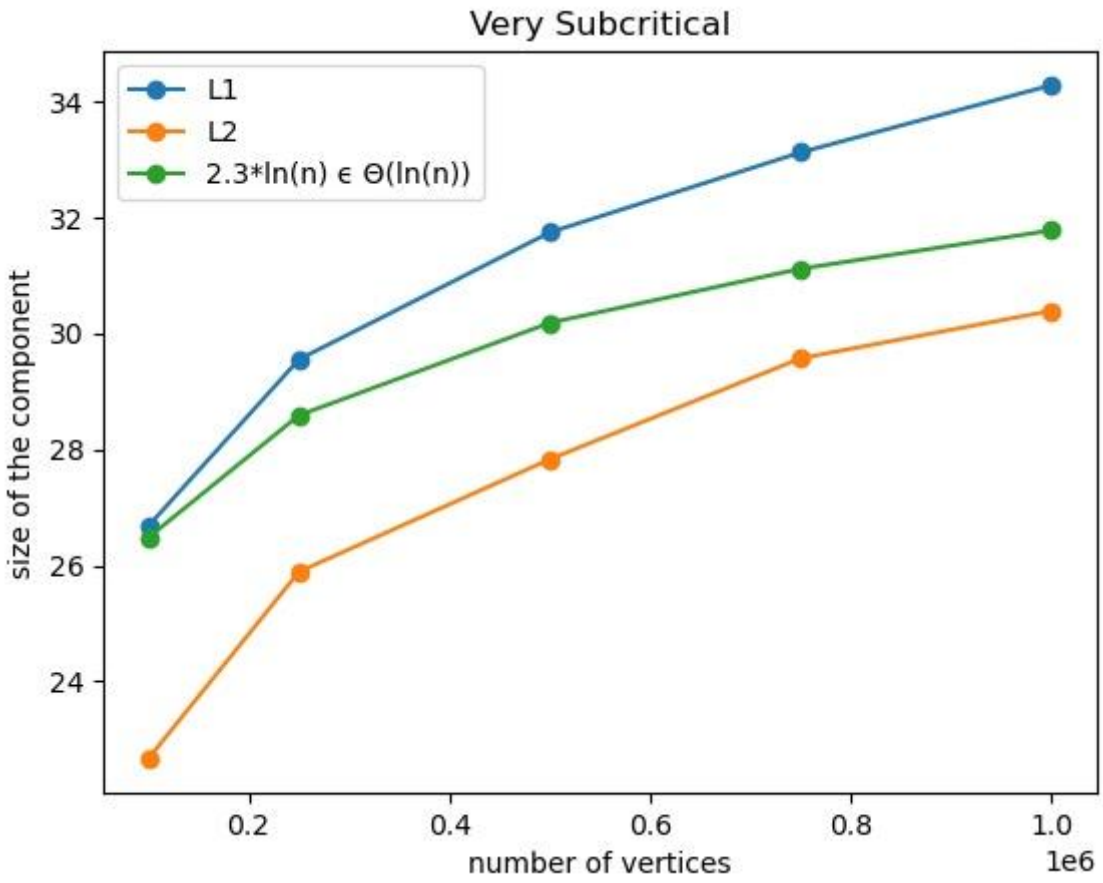
ג. **The Critical Window**, נשתמש ב – 2, כאן $\lambda$ הינו קבוע וכאשר $\lambda = 0$ אינה מתנהגת בצורה מיוחדת, לכן נבחן כאשר $\lambda \neq 0$ מלמעלה ומלמטה לדוגמה: $p = \frac{1}{n} \pm 2n^{-\frac{4}{3}}$.

ד. **Barely Supercritical**, נשתמש ב – 2, $p = \frac{1+\epsilon}{n}$ כאשר $\epsilon = \lambda n^{-\frac{1}{3}}$ נניח כי $\epsilon =$ $O(1)$ וכי $\lambda \to \infty$, לדוגמה: $p = \frac{1}{n} + n^{-\frac{4}{3}} n^{0.01}$.

