**Sistemas Concurrentes y Distribuidos**

**CC462 A**

**Laboratorio Calificado 1**

Miembros:

* Ingrid Ipanaqué Casquina
* Junior Castillo Flores

Contenido

[PROGRAMA PI 3](#_Toc43036137)

[SERIE DE LEIBNIZ 3](#_Toc43036138)

[CÓDIGO SECUENCIAL 3](#_Toc43036139)

[CalculatePi.java 3](#_Toc43036140)

[Resultados 4](#_Toc43036141)

[CÓDIGO HILOS 5](#_Toc43036142)

[ParallelPI.java 5](#_Toc43036143)

[PIThread.java 5](#_Toc43036144)

[Resultados 6](#_Toc43036145)

[PROGRAMA MÉTODO DE ORDENAMIENTO 7](#_Toc43036146)

[MERGESORT EN PARALELO 7](#_Toc43036147)

[CÓDIGO SECUENCIAL 8](#_Toc43036148)

[Resultados 8](#_Toc43036149)

[CÓDIGO CON HILOS 9](#_Toc43036150)

[Resultados 9](#_Toc43036151)

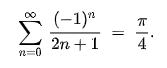
# PROGRAMA PI

## SERIE DE LEIBNIZ

En matemáticas, la fórmula de Leibniz sirve para el cálculo de π, nombradas así en honor de Gottfried Leibniz, dice que:



La expresión anterior es una [serie infinita](https://es.wikipedia.org/wiki/Serie_infinita) denominada serie de Leibniz, que [converge](https://es.wikipedia.org/wiki/Serie_convergente) a π ⁄ 4. También se la denomina serie de Gregory-Leibniz para reconocer el trabajo de [James Gregory](https://es.wikipedia.org/wiki/James_Gregory), contemporáneo de Leibniz. Usando el símbolo de [suma](https://es.wikipedia.org/wiki/Sumatoria), la serie se puede expresar como:



## CÓDIGO SECUENCIAL

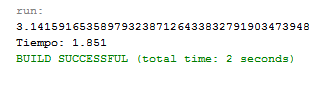
### CalculatePi.java

1. **package** calculatepi;
3. **import** java.math.BigDecimal;
4. **import** java.math.MathContext;
5. **import** java.math.RoundingMode;
7. ***/\*\****
8. ***\****
9. ***\* @author ingrid.ic***
10. ***\*/***
11. **public** **class** CalculatePi {
13. **public** **static** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) Pi(**int** inicio,**int** fin){
14. [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) pi = [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal).ZERO;
15. [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) sumPi = [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal).ZERO;
16. [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) variable = [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal).ZERO;
17. [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) numerador;
18. [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) denominador;
19. **int** i;
21. **for**(i=inicio; i<fin;i++){
22. numerador = **new** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal)(-1).pow(i);
23. denominador = **new** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal)(2).multiply( **new** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal)(i));
24. denominador = denominador.add( **new** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal)(1));
25. variable = numerador.divide(denominador, MathContext.DECIMAL128);
26. sumPi = sumPi.add(variable);
27. }
28. pi = sumPi.multiply(**new** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal)(4));
29. **return** pi;
30. }
32. **public** **static** **void** main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args) {
33. [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) pi;
34. **int** fin = 1000000;
35. **int** inicio = 0;
37. **long** inicioTime = [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).currentTimeMillis();
39. pi = Pi(inicio,fin);
41. **long** finTime = [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).currentTimeMillis();
43. **double** tiempo = (**double**) (finTime - inicioTime);
44. [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(pi);
45. [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.printf("Tiempo: %s\n",tiempo / 1000.0);
46. }
47. }

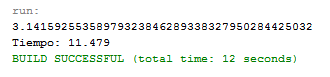
### 

### Resultados

Con n = 1,000,000



Con n = 10,000,000



## CÓDIGO HILOS

### ParallelPI.java

1. **package** threadpi;
3. **import** java.math.BigDecimal;
5. **public** **class** ParallelPI {
6. **private** **static** **final** **short** CPU\_CORES = 8;
7. **private** **static** **final** **long** FINAL\_LIMIT = 100000;
9. **public** **static** **void** main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args)  **throws** [InterruptedException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+interruptedexception){
10. PIThread threads[] = **new** PIThread[CPU\_CORES];
11. [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) sum = [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal).ZERO;
12. [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) pi = [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal).ZERO;
14. **final** **long** startTime = [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).currentTimeMillis();
15. **for**(**int** i=0; i<CPU\_CORES;i++){
16. **long** start = (**long**) (i\*FINAL\_LIMIT/CPU\_CORES);
17. **long** end = (**long**)((i+1)\*FINAL\_LIMIT/CPU\_CORES);
19. threads[i] = **new** PIThread(start,end);
20. threads[i].start();
21. }
23. **for**(**int** i=0; i<CPU\_CORES; i++){
24. threads[i].join();
25. sum = sum.add(threads[i].getSum());
26. }
28. pi = sum.multiply(**new** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal)(4));
30. **final** **long** endTime = [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).currentTimeMillis();
31. [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(pi);
32. [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.printf("Tiempo: %s\n",(endTime - startTime) / 1000.0);
33. }
34. }

