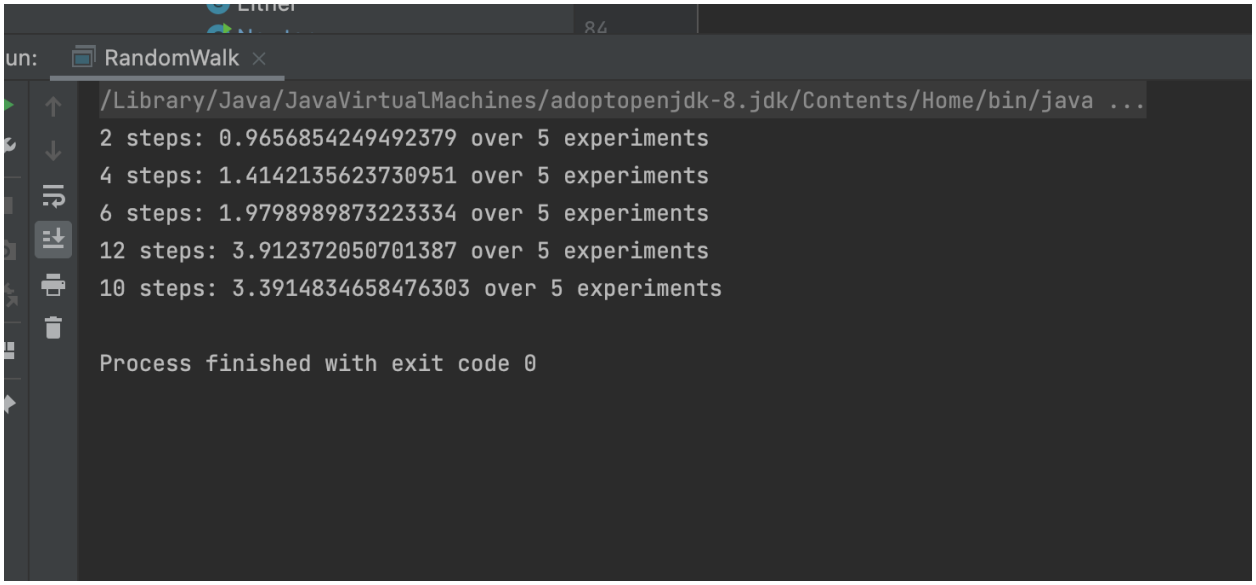
Program Structures & Algorithms
Spring 2022
Assignment No. 1 (Random Walk)

Name: Sri Vyshnavi Kotha
(NUID): 002985810

Task

Deduce the relationship between *d* and *n* and implement the code for the experiment

