**INTRODUCTION TO JAVA**

**ACTIVITY 4: FUNCTIONS (PART. 2)**

**14. AnimatedHtree**

OUTPUT:

//BASE CASE

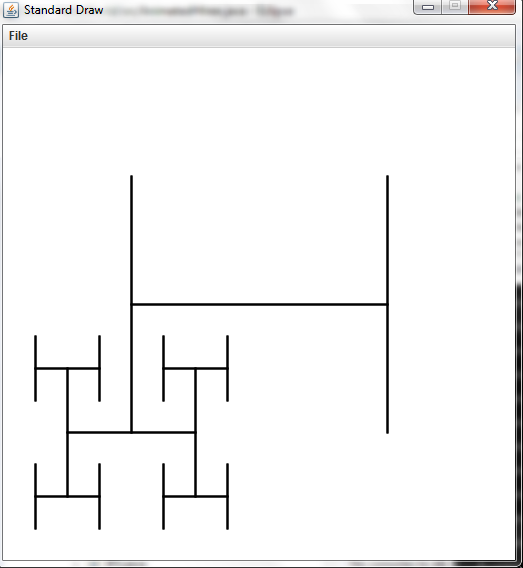
draw(n-1, x0, y0, size/2); // lower left

draw(n-1, x0, y1, size/2); // upper left

draw(n-1, x1, y0, size/2); // lower right

draw(n-1, x1, y1, size/2); // upper right

The animation start from lower left to upper right



//CASE 2

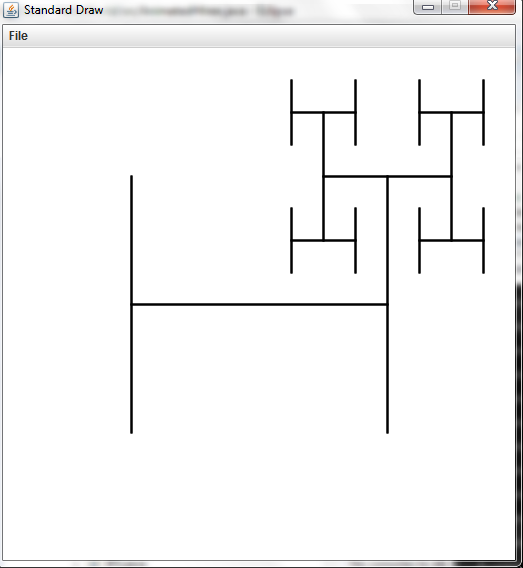
*draw*(n-1, x1, y1, size/2); // upper right

*draw*(n-1, x1, y0, size/2); // lower right

*draw*(n-1, x0, y1, size/2); // upper left

*draw*(n-1, x0, y0, size/2); // lower left

The animation start from upper right to lower left



//CASE 3

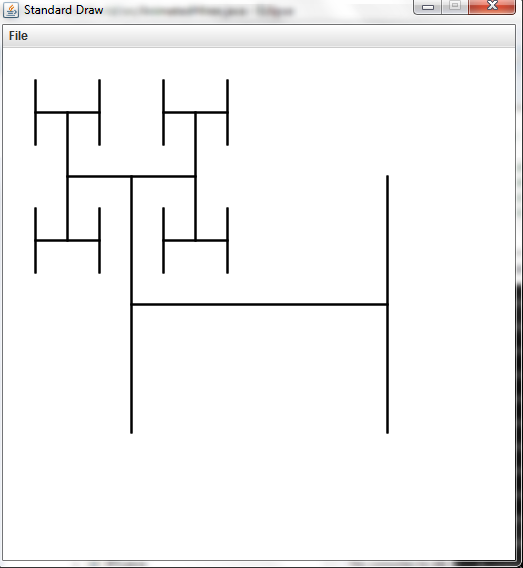
*draw*(n-1, x0, y1, size/2); // upper left

*draw*(n-1, x1, y1, size/2); // upper right

*draw*(n-1, x0, y0, size/2); // lower left

*draw*(n-1, x1, y0, size/2); // lower right

The animation start from upper left to lower right



**19. Combinations**

OUTPUT:

a

ab

abc

ac

b

bc

c

a

ab

abc

ac

b

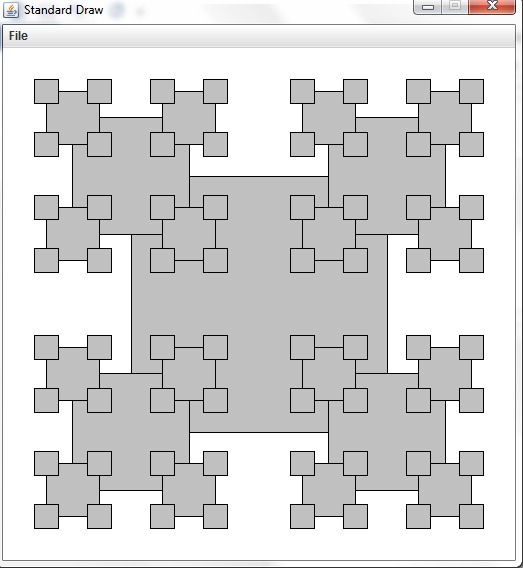
bc

c

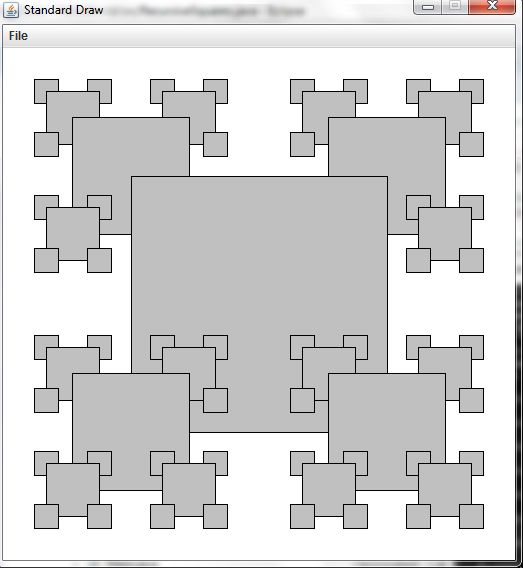
**22. Recursive Squares**

OUTPUT:

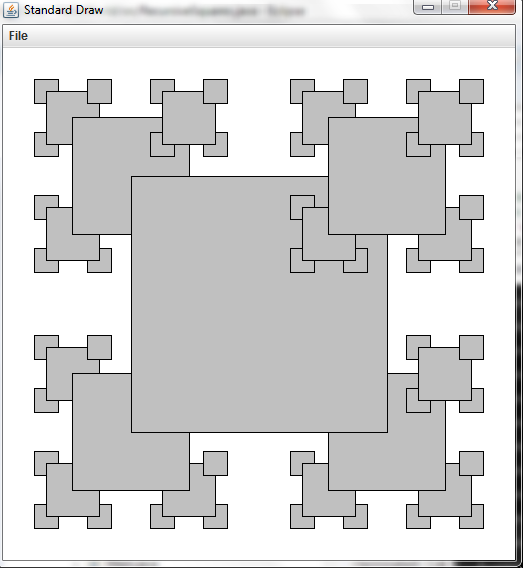
a)



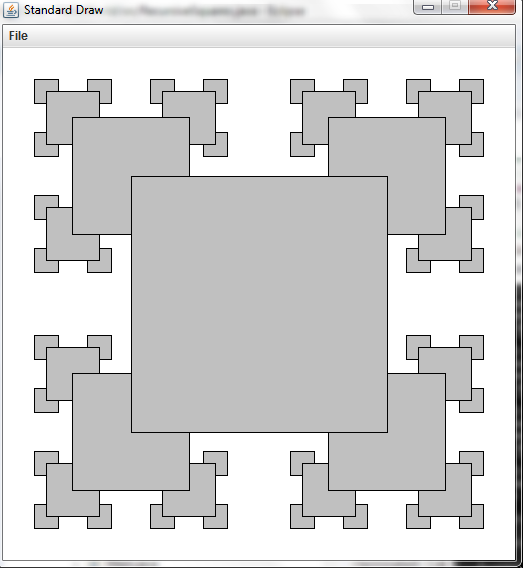
b)



c)

