Project 1 main given code

/\*

\* Project 1 Winter 17

\*/

package project1;

/\*\*

\*

\* @author lulofse

\*/

public class Project1 {

public static void main(String[] args) {

new Project1().run();

}

public void run() {

// demo JUnit test of exception handeling

DemoBigO objDivide = new DemoBigO();

objDivide.throwDivideException();

// demo throwing exception

// comment out when not using – it throws an exception and stops

/\*DemoBigO objThrow = new DemoBigO();

objThrow.throwAnException();\*/

// setup O(n) demos

long j, k, n;

long startTime, finishTime, elapsedTime = 0;

double doubleN, time;

DemoBigO obj = new DemoBigO();

// demo O(1) method

System.out.println("O(1) method calculational verification");

for (n = 1; n <= 10000000; n \*= 10) {

startTime = System.nanoTime();

obj.demoBigO1(n);

// Calculate the elapsed time:

finishTime = System.nanoTime();

elapsedTime = finishTime - startTime;

doubleN = (double) n;

time = (double) elapsedTime / 1.;

System.out.println("n= " + n + " time/1= " + time + " ");

}

System.out.print("Since time/constant is not growing with n ");

System.out.println("demoBigO1 is O(1) ");

System.out.println();

// demo O(1) method is not O( Log(n) )

System.out.println("O(1) is not O( log(n) ) method calculational verification");

for (n = 1; n <= 10000000; n \*= 10) {

startTime = System.nanoTime();

obj.demoBigO1(n);

// Calculate the elapsed time:

finishTime = System.nanoTime();

elapsedTime = finishTime - startTime;

doubleN = (double) n;

time = (double) elapsedTime / Math.log(doubleN);

System.out.println("n= " + n + " time/Log(n)= " + time + " ");

}

System.out.print("Since time/Log(n) is shrinking too rapidly with n ");

System.out.println("demoBigO1 is worstTime(n) less than O(Log(n)) ");

System.out.println();

// demo O(log n) method

System.out.println("O( Log(n) ) method calculational verification");

for (n = 1; n <= 1000000; n \*= 10) {

startTime = System.nanoTime();

obj.demoBigOLogN(n);

// Calculate the elapsed time:

finishTime = System.nanoTime();

elapsedTime = finishTime - startTime;

doubleN = (double) n;

time = (double) elapsedTime / Math.log(doubleN);

System.out.println("n= " + n + " time/Log(n)= " + time + " ");

}

System.out.print("Since time/log(n) is not growing with n ");

System.out.println("demoBigOLogN is O(log n) \n");

// demo O(n) method

System.out.println("O(n) method calculational verification");

for (n = 1; n <= 1000000; n \*= 10) {

startTime = System.nanoTime();

obj.demoBigOn(n);

// Calculate the elapsed time:

finishTime = System.nanoTime();

elapsedTime = finishTime - startTime;

doubleN = (double) n;

time = (double) elapsedTime / (doubleN);

System.out.println("n= " + n + " time/n= " + time + " ");

}

System.out.print("Since time/n is not growing with n ");

System.out.println("demoBigOLogN is O(n) \n");

// demo O(n) method is BIGGER than and not O(Log(n)

System.out.println("O(n) is BIGGER than and not O( log(n) ) method calculational verification");

for (n = 1; n <= 1000000; n \*= 10) {

startTime = System.nanoTime();

obj.demoBigOn(n);

// Calculate the elapsed time:

finishTime = System.nanoTime();

elapsedTime = finishTime - startTime;

doubleN = (double) n;

time = (double) elapsedTime / (Math.log(doubleN));

System.out.println("n= " + n + " time/Log(n)= " + time + " ");

}

System.out.print("Since time/Log(n) is growing with n ");

System.out.println("demoBigOn is worstTime(n) bigger and worse than O(Log(n)) \n");

// demo O(n) method is not O(nLog(n)

System.out.println("O(n) is not O(n log(n) ) method calculational verification");

for (n = 1; n <= 1000000; n \*= 10) {

startTime = System.nanoTime();

obj.demoBigOn(n);

// Calculate the elapsed time:

finishTime = System.nanoTime();

elapsedTime = finishTime - startTime;

doubleN = (double) n;

time = (double) elapsedTime / (doubleN \* Math.log(doubleN));

System.out.println("n= " + n + " time/nLog(n)= " + time + " ");

}

System.out.print("Since time/nLog(n) is shrinking too rapidly with n ");

System.out.println("demoBigOn is worstTime(n) less than O(Log(n)) \n");

// demo O(n\*Log(n)) method

// This is the most difficult test

System.out.println("O( nLog(n) ) method calculational verification");

for (n = 10; n <= 100000; n \*= 10) {

startTime = System.nanoTime();

// create tree

//System.out.println("create tree ");

obj.demoBigONLogNInit(n);

//System.out.println("createD tree ");

elapsedTime = finishTime - startTime;

doubleN = (double) n;

time = (double) elapsedTime / (doubleN \* Math.log(doubleN));

System.out.println("n= " + n + " time/n\*log(n)= " + time + " ");

}

System.out.println("check this range in more detail");

for (n = 100000; n <= 1300000; n += 100000) {

startTime = System.nanoTime();

// create tree

//System.out.println("create tree ");

obj.demoBigONLogNInit(n);

//System.out.println("createD tree ");

finishTime = System.nanoTime();

elapsedTime = finishTime - startTime;

doubleN = (double) n;

time = (double) elapsedTime / (doubleN \* Math.log(doubleN));

System.out.println("n= " + n + " time/n\*log(n)= " + time + " ");

}

System.out.print("Since time/n\*Log(n) is not growing with n ");

System.out.println("demoBigONLogN is O(n) \n");

// demo O(n\*n) method

System.out.println("O(n\*n) method calculational verification");

for (n = 1; n <= 100000; n \*= 10) {

startTime = System.nanoTime();

obj.demoBigOnn(n);

// Calculate the elapsed time:

finishTime = System.nanoTime();

elapsedTime = finishTime - startTime;

doubleN = (double) n;

time = (double) elapsedTime / (doubleN \* doubleN);

System.out.println("n= " + n + " time/n\*n= " + time + " ");

}

System.out.print("Since time/n\*n is not growing with n ");

System.out.println("demoBigOnn is O(n\*n) \n");

}

}