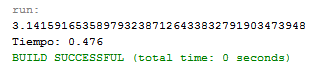
### PIThread.java

1. **package** threadpi;
3. **import** java.math.BigDecimal;
4. **import** java.math.MathContext;
6. **public** **class** PIThread **extends** [Thread](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+thread){
7. **private** **final** **long** start;
8. **private** **final** **long** end;
9. **private** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) sum;
11. **public** PIThread(**long** start,**long** end){
12. **this**.start = start;
13. **this**.end = end;
14. **this**.sum = [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal).ZERO;
15. }
17. @Override
18. **public** **void** run(){
19. [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) variable = [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal).ZERO;
20. [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) numerador;
21. [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) denominador;
23. **for** (**long** i=start;i<end;i++){
24. numerador = **new** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal)(-1).pow((**int**) i);
25. denominador = **new** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal)(2).multiply( **new** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal)(i));
26. denominador = denominador.add( **new** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal)(1));
27. variable = numerador.divide(denominador, MathContext.DECIMAL128);
28. sum = sum.add(variable);
29. //System.out.println(sum);
30. }
31. }
33. **public** [BigDecimal](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bigdecimal) getSum(){
34. **return** sum;
35. }
36. }

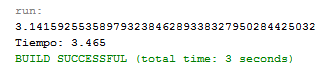
## 

## Resultados

Con n = 1,000,000



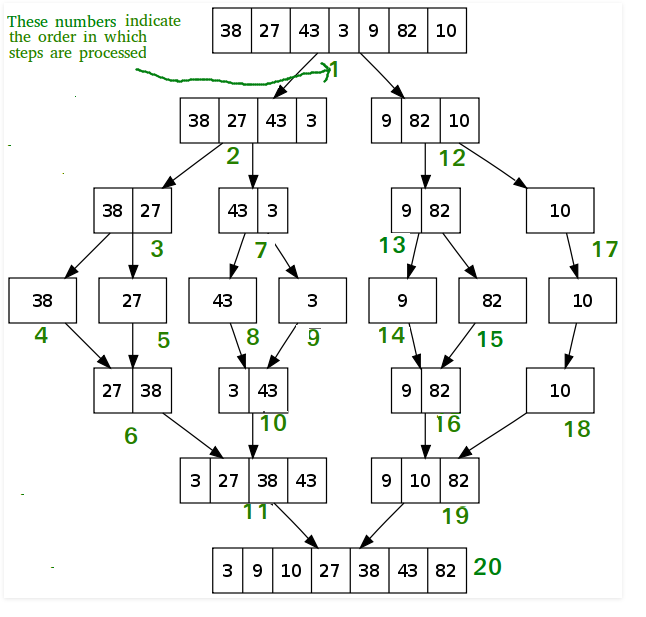
Con n = 10,000,000



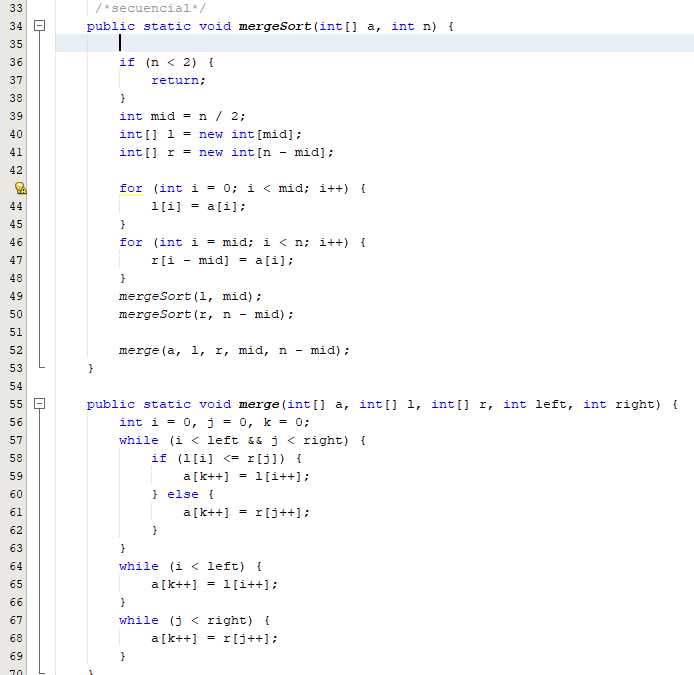