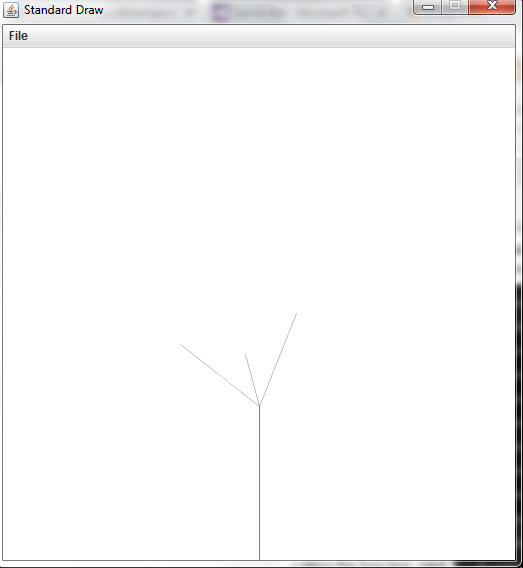
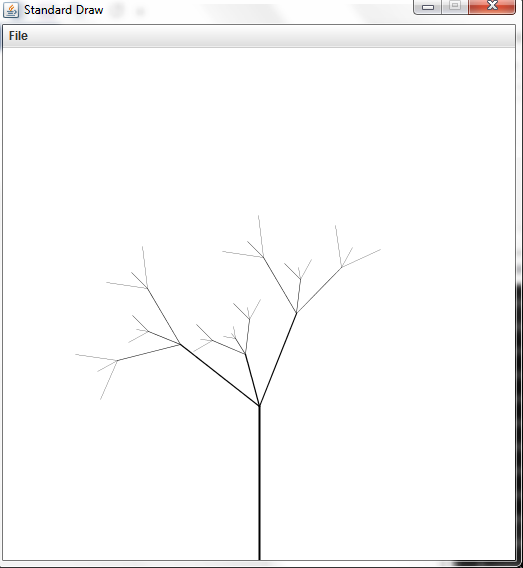
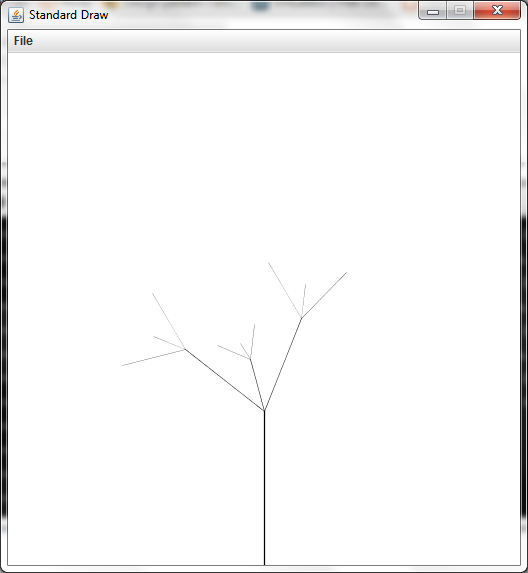


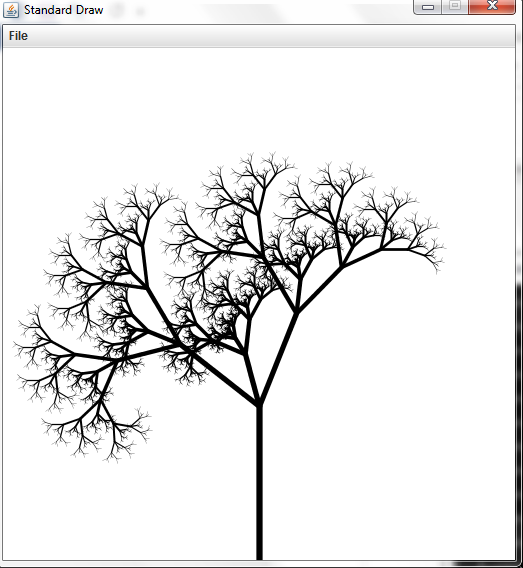
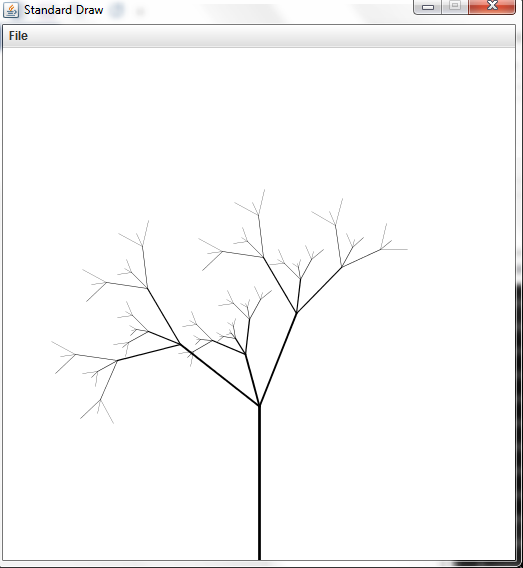
d)



**33. Recursive tree**

OUTPUT:



**WEB EXERCISES**

**2. Golden Ratio**

OUTPUT:

N=10 => 1.6181818181818182

N=20 => 1.6180339985218033

N=30 => 1.6180339887505408

**10. Fibonacci2**

With the code from Fibonacci.java, the maxim number to compute in less than 1 minute is 48:

1: 1 Total computing time: 0.0 seconds.

2: 1 Total computing time: 0.001 seconds.

3: 2 Total computing time: 0.001 seconds.

4: 3 Total computing time: 0.001 seconds.

5: 5 Total computing time: 0.002 seconds.