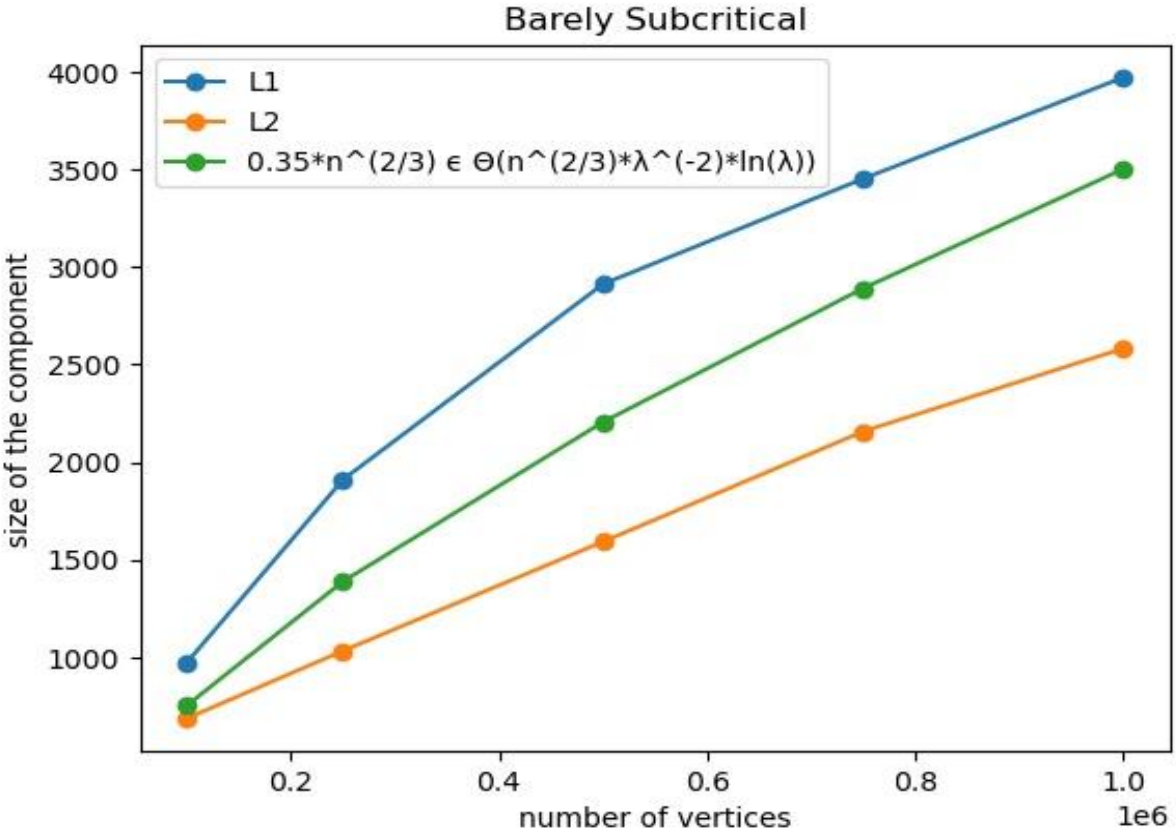
ה. **Very Supercritical**, נשתמש ב – 1 ונניח כי $c > 1$ לדוגמה: $p = 2/n$ .

נציג כעת את התוצאות אשר התקבלו מהתוכנית שרשמנו ע"י ייצוג בגרף:

א. **<u>Very Subcritical:</u>**

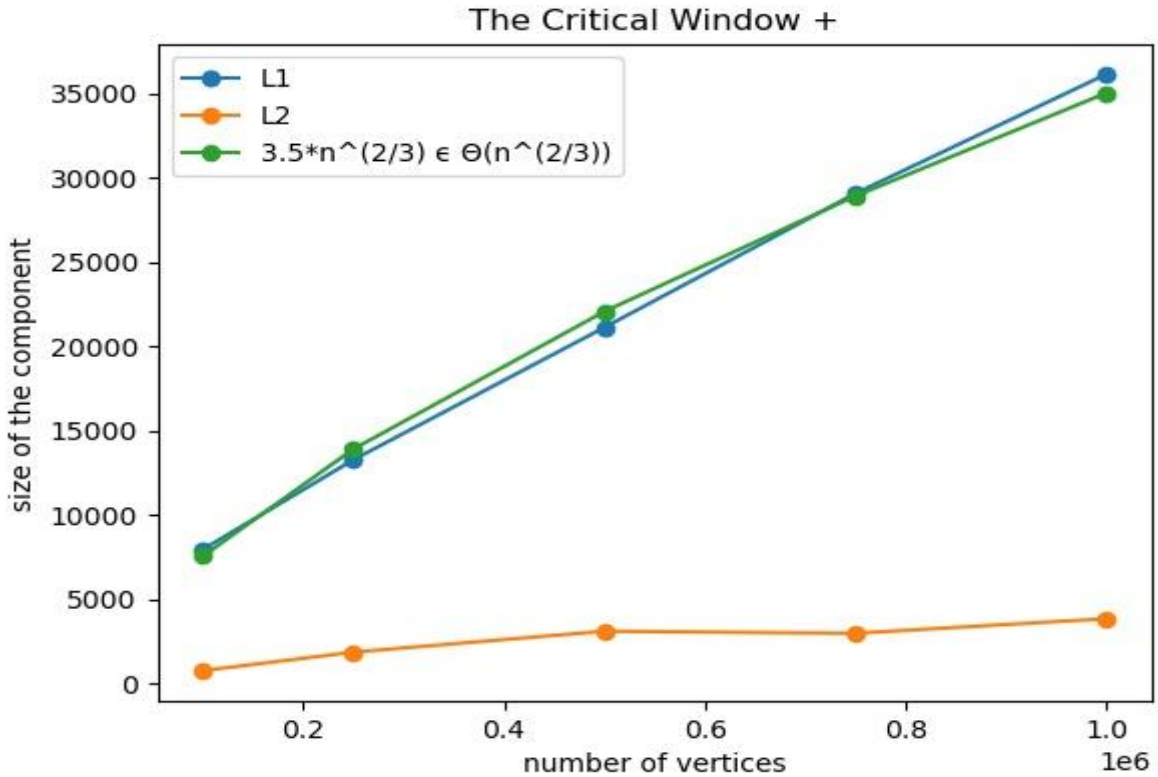


ב. **Barely Subcritical:**

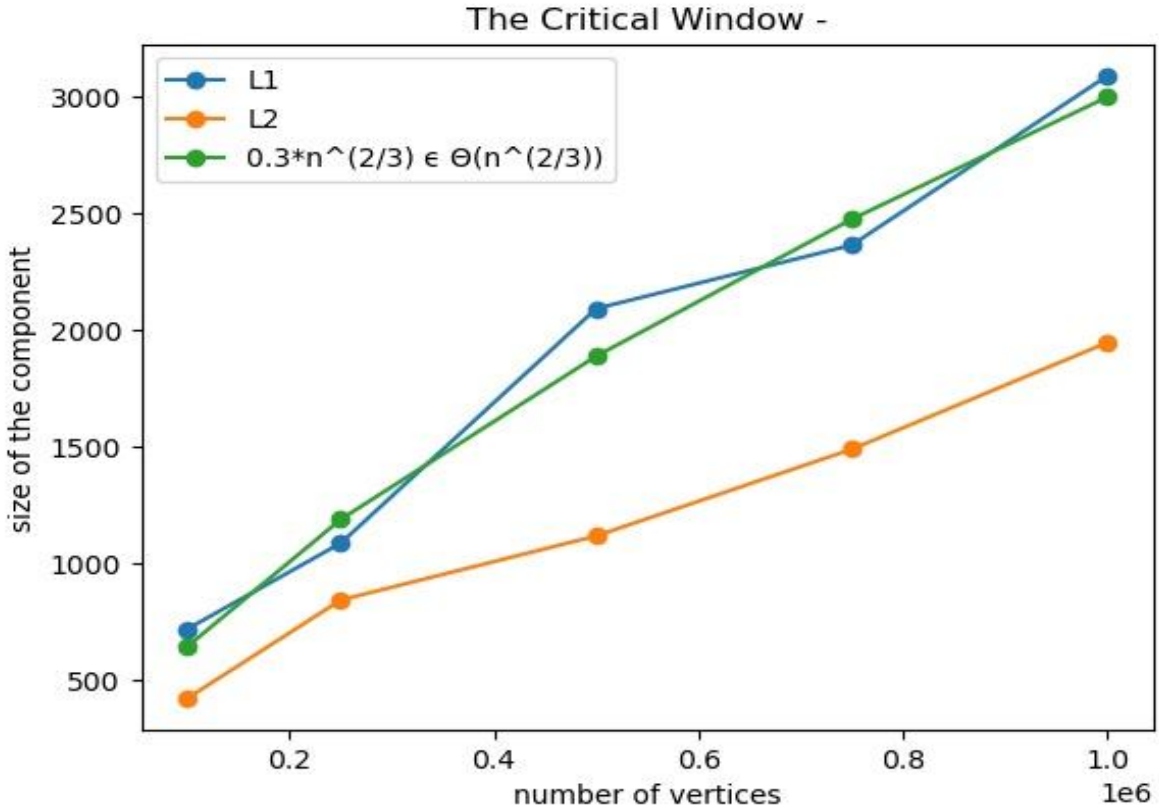



ג. **The Critical Window:**

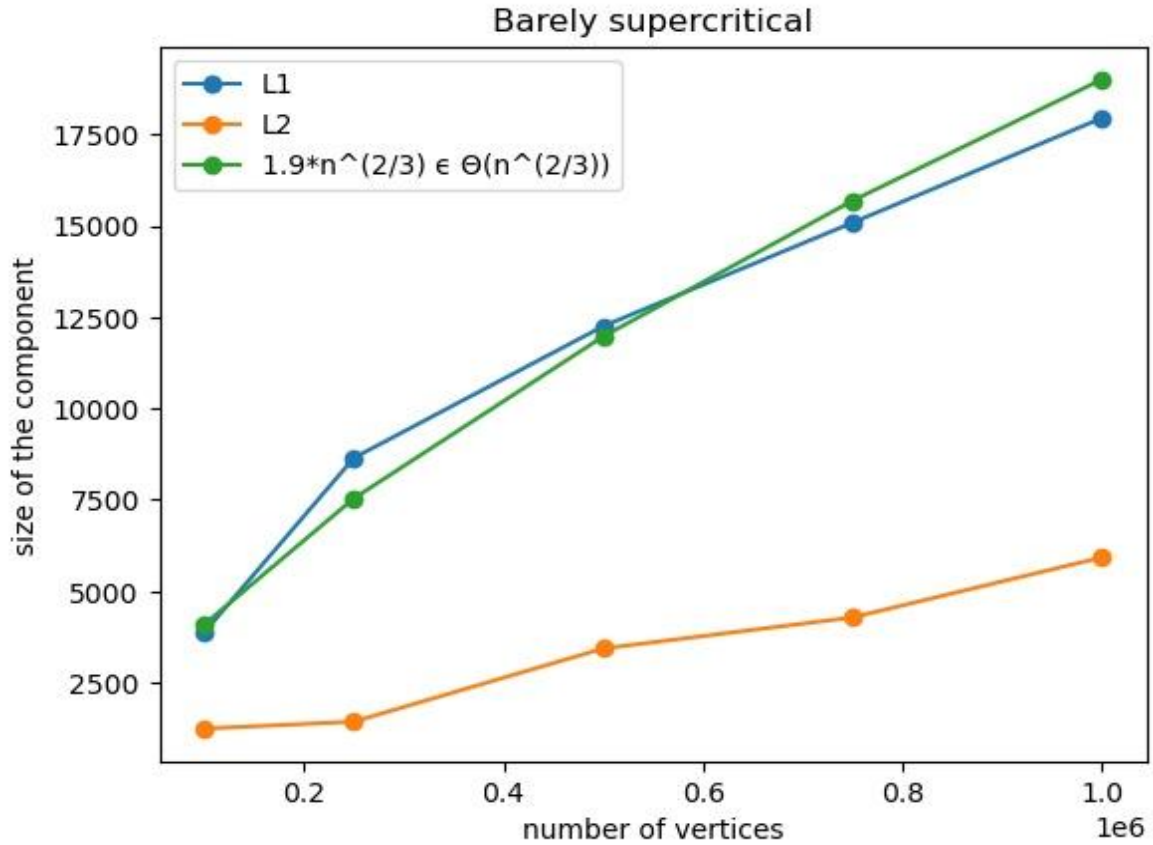
מקרה 1: $\lambda > 0$



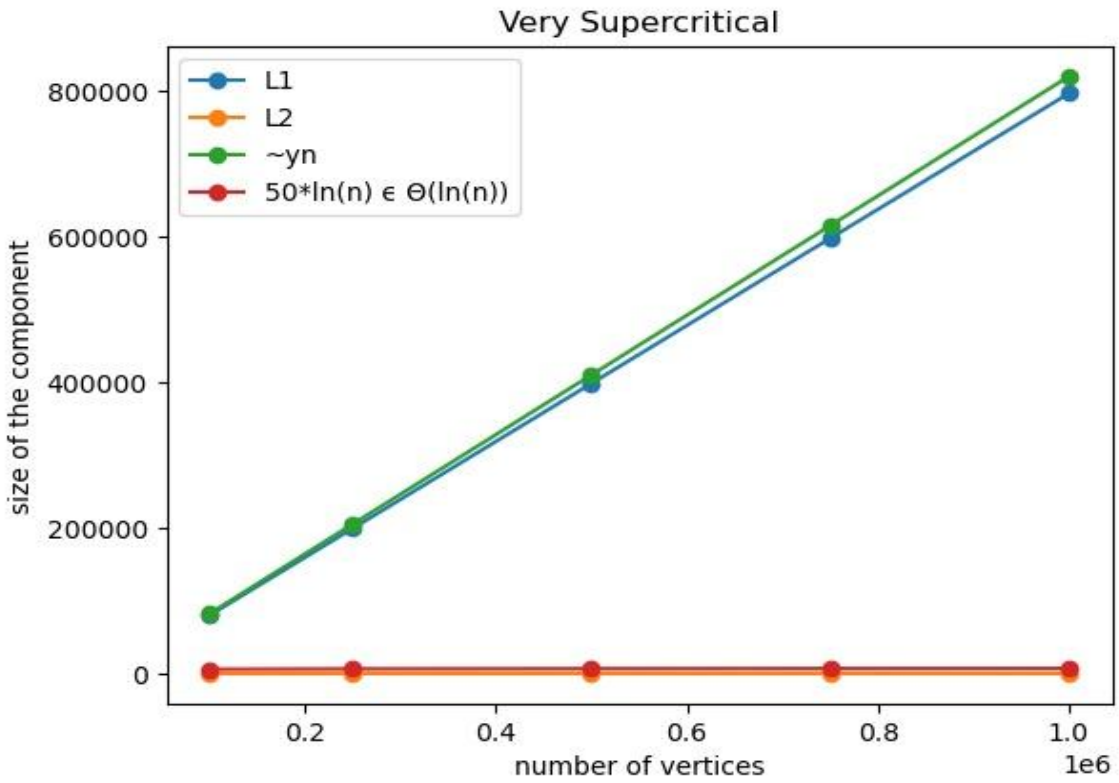
מקרה 2: $\lambda < 0$




ד. ***Barely Supercritical:***



ה. **Very Supercritical:**




**מסקנות:**

על פי תוצאות הניסוי שערכנו, ניתן להסיק את המסקנות הבאות:

א. *Very Subcritical*: $L_1 = \theta(\ln(n))$ וגם $L_1 \sim L_2$.

ב. *Barely Subcritical*: $L_1 = \theta(n^{\frac{2}{3}} * \lambda^{-2} * \ln(\lambda))$ וגם $L_1 \sim L_2$.

ג. *The Critical Window*: מקרה 1, $\lambda > 0$: $L_1 = \theta(n^{\frac{2}{3}})$, $L_1 \gg L_2$.

מקרה 2, $\lambda < 0$: $L_1 \sim L_2 = \theta(n^{\frac{2}{3}})$.

ד. *Barely Supercritical*: $L_1 \sim 2\lambda n^{\frac{2}{3}}$.

ה. *Very Supercritical*: $L_1 \sim yn$ כאשר $y = y(c)$ הוא קבוע חיובי ממשי המספק את המשוואה $e^{-cy} = 1 - y$.

**מקורות:**


***N. Alon and J. Spencer, The Probabilistic Method***

## הקוד בג'אווה:

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.Random;

public class miniProj {

    //finding the component representative
    public static int findRep(int[] represents, int index) {
        int original_index = index;
        while (represents[index] != index) {
            index = represents[index];
        }
        represents[original_index] = index;
        return index;
    }

    static long findNcR(int n, int r) {
        long p = 1, k = 1;
        if (n - r < r) {
            r = n - r;
        }
        if (r != 0) {
            while (r > 0) {
                p *= n;
                k *= r;

                long m = __gcd(p, k);

                p /= m;
                k /= m;

                n--;
                r--;
            }
        }
        else {
            p = 1;
        }
        return p;
    }

    static long __gcd(long n1, long n2)
    {
        long gcd = 1;
        for (int i = 1; i <= n1 && i <= n2; ++i) {
            if (n1 % i == 0 && n2 % i == 0) {
                gcd = i;
            }
        }
        return gcd;
    }

    public static int[] connections(int n, double p, long edges) {
        int[] represents = new int[n];
        int[] component_size = new int[n];
        long edges_left = edges;
        int rep_j, rep_i;
        Arrays.fill(component_size, 0);
        for (int i = 0; i < n; i++) {
```

```
            represents[i] = i;
        }
        Random rand = new Random();
        while(edges_left > 0) {
            int j = rand.nextInt(n);
            int i = rand.nextInt(n);

            //checking for the representative of the component
            if (represents[j] == represents[represents[j]])
                rep_j = represents[j];
            else
                rep_j = findRep(represents, j);

            if (represents[i] == represents[represents[i]])
                rep_i = represents[i];
            else
                rep_i = findRep(represents, i);
            //update the representative of the current vertices
            represents[rep_j] = rep_i;
            edges_left--;
        }
        //sum up the components size
        for (int i = 0; i < n; i++) {
            int rep = findRep(represents,i);
            component_size[rep]++;
        }

        Arrays.sort(component_size);
        return new int[] {component_size[n-1], component_size[n-2]};
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("Enter Number of Vertices: ");
        int vertices = in.nextInt();
        double[] prob = new double[] {1.0/(2.0*vertices),
                1.0/vertices - Math.pow(vertices,(-4.0/3.0)) *
Math.pow(vertices,(0.01)),
                1.0/vertices + 2*Math.pow(vertices,(-4.0/3.0)),
                1.0/vertices - Math.pow(vertices,(-4.0/3.0)),
                1.0/vertices + Math.pow(vertices,(-4.0/3.0)) *
Math.pow(vertices,(0.01)),
                2.0/vertices};
        System.out.println("Choose probability option:\n" +
                "1) Very subcritical \n" +
                        "2) Barely subcritical\n" +
                        "3) Critical window +\n" +
                        "4) Critical window -\n" +
                        "5) Barely supercritical\n" +
                        "6) Very supercritical"
                );
        double p = prob[in.nextInt() - 1];
        System.out.println(p);
        long edges = (long)((findNcR(vertices,2)) * p);
        System.out.println(edges);
        System.out.println("Enter number of times to run: ");
        int times = in.nextInt();
        double largest = 0;
        double second = 0;
        for (int i = 0; i < times; i++) {
            int[] ret = connections(vertices, p, edges);
```

```
            largest += ret[0];
            second += ret[1];
            System.out.println("Run number: " + (i + 1));
            System.out.println("L1 = " + ret[0]);
            System.out.println("L2 = " + ret[1]);

System.out.println("##############################################################
####");
        }
        System.out.println("Average largest component: " + largest/times);
        System.out.println("Average second largest component: " + second/times);
    }
}
```