Output screenshot

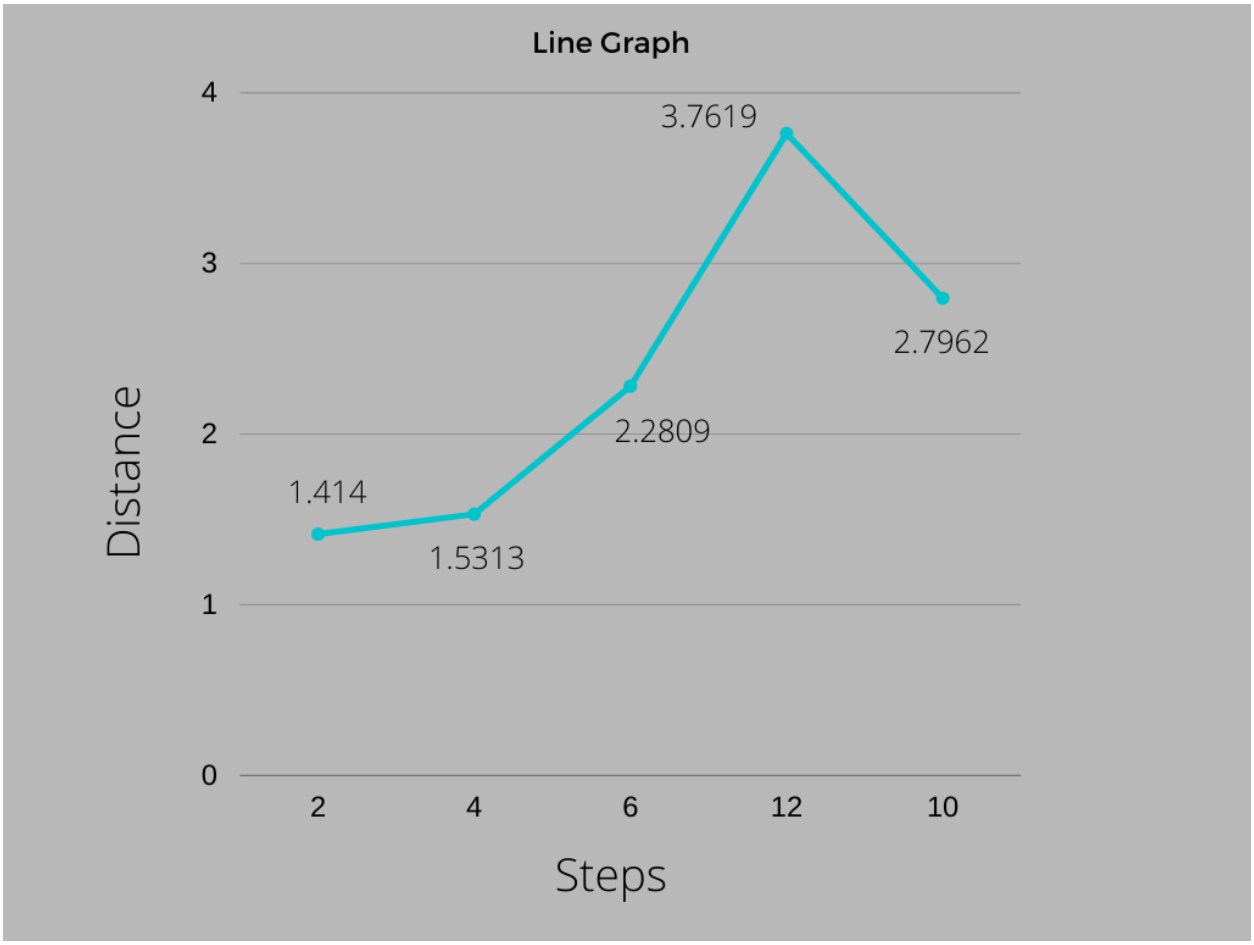


Relationship Conclusion

If ‘d’ is the distance and ‘n’ is the number of steps then **d=√(n) or d = sqrt(n)** Is the relationship deduced.

Evidence / Graph

Below graph depicts the relation between the distance and number of random steps taken by a man for different values of steps (2,4,6,12,10)

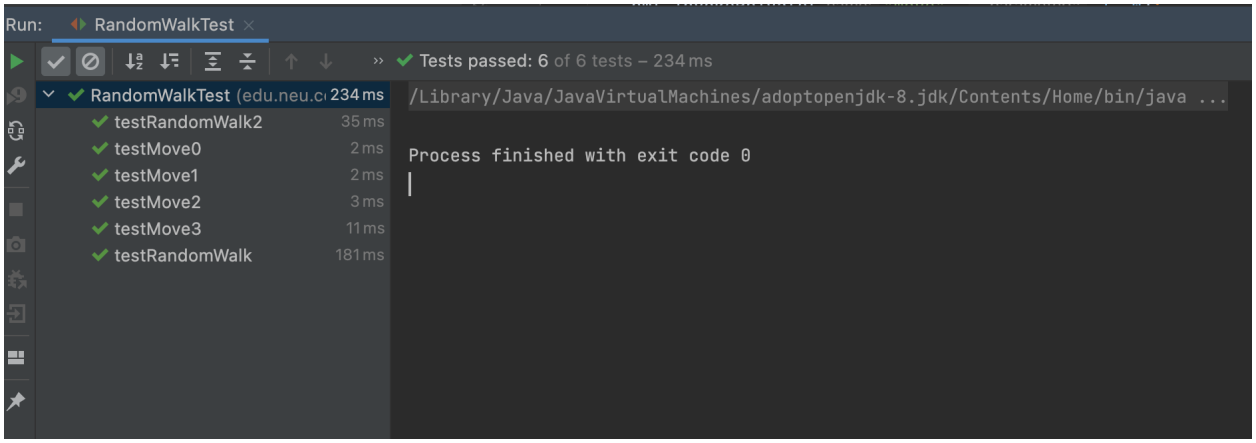


It is observed that the distance is approximately close to the sqrt(number of steps) which satisfies the relationship established.

Ex:

No. of steps (n) = 2
Distance d = sqrt(2) = 1.414 approx.

Unit tests result



Code

```
/*
* Copyright (c) 2017. Phasmid Software
*/

package edu.neu.coe.info6205.randomwalk;

import java.util.Random;

public class RandomWalk {

    private int x = 0;
    private int y = 0;

    private final Random random = new Random();

    /**
     * Private method to move the current position, that's to say the drunkard
moves
     *
     * @param dx the distance he moves in the x direction
     * @param dy the distance he moves in the y direction
     */
    private void move(int dx, int dy) {
        this.x =  this.x + dx;
        this.y =  this.y + dy;
    }

    /**
     * Perform a random walk of m steps
     *
```

```
    * @param m the number of steps the drunkard takes
    */
   private void randomWalk(int m) {
       for (int i = 0; i <m; i++) {
           randomMove();
       }
   }

   /**
    * Private method to generate a random move according to the rules of the
situation.
    * That's to say, moves can be (+-1, 0) or (0, +-1).
    */
   private void randomMove() {
       boolean ns = random.nextBoolean();
       int step = random.nextBoolean() ? 1 : -1;
       move(ns ? step : 0, ns ? 0 : step);
   }

   /**
    * Method to compute the distance from the origin (the lamp-post where the
drunkard starts) to his current position.
    *
    * @return the (Euclidean) distance from the origin to the current
position.
    */
   public double distance() {
       int x1 = 0, y1 = 0;           //Distance formula - Sqrt((x2-x1)^2 +
(y2-y1)^2)
       return Math.sqrt((Math.abs(this.x - x1))*(Math.abs(this.x - x1)) +
(Math.abs(this.y - y1))*(Math.abs(this.y - y1)));
   }

   /**
    * Perform multiple random walk experiments, returning the mean distance.
    *
    * @param m the number of steps for each experiment
    * @param n the number of experiments to run
    * @return the mean distance
    */
   public static double randomWalkMulti(int m, int n) {
       double totalDistance = 0;
       for (int i = 0; i < n; i++) {
           RandomWalk walk = new RandomWalk();
           walk.randomWalk(m);
           totalDistance = totalDistance + walk.distance();
       }
       return totalDistance / n;
   }
```

```
    public static void main(String[] args) {
        int[] rmWlk = new int[] { 2, 4, 6, 12, 10 };
        int n = 5;
        if (args.length > 1) n = Integer.parseInt(args[1]);
        for(int m=0; m < rmWlk.length;m++) {
            double meanDistance = randomWalkMulti(rmWlk[m], n);
            System.out.println(rmWlk[m] + " steps: " + meanDistance + " over "
+ n + " experiments ");
        }
    }
}
```