6: 8 Total computing time: 0.002 seconds.

7: 13 Total computing time: 0.002 seconds.

8: 21 Total computing time: 0.002 seconds.

9: 34 Total computing time: 0.002 seconds.

10: 55 Total computing time: 0.002 seconds.

11: 89 Total computing time: 0.002 seconds.

12: 144 Total computing time: 0.002 seconds.

13: 233 Total computing time: 0.002 seconds.

14: 377 Total computing time: 0.002 seconds.

15: 610 Total computing time: 0.002 seconds.

16: 987 Total computing time: 0.002 seconds.

17: 1597 Total computing time: 0.003 seconds.

18: 2584 Total computing time: 0.003 seconds.

19: 4181 Total computing time: 0.003 seconds.

20: 6765 Total computing time: 0.004 seconds.

21: 10946 Total computing time: 0.005 seconds.

22: 17711 Total computing time: 0.007 seconds.

23: 28657 Total computing time: 0.007 seconds.

24: 46368 Total computing time: 0.007 seconds.

25: 75025 Total computing time: 0.007 seconds.

26: 121393 Total computing time: 0.008 seconds.

27: 196418 Total computing time: 0.008 seconds.

28: 317811 Total computing time: 0.01 seconds.

29: 514229 Total computing time: 0.011 seconds.

30: 832040 Total computing time: 0.014 seconds.

31: 1346269 Total computing time: 0.019 seconds.

32: 2178309 Total computing time: 0.026 seconds.

33: 3524578 Total computing time: 0.039 seconds.

34: 5702887 Total computing time: 0.061 seconds.

35: 9227465 Total computing time: 0.095 seconds.

36: 14930352 Total computing time: 0.154 seconds.

37: 24157817 Total computing time: 0.243 seconds.

38: 39088169 Total computing time: 0.398 seconds.

39: 63245986 Total computing time: 0.633 seconds.

40: 102334155 Total computing time: 1.003 seconds.

41: 165580141 Total computing time: 1.621 seconds.

42: 267914296 Total computing time: 2.601 seconds.

43: 433494437 Total computing time: 4.224 seconds.

44: 701408733 Total computing time: 6.91 seconds.

45: 1134903170 Total computing time: 11.515 seconds.

46: 1836311903 Total computing time: 19.094 seconds.

47: 2971215073 Total computing time: 29.962 seconds.

48: 4807526976 Total computing time: 47.721 seconds.

49: 7778742049 Total computing time: 73.402 seconds.

With the original code Fibonacci2.java, the maximum number to compute is 93 because a validation: “Input out of bounds”. Removing this validation, I was able to compute the number 135,000 in a total in less than 1 minute:

135000: -3047139614637634528 Total computing time: 57.676 seconds.

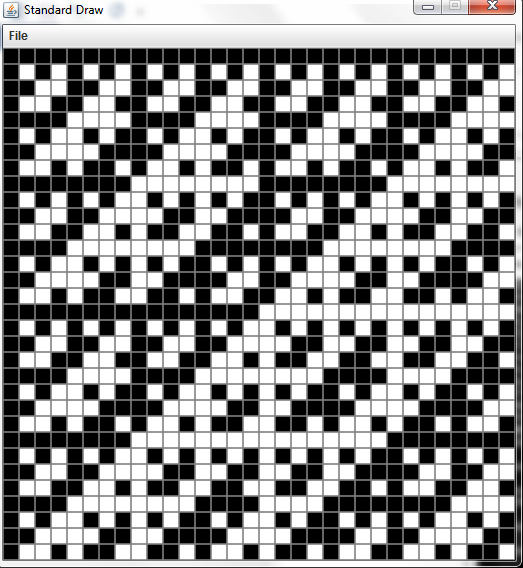
Fibonacci2 is much faster than Fibonacci

**11. Dijkstra**

DONE

**20. Hadamard matrix.**

Output using N = 5



**39. Tribonacci Numbers.**

Radio successive terms: *fib*(n-1) + *fib*(n-2) + *fib*(n-3)

OUTPUT:

1: 0 Total computing time: 0.0 secods.

2: 1 Total computing time: 0.001 secods.

3: 1 Total computing time: 0.001 secods.

4: 2 Total computing time: 0.002 secods.

5: 4 Total computing time: 0.002 secods.

6: 7 Total computing time: 0.002 secods.

7: 13 Total computing time: 0.002 secods.

8: 24 Total computing time: 0.002 secods.

9: 44 Total computing time: 0.002 secods.

10: 81 Total computing time: 0.002 secods.

11: 149 Total computing time: 0.002 secods.

**42. Maze generation.**

OUTPUT:

