Program Structures and Algorithms
Fall 2024

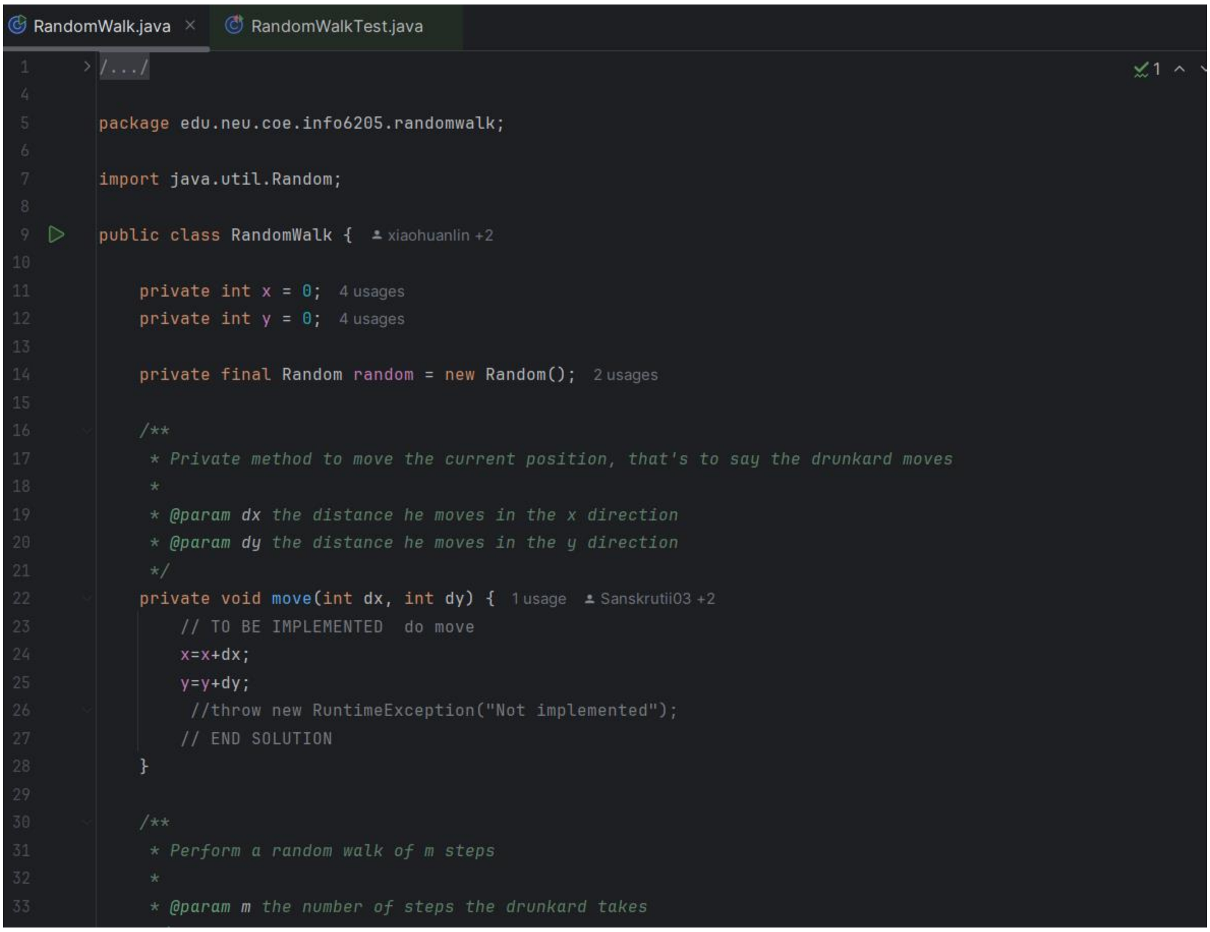
NAME: Sanskruti Manoria
NUID: 002643300
GITHUB LINK: https://github.com/sannskruti/INFO6205

Assignment 1

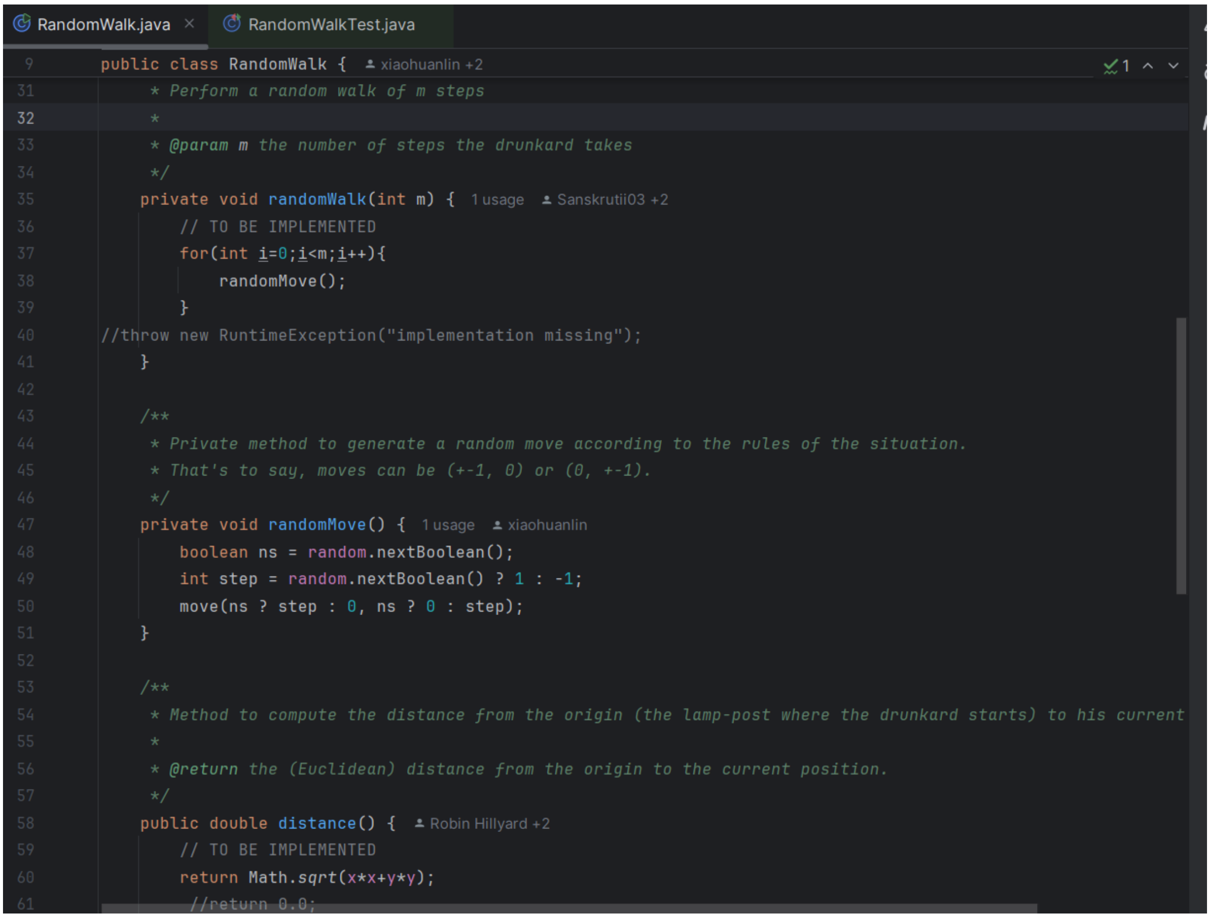
Task:

1. Your **conclusion** about the relationship between *d* and *m*;
2. Your **evidence** to support that relationship (screen shot and/or graph and/or spreadsheet)
3. Your **code** (*RandomWalk.java* plus anything else that you changed or created);
4. A **screen shot** of the unit tests all passing.

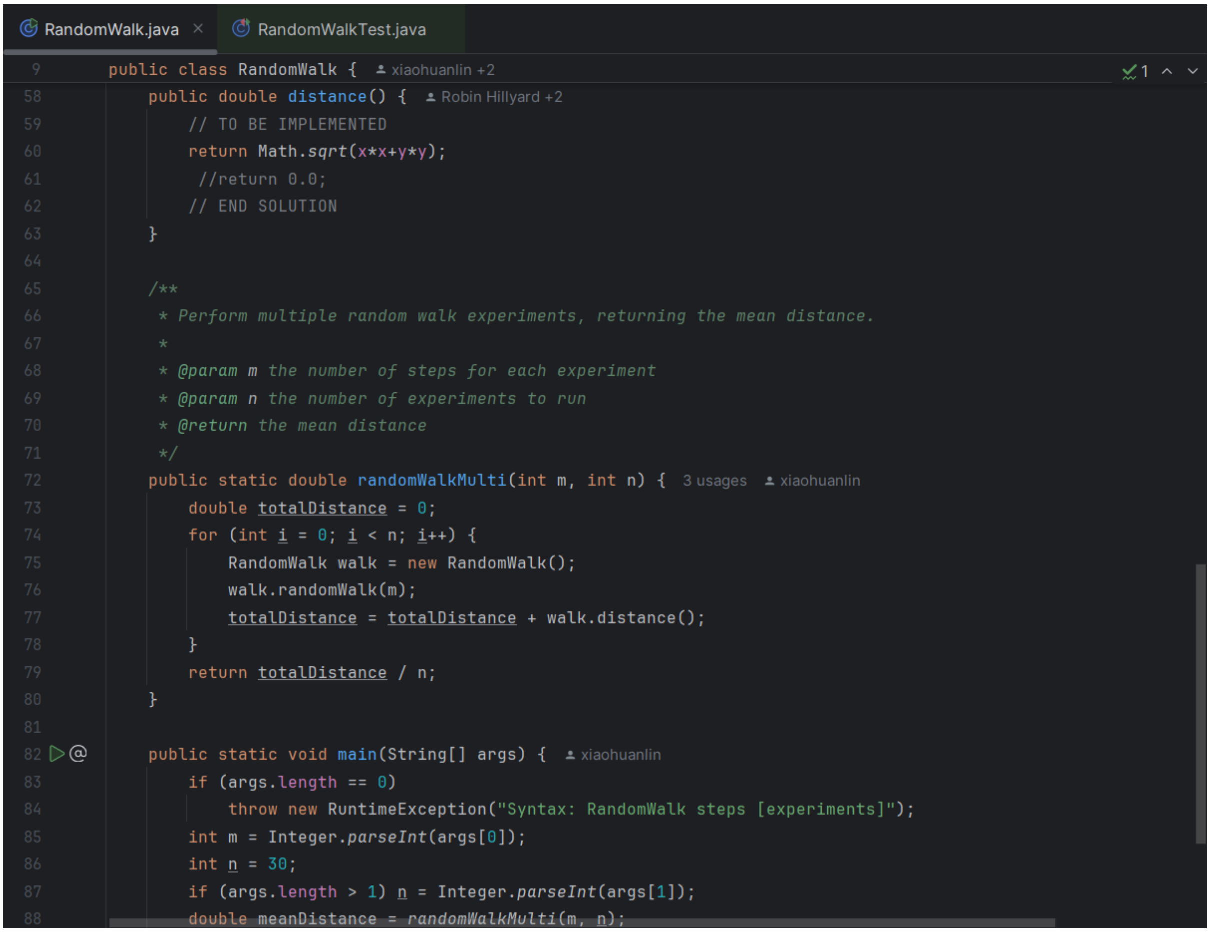
Code Screenshots:



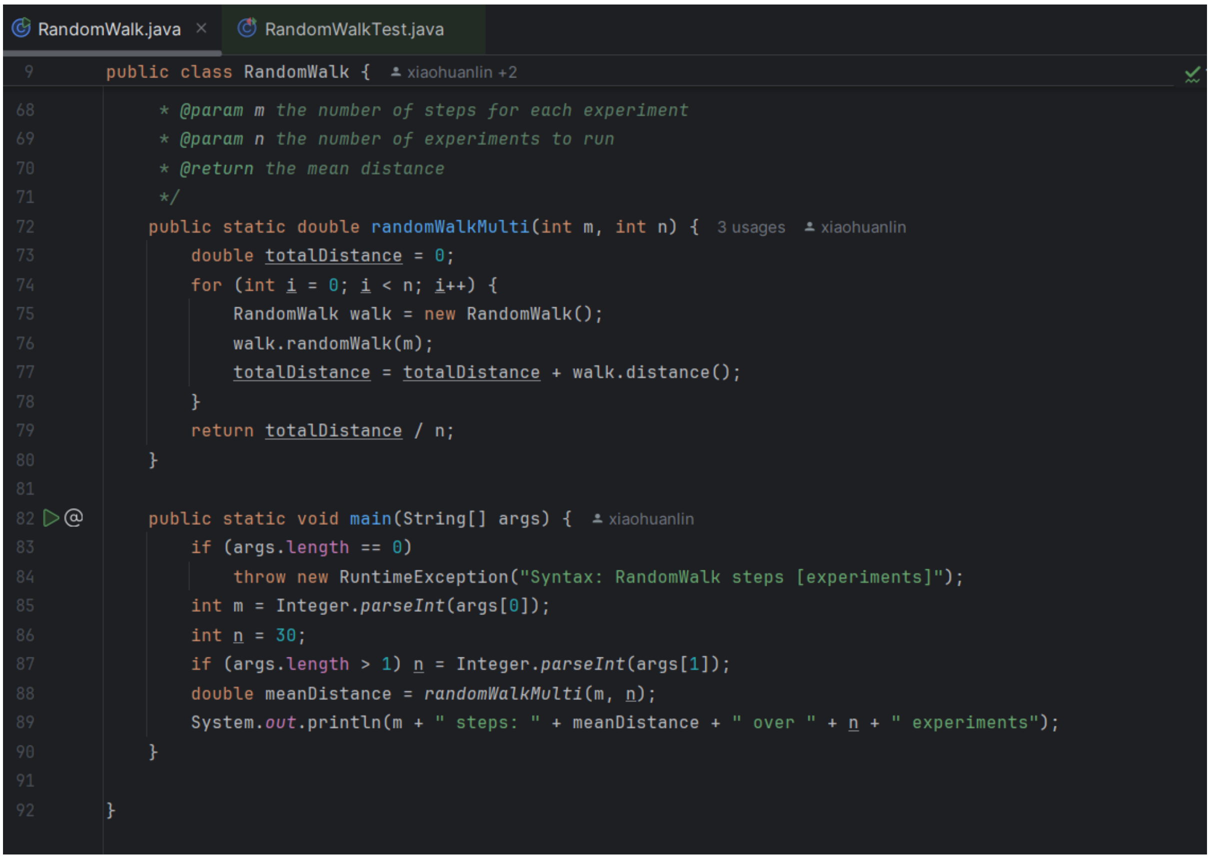
```
/.../

package edu.neu.coe.info6205.randomwalk;

import java.util.Random;

public class RandomWalk {  xiaohuanlin +2

    private int x = 0;  4 usages
    private int y = 0;  4 usages

    private final Random random = new Random();  2 usages

    /**
     * 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) {  1 usage  Sanskrutii03 +2
        // TO BE IMPLEMENTED  do move
        x=x+dx;
        y=y+dy;
         //throw new RuntimeException("Not implemented");
        // END SOLUTION
    }

    /**
     * Perform a random walk of m steps
     *
     * @param m the number of steps the drunkard takes
```



```
public class RandomWalk {  xiaohuanlin +2
     * Perform a random walk of m steps
     *
     * @param m the number of steps the drunkard takes
     */
    private void randomWalk(int m) {  1 usage  Sanskrutii03 +2
        // TO BE IMPLEMENTED
        for(int i=0;i<m;i++){
            randomMove();
        }
//throw new RuntimeException("implementation missing");
    }


    /**
     * 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() {  1 usage  xiaohuanlin
        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
     *
     * @return the (Euclidean) distance from the origin to the current position.
     */
    public double distance() {  Robin Hillyard +2
        // TO BE IMPLEMENTED
        return Math.sqrt(x*x+y*y);
         //return 0.0;
```

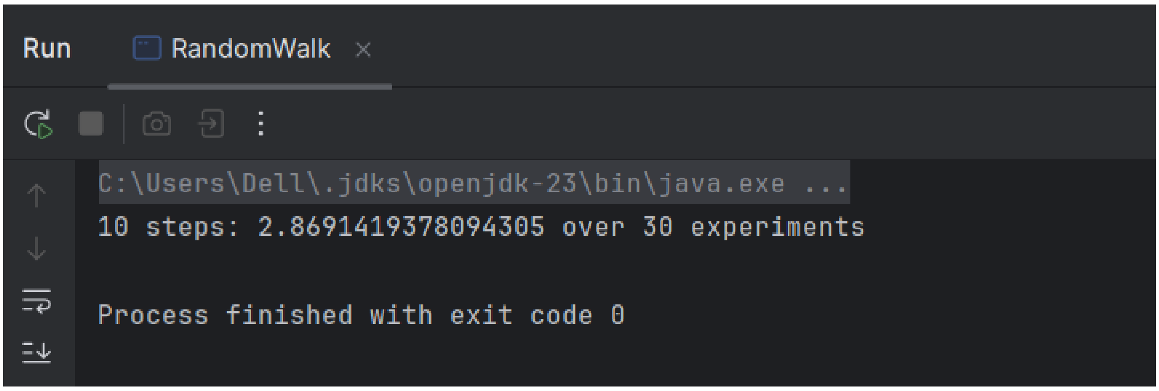


```java
public class RandomWalk {
    public double distance() {
        // TO BE IMPLEMENTED
        return Math.sqrt(x*x+y*y);
         //return 0.0;
        // END SOLUTION
    }

    /**
     * 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) {
        if (args.length == 0)
            throw new RuntimeException("Syntax: RandomWalk steps [experiments]");
        int m = Integer.parseInt(args[0]);
        int n = 30;
        if (args.length > 1) n = Integer.parseInt(args[1]);
        double meanDistance = randomWalkMulti(m, n);
```

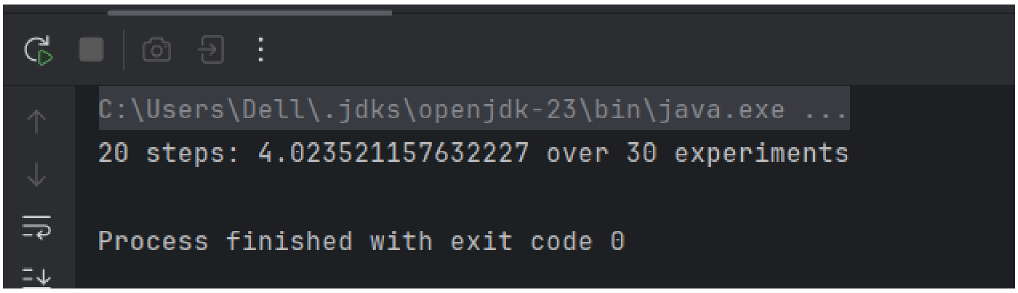


```java
public class RandomWalk {
     * @param m the number of steps for each experiment
     * @param n the number of experiments to run
     * @return the mean distance
     */
    public static double randomWalkMulti(int m, int n) {
        double totalDistance = 0;
        for (int i = 0; i < n;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

    public static void main(String[] args) {
        if (args.length == 0)
            throw new RuntimeException("Syntax: RandomWalk steps [experiments]");
        int m = Integer.parseInt(args[0]);
        int n = 30;
        if (args.length > 1) n = Integer.parseInt(args[1]);
        double meanDistance = randomWalkMulti(m, n);
        System.out.println(m + " steps: " + meanDistance + " over " + n + " experiments");
    }

}
```

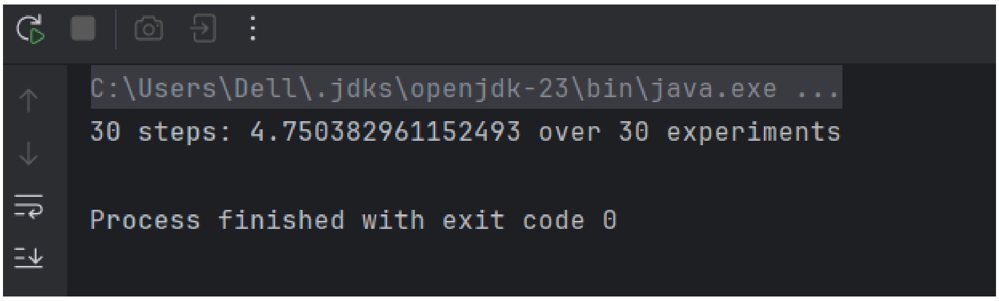
Output-



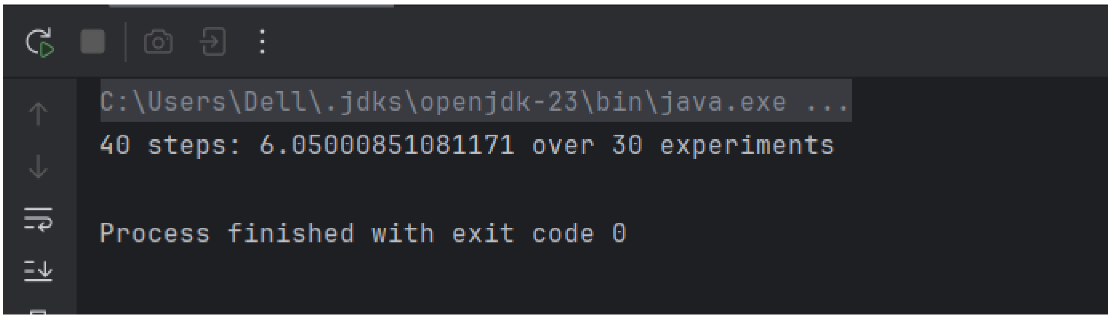
```
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
10 steps: 2.8691419378094305 over 30 experiments

Process finished with exit code 0
```



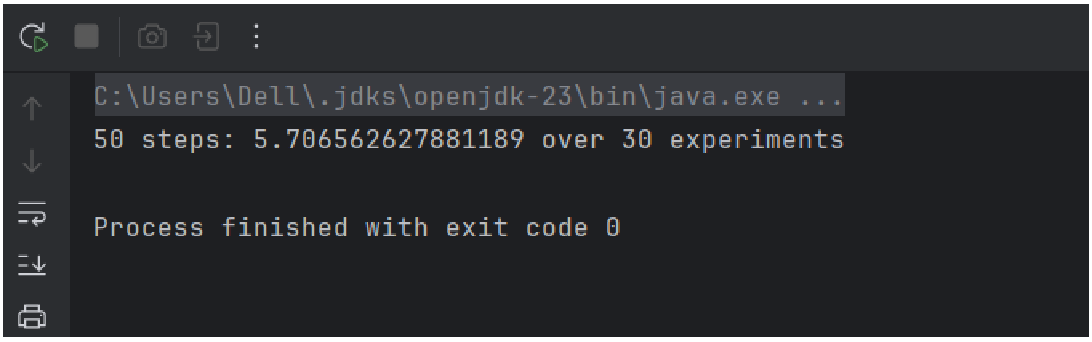
```
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
20 steps: 4.023521157632227 over 30 experiments

Process finished with exit code 0
```



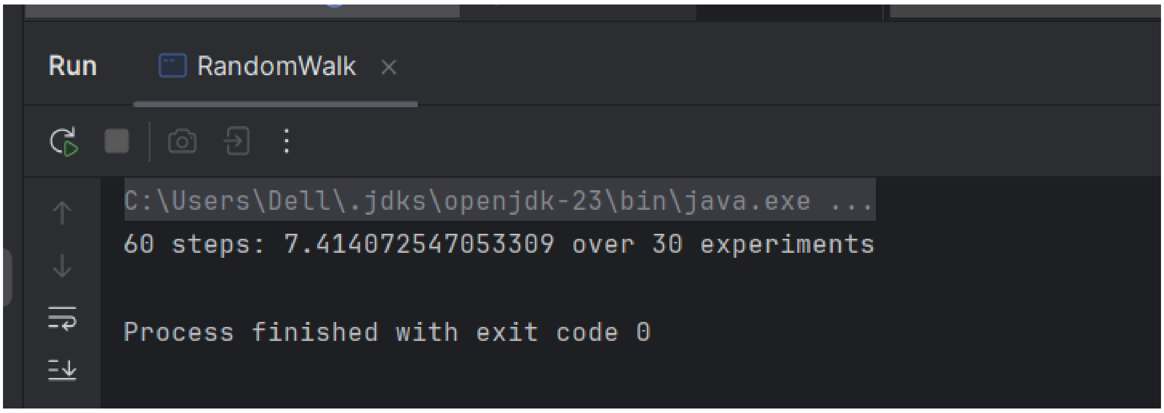
```
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
30 steps: 4.750382961152493 over 30 experiments

Process finished with exit code 0
```



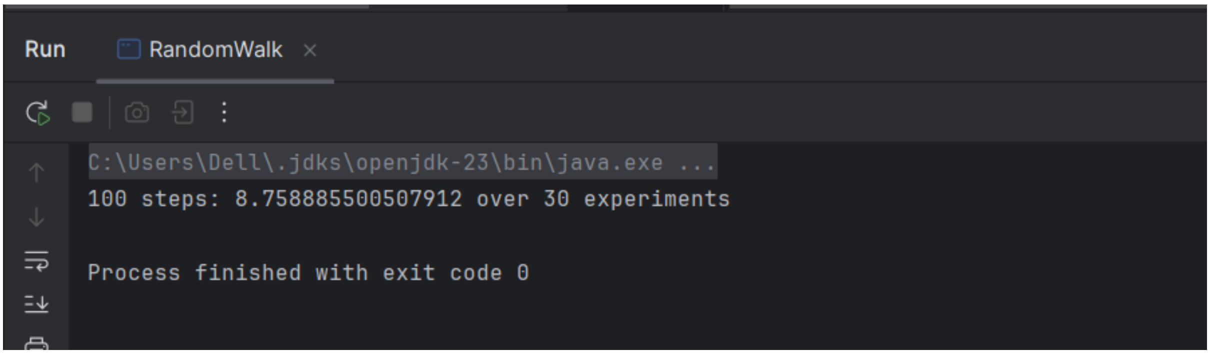
```
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
40 steps: 6.05000851081171 over 30 experiments

Process finished with exit code 0
```



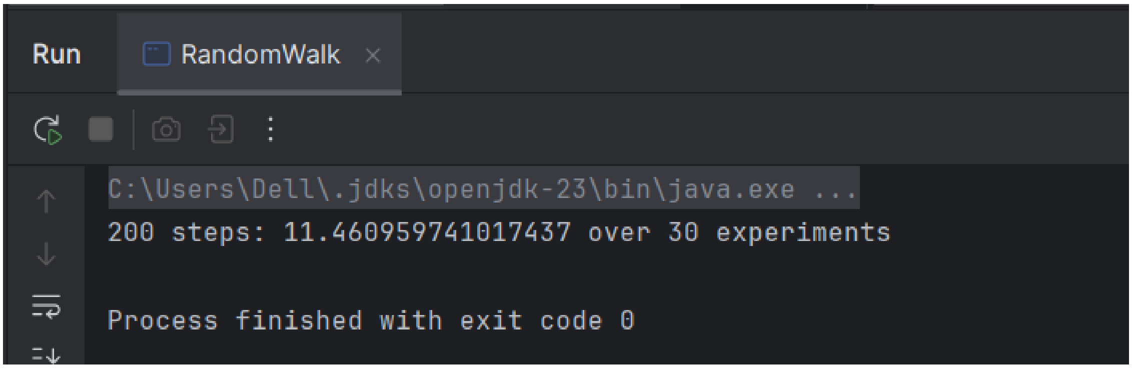
```
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
50 steps: 5.706562627881189 over 30 experiments

Process finished with exit code 0
```



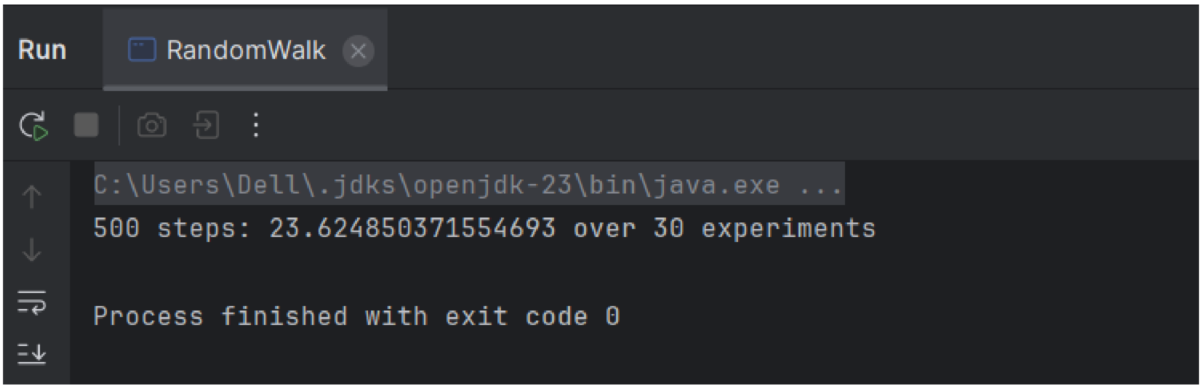
```
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
60 steps: 7.414072547053309 over 30 experiments

Process finished with exit code 0
```



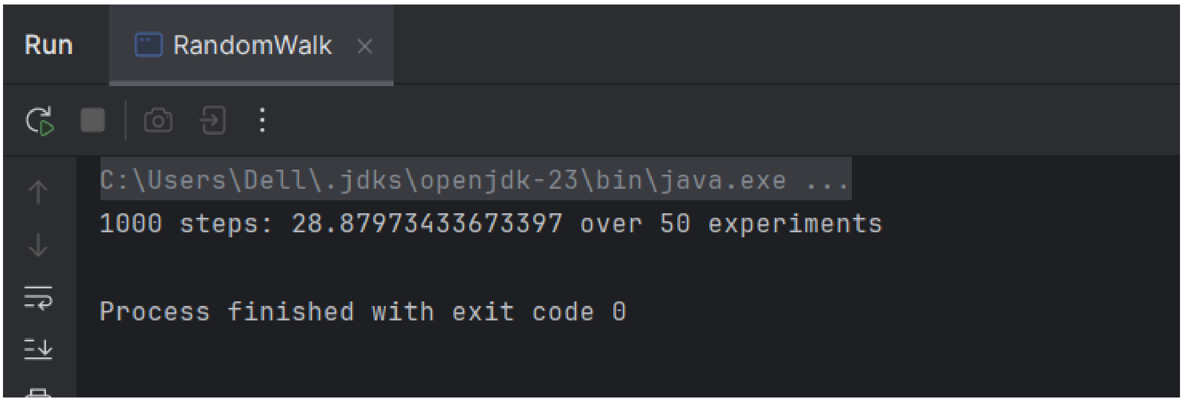
```
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
100 steps: 8.758885500507912 over 30 experiments

Process finished with exit code 0
```



```
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
200 steps: 11.460959741017437 over 30 experiments

Process finished with exit code 0
```

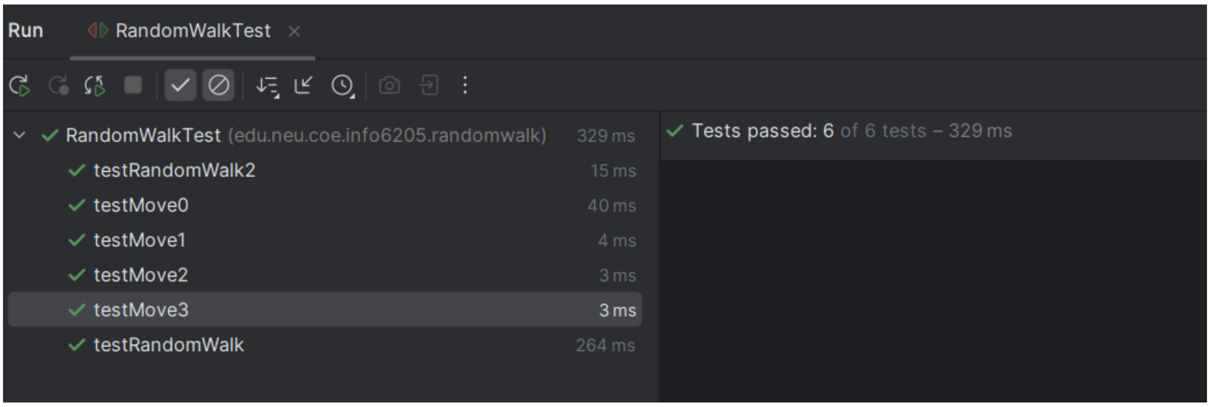


```
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
500 steps: 23.624850371554693 over 30 experiments

Process finished with exit code 0
```



```
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
1000 steps: 28.87973433673397 over 50 experiments

Process finished with exit code 0
```

Unit Test Screenshots:



```
✓ RandomWalkTest (edu.neu.coe.info6205.randomwalk)   329 ms
    ✓ testRandomWalk2                                  15 ms
    ✓ testMove0                                        40 ms
    ✓ testMove1                                         4 ms
    ✓ testMove2                                         3 ms
    ✓ testMove3                                         3 ms
    ✓ testRandomWalk                                  264 ms
```

✓ Tests passed: 6 of 6 tests – 329 ms

Observations:

A drunk man is standing at a lamp post. He's feeling wobbly and starts to take steps. He can take steps in four directions: North, South, East, or West. Every time he takes a step, it’s in one of those directions, but we don’t know which one because it’s totally random.
Now, after he takes 1 step, he’s moved a little bit away from the lamp post.

After he takes 10 steps, you might think he’ll be really far, But Some of those steps might cancel each other out, like one step North followed by one step South. So, he won’t be as far away as if he had walked straight in one direction.

How far is he from the lamp post after taking a lot of random steps?

The Answer: d ~ √ (m)

If he takes 100 steps, his distance isn’t 100 meters away. It’s more like 10 meters away because:

d~√100=10

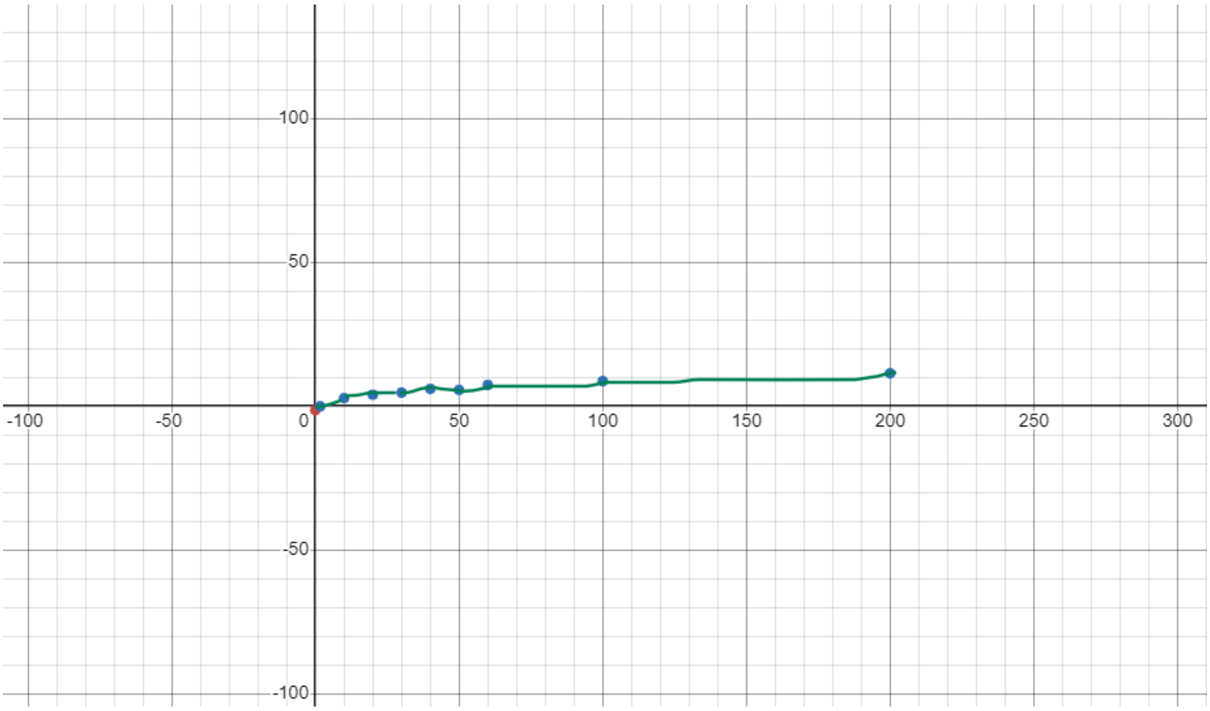
If he takes 400 steps, his distance is around:
d~√400= 20
So, the more steps he takes, the farther he gets from the lamp post, but not in a straight line. His distance grows **slower** than the number of steps.

Conclusion:

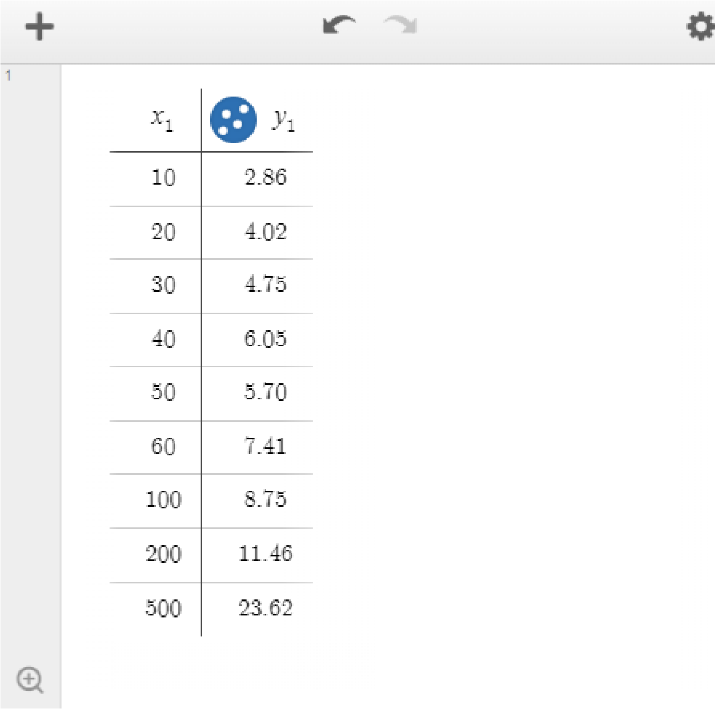
The relationship between the drunk man's distance d and the number of steps m is:
d~√m

This means that if he takes 4 times as many steps, he'll only get twice as far. That's because random steps in different directions start to balance each other out, so the man doesn't get as far as you'd expect if he walked in just one direction.

Evidence:



y=k*√x where k is constant k=1,2,3,….

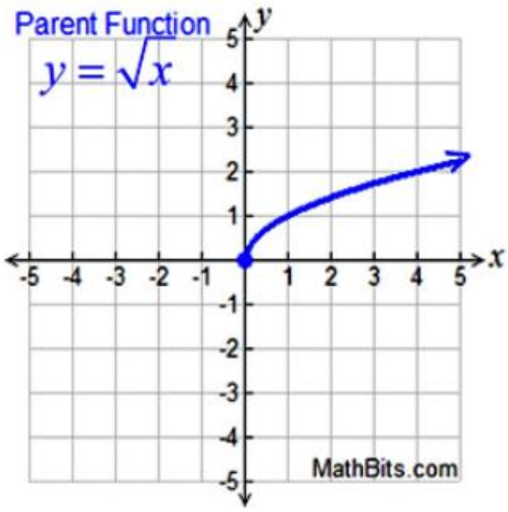


x_1	y_1
10	2.86
20	4.02
30	4.75
40	6.05
50	5.70
60	7.41
100	8.75
200	11.46
500	23.62

Where x1 is m steps and y1 is d distance. (check plotting points from above output screenshot where n =30)

for graph= desmos.com

Graph of sqrt is :


Parent Function
$y = \sqrt{x}$
y
x
MathBits.com