# PROGRAMA MÉTODO DE ORDENAMIENTO

## MERGESORT

En el curso de algoritmos, vimos que este tipo de ordenamiento se basaba en la técnica divide y vencerás, este método consiste en ir dividiendo la [secuencia](https://www.ecured.cu/index.php?title=Secuencia&action=edit&redlink=1) de datos en dos subsecuencias hasta que las subsecuencias tengan un único [elemento](https://www.ecured.cu/index.php?title=Elemento&action=edit&redlink=1), luego se ordenan mezclando dos subsecuencias ordenadas en una [secuencia ordenada](https://www.ecured.cu/index.php?title=Secuencia_ordenada&action=edit&redlink=1), en forma sucesiva hasta obtener una [secuencia](https://www.ecured.cu/index.php?title=Secuencia&action=edit&redlink=1) única ya ordenada. Si n = 1 solo hay un [elemento](https://www.ecured.cu/index.php?title=Elemento&action=edit&redlink=1) por ordenar, sino se hace una [ordenación](https://www.ecured.cu/index.php?title=Ordenaci%C3%B3n&action=edit&redlink=1) de [mezcla](https://www.ecured.cu/index.php?title=Mezcla&action=edit&redlink=1) de la primera mitad del [arreglo](https://www.ecured.cu/index.php?title=Arreglo&action=edit&redlink=1) con la segunda mitad. Las dos mitades se ordenan de igual forma.

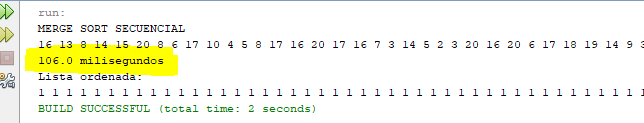


## CÓDIGO SECUENCIAL

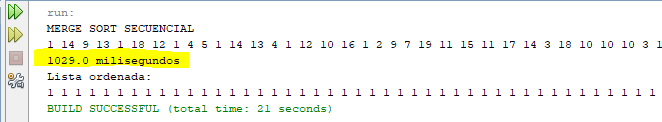


## Resultados

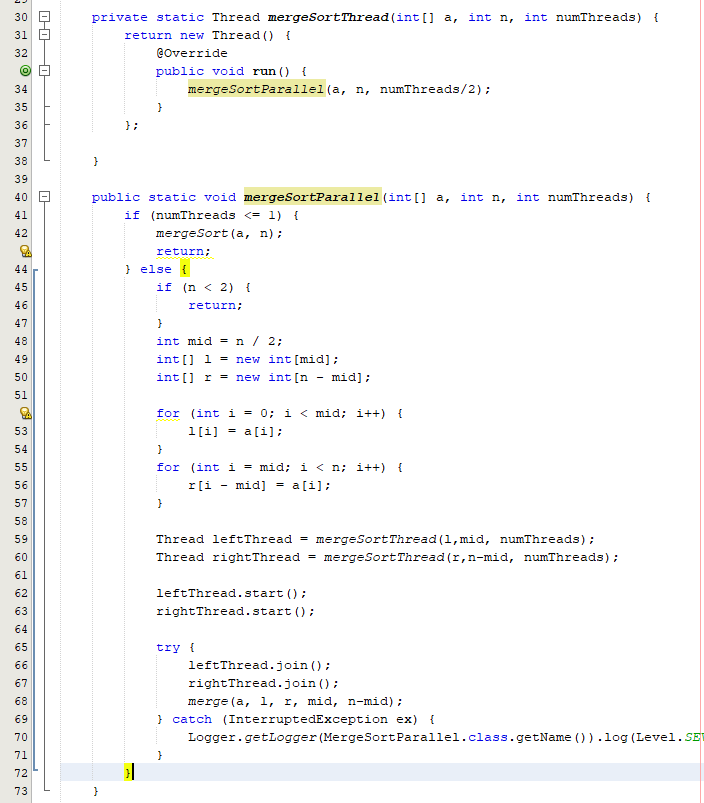
Con n = 1 000 000



Con n = 100 000 000



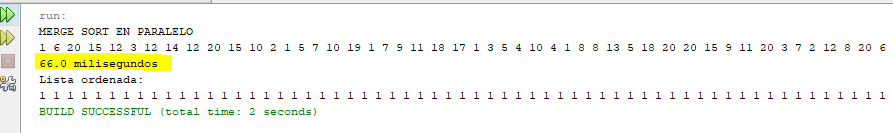
## CÓDIGO CON HILOS



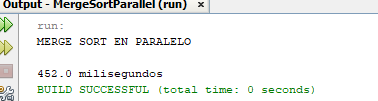
## 

## Resultados

Con n = 1 000 000 y 8 hilos



Con n = 10 000 000 y 8 hilos

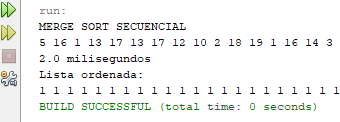


Observacion:

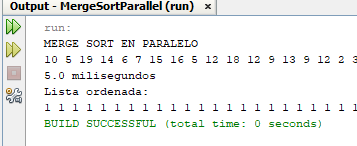
Cuando tenemos arreglos de orden menor a 100 000 el algoritmo secuencial, es incluso más rápido que el que utiliza hilos.

Por ejemplo para n = 10 000

Secuencial:

****

Paralelo:

****

Se puede observar una mejora en el tiempo, al usar hilos para dividir el trabajo de dividir el arreglo, esto no se observa por debajo del orden n = 100 000.