INFORME TI2

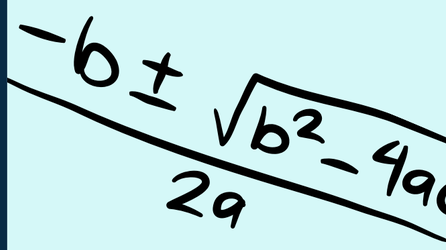
DAVIDE FLAMINI

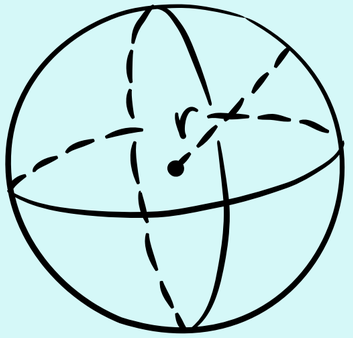
NICOLAS CUELLAR

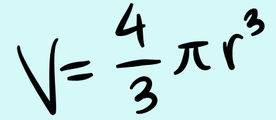
ANDRES CABEZAS

CHECKSUM

Una suma de verificación o checksum , también conocida como suma de chequeo, es una función matemática de redundancia cuyo principal objetivo es el de detectar cambios malintencionados o accidentales en una transmisión de datos con el fin de proteger la integridad de la información.

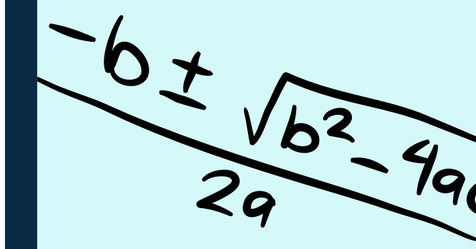




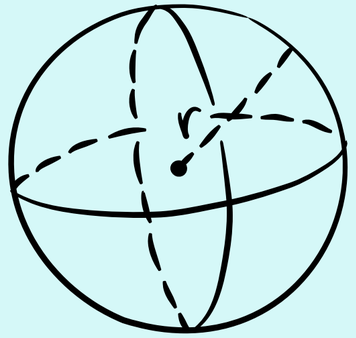


Modular Single-Sum Checksum:

En términos sencillos, estos algoritmos toman los datos, los dividen en bloques y luego suman estos bloques de una manera específica (modular).



$y = mx + b$



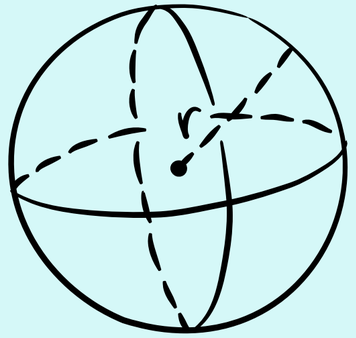
$V = \frac{4}{3}\pi r^3$

Modular Dual-Sum Checksum:

En un algoritmo de Dual-Sum, cada suma modular es la mitad del tamaño del valor de comprobación, y las dos sumas modulares se concatenan para formar un solo valor de comprobación.

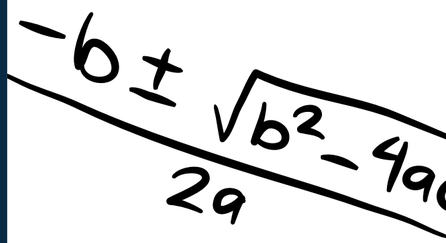
$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$y = mx + b$$

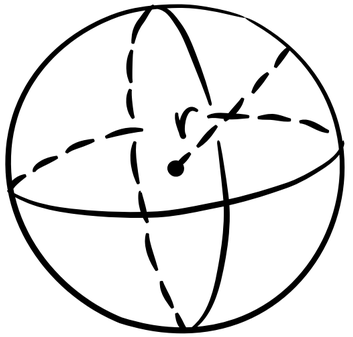


$$V = \frac{4}{3}\pi r^3$$

IMPLEMENTACION CHECKSUM

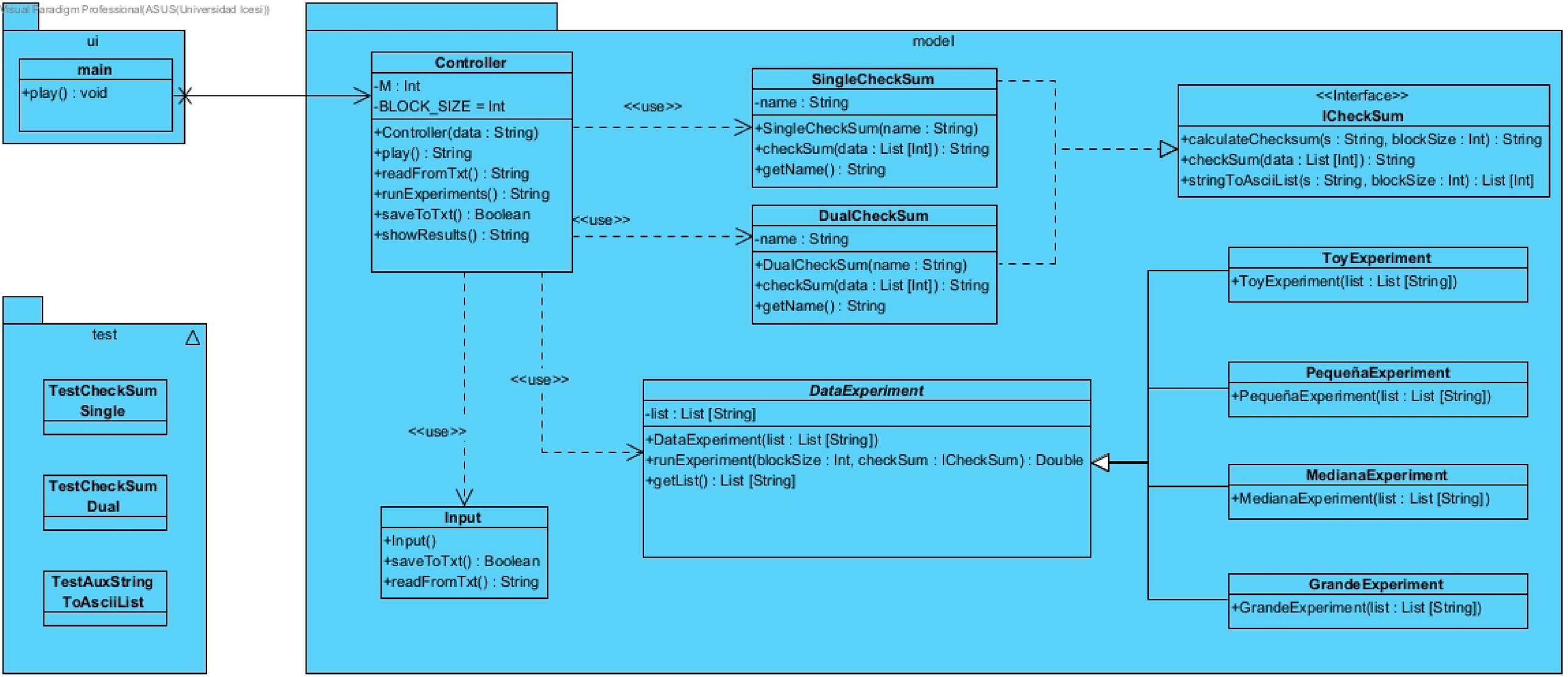


$y = mx + b$



$V = \frac{4}{3}\pi r^3$

Diagrama de clases



```scala
trait ICheckSum {

  /**
   * Calcula el checksum para una lista de bloques de datos.
   *
   * @param data Lista de bloques de datos representados como enteros.
   * @return Resultado del checksum en formato de cadena.
   */
  def checkSum(data: List[Int]): String

  /**
   * Calcula el checksum para una cadena de datos dividida en bloques.
   *
   * @param s Cadena de datos.
   * @param blockSize Tamaño de los bloques para el cálculo del checksum.
   * @return Resultado del checksum en formato de cadena.
   */
  def calculateChecksum(s: String, blockSize: Int): String =
    checkSum(stringToAsciiList(s, blockSize))

  /**
   * Convierte una cadena de texto en una lista de valores ASCII, divididos en bloques.
   *
   * @param s Cadena de texto.
   * @param blockSize Tamaño de los bloques para la conversión.
   * @return Lista de valores ASCII representando los bloques de datos.
   */
  def stringToAsciiList(s: String, blockSize: Int): List[Int] = {
    s.grouped(blockSize).flatMap(block => block.map(_.toInt)).toList
  }
}
```

INTERFACE

```scala
package model

/** Clase que implementa la interfaz ICheckSum para calcular el checkSum de un solo bloque de datos.
 *
 *  @param name Nombre del bloque de datos.
 *  @param M Valor M utilizado en el cálculo de la suma de verificación.
 */
class SingleCheckSum(name: String, M: Int) extends ICheckSum {

    /** Calcula la suma de verificación para un solo bloque de datos.
     *
     *  @param data Lista de enteros que representa el bloque de datos.
     *  @return La suma de verificación como una cadena de texto.
     */
    override def checkSum(data: List[Int]): String =
        data.foldLeft(0)((sum, block) => (sum + block) % M).toString
}
```

SINGLE-SUM

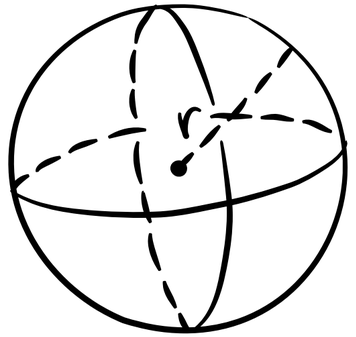
```scala
  /**
   * Calcula el checksum dual para una lista de bloques de datos.
   *
   * @param data Lista de bloques de datos representados como enteros.
   * @return Resultado del checksum dual en formato de cadena.
   */
  override def checkSum(data: List[Int]): String =
    val (sumA, sumB) = data.foldLeft((0, 0)) {
      case ((sumA, sumB), block) =>
        val newSumA = (sumA + block) % M
        val newSumB = (sumB + newSumA) % M
        (newSumA, newSumB)
    }
    sumA.toString + sumB.toString
}
```

DUAL-SUM

EXPERIMENTACION

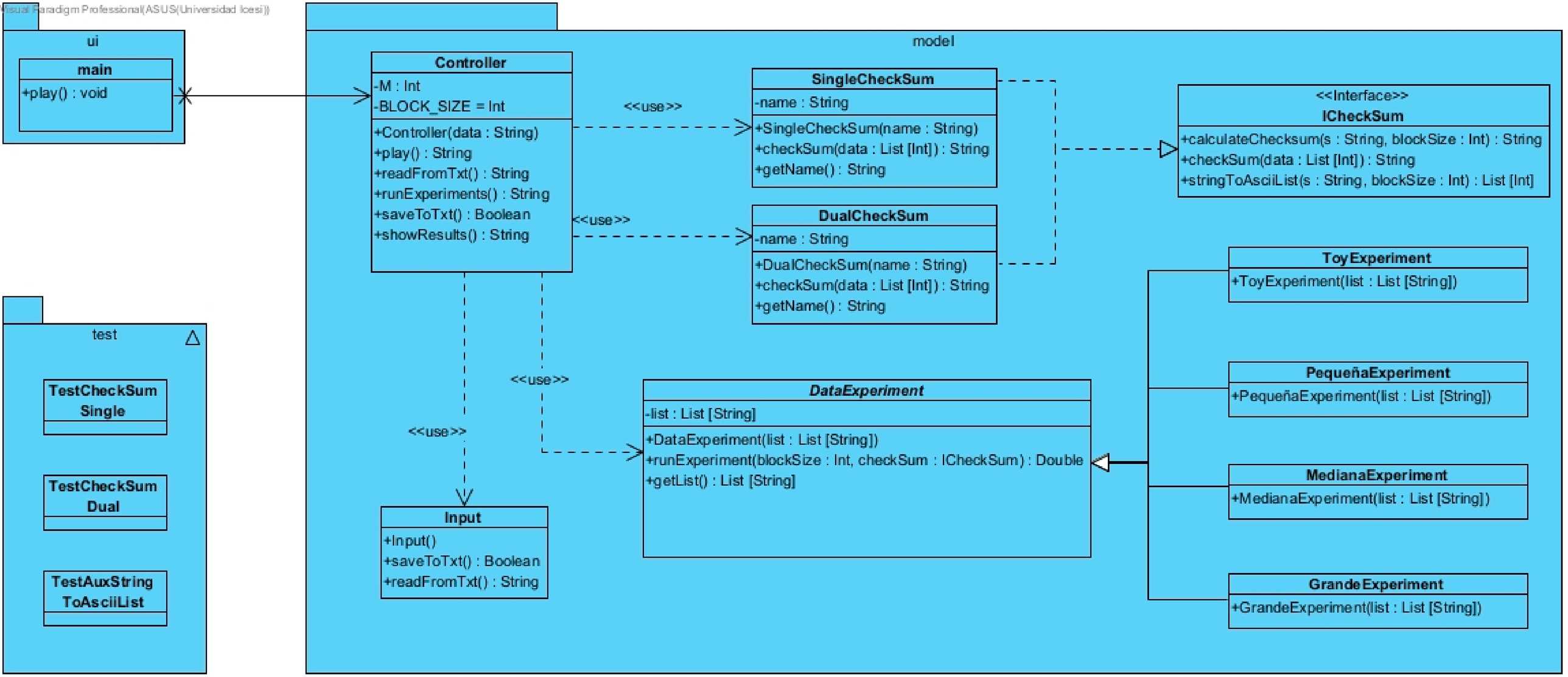
$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$y = mx + b$$



$$V = \frac{4}{3}\pi r^3$$

Diagrama de clases



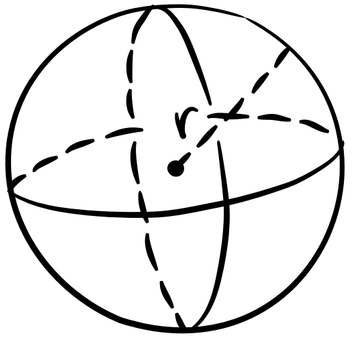
COMPLEJIDAD TEORICA SINGLE-SUM

El método **foldLeft** recorre cada elemento de la lista una vez, aplicando la función proporcionada en el argumento a medida que avanza. En este caso, la función suma el bloque actual al acumulador y luego toma el módulo **M**. La operación **foldLeft** se realiza en un solo pase sobre la lista, por lo que la complejidad es O(n), donde n es el número de elementos en la lista **data**.

Entonces, la complejidad del algoritmo es O(n), donde n es el tamaño de la lista de entrada **data**.

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$y = mx + b$$



$$V = \frac{4}{3}\pi r^3$$

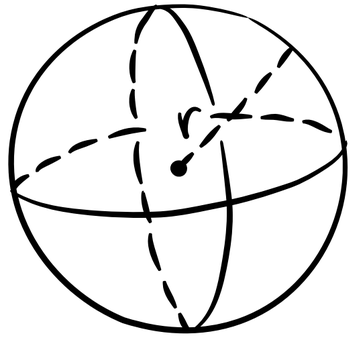
COMPLEJIDAD TEORICA DUAL-SUM

Es en función del tamaño de la lista **data**.
La función **foldLeft** recorre la lista una vez y realiza operaciones constantes en cada elemento. En este caso, realiza dos sumas y dos módulos en cada iteración, pero estas operaciones son constantes en términos de la longitud de la lista.

Por lo tanto, la complejidad de este algoritmo también es O(n), donde n es el número de elementos en la lista **data**.

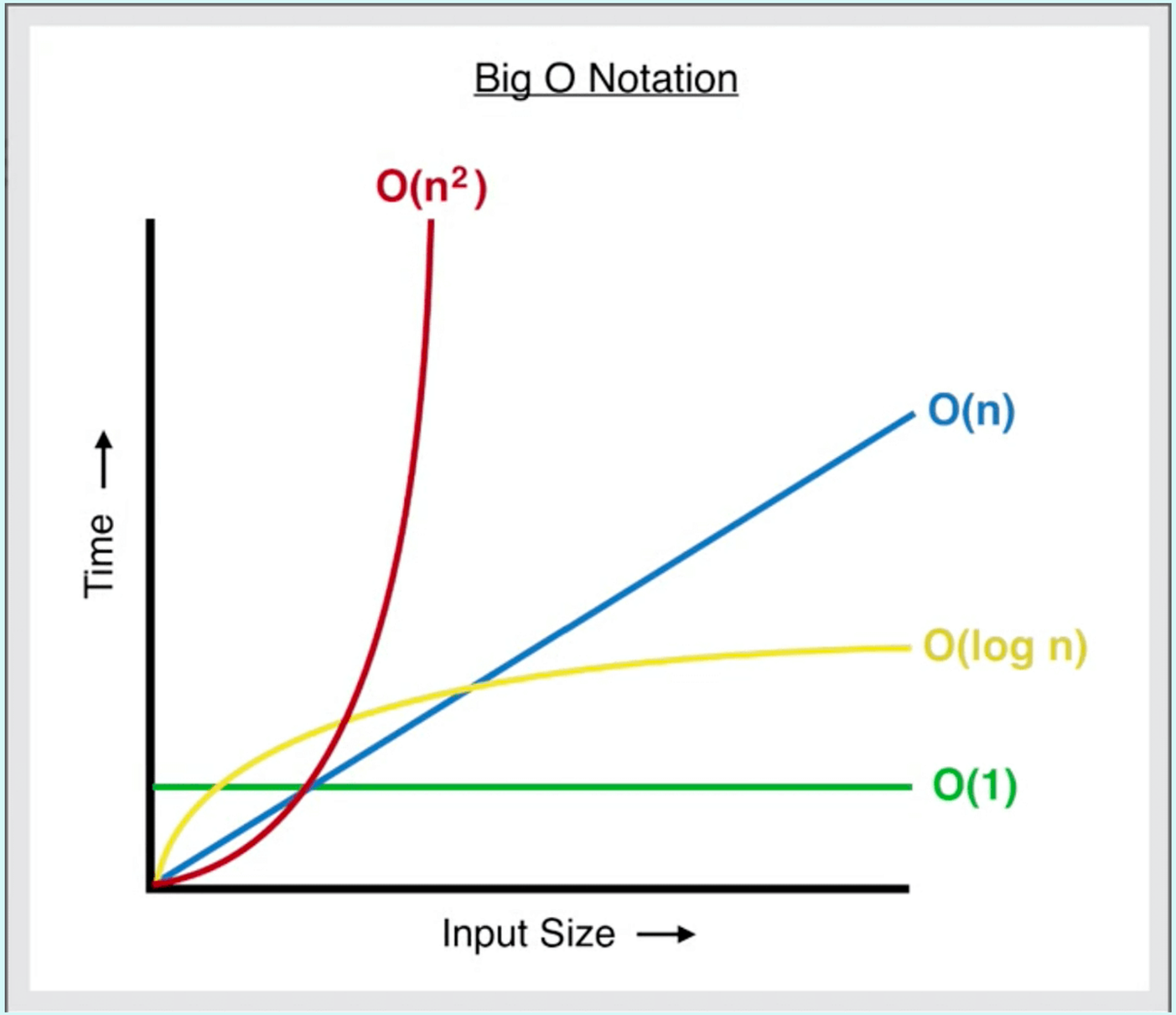
$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$y = mx + b$$



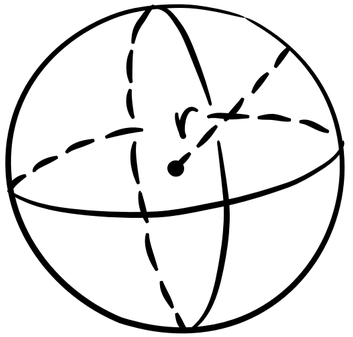
$$V = \frac{4}{3}\pi r^3$$

GRAFICA COMPLEJIDADES



$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$y = mx + b$$



$$V = \frac{4}{3}\pi r^3$$

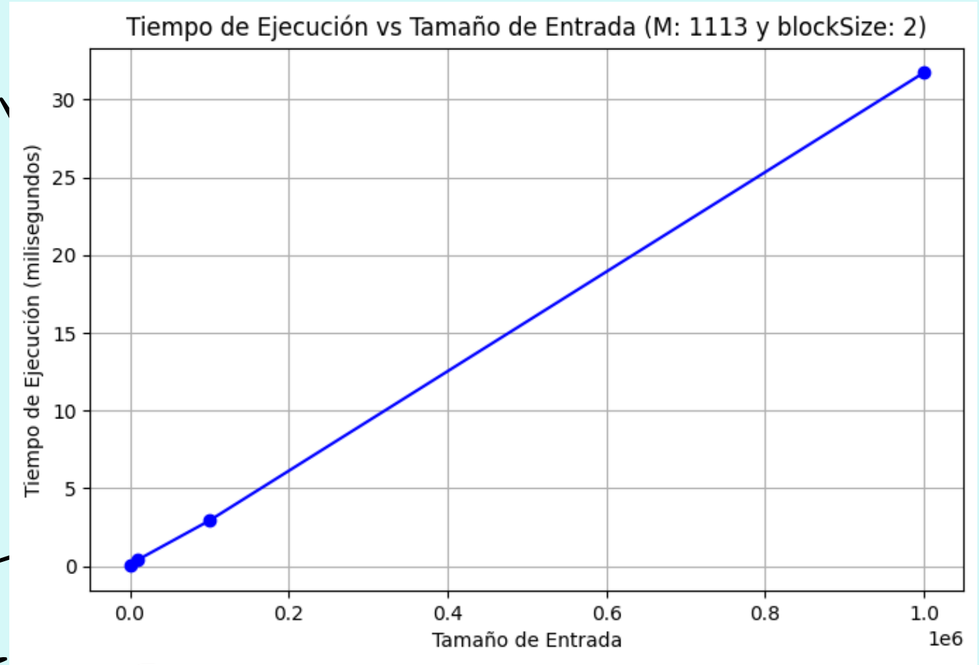
Google Colaboratory

google.com

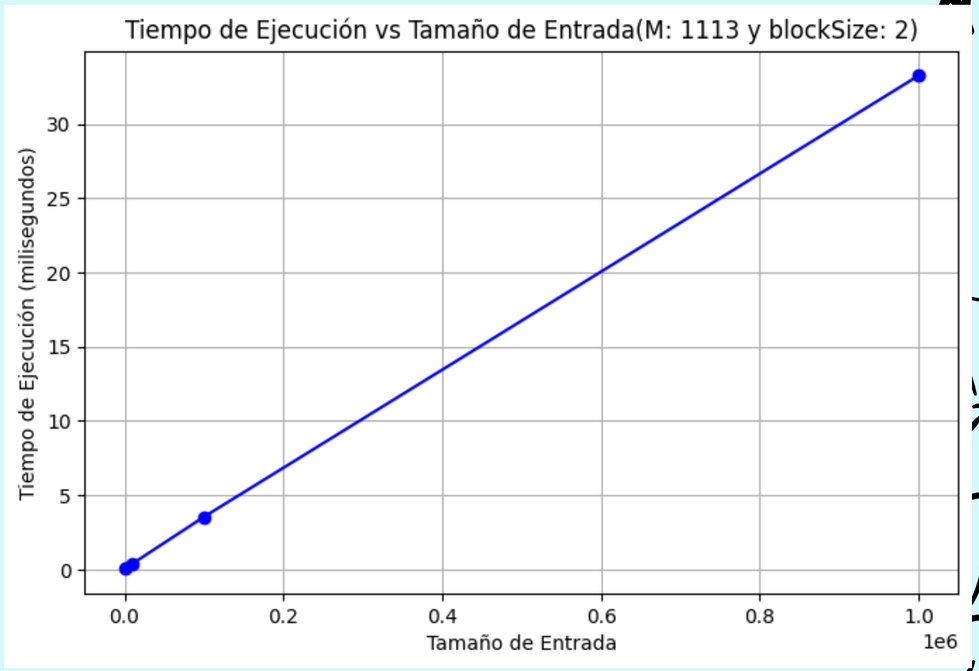
Primer Informe

M:1113 blockSize: 2

$Y = 31.75\ X + (-0.03)$



$Y = 33.15\ X + (0.11)$



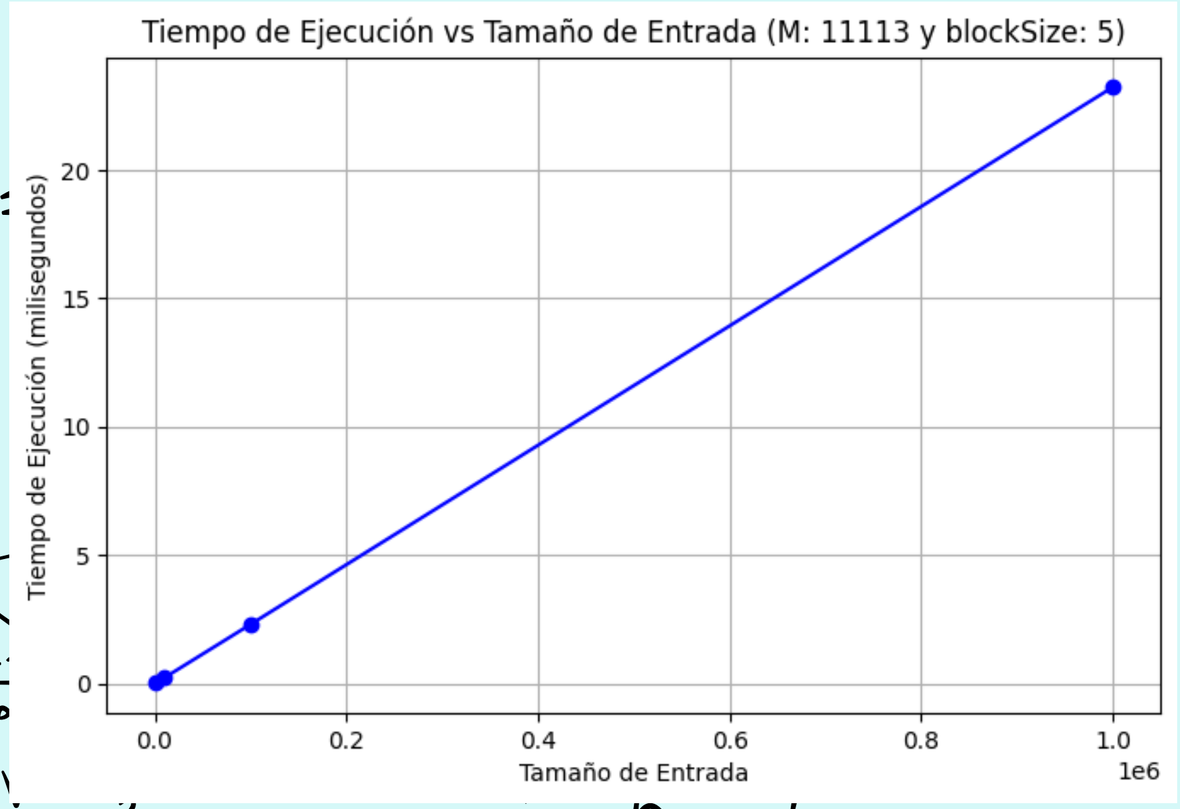
Google Colaboratory

google.com

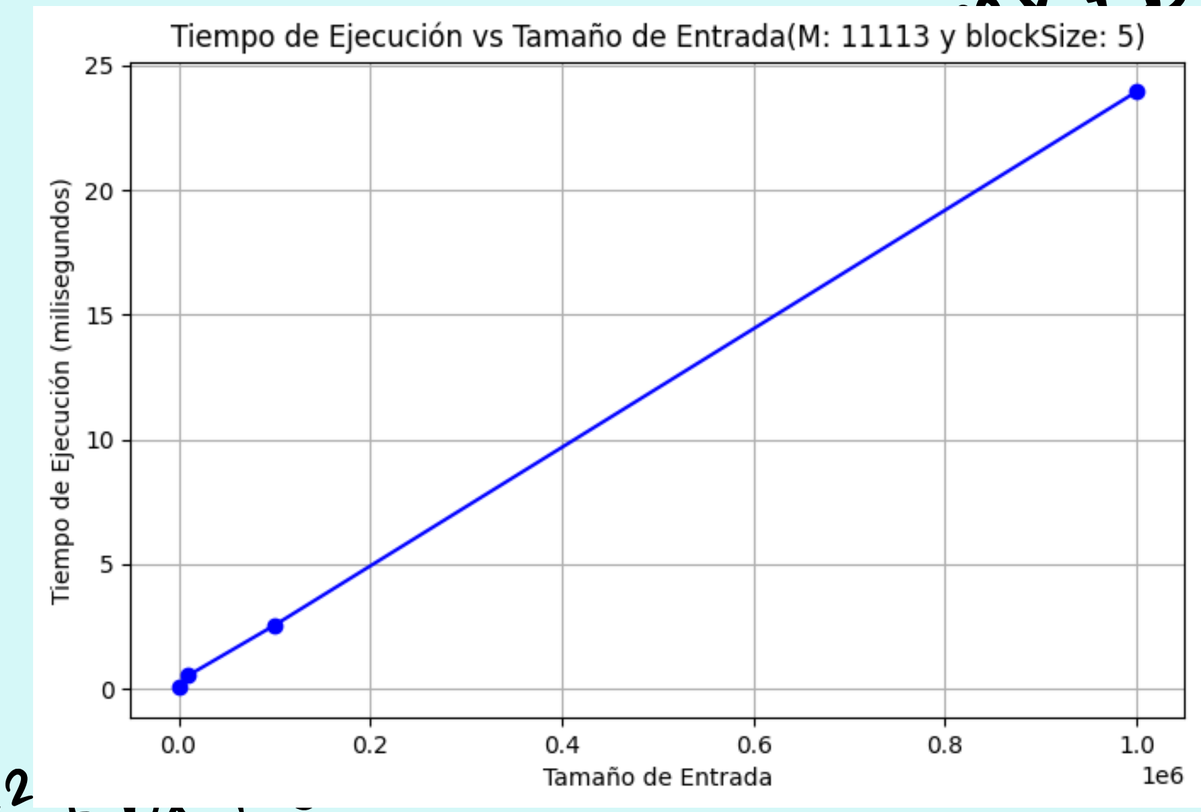
Segundo Informe

M:11113 blockSize: 5

$Y = 23.21\,X + (0.01)$



$Y = 23.78\,X + (0.17)$



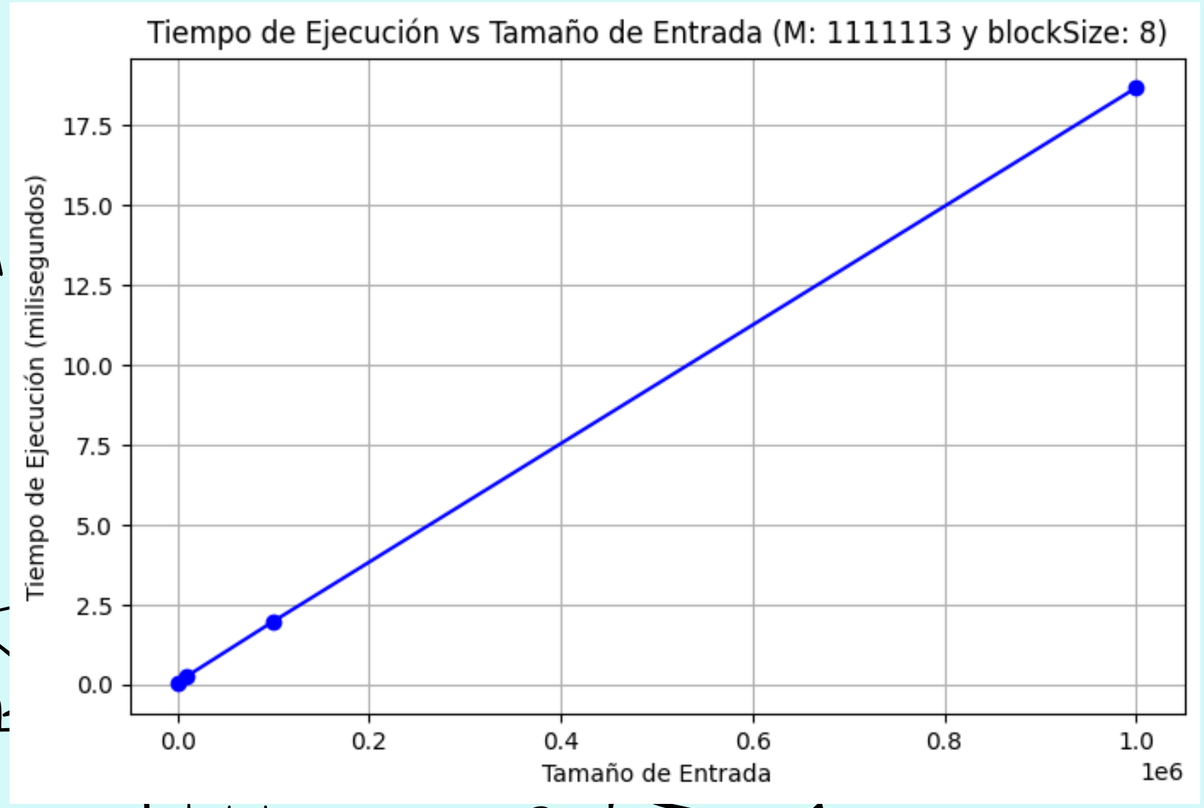
Google Colaboratory

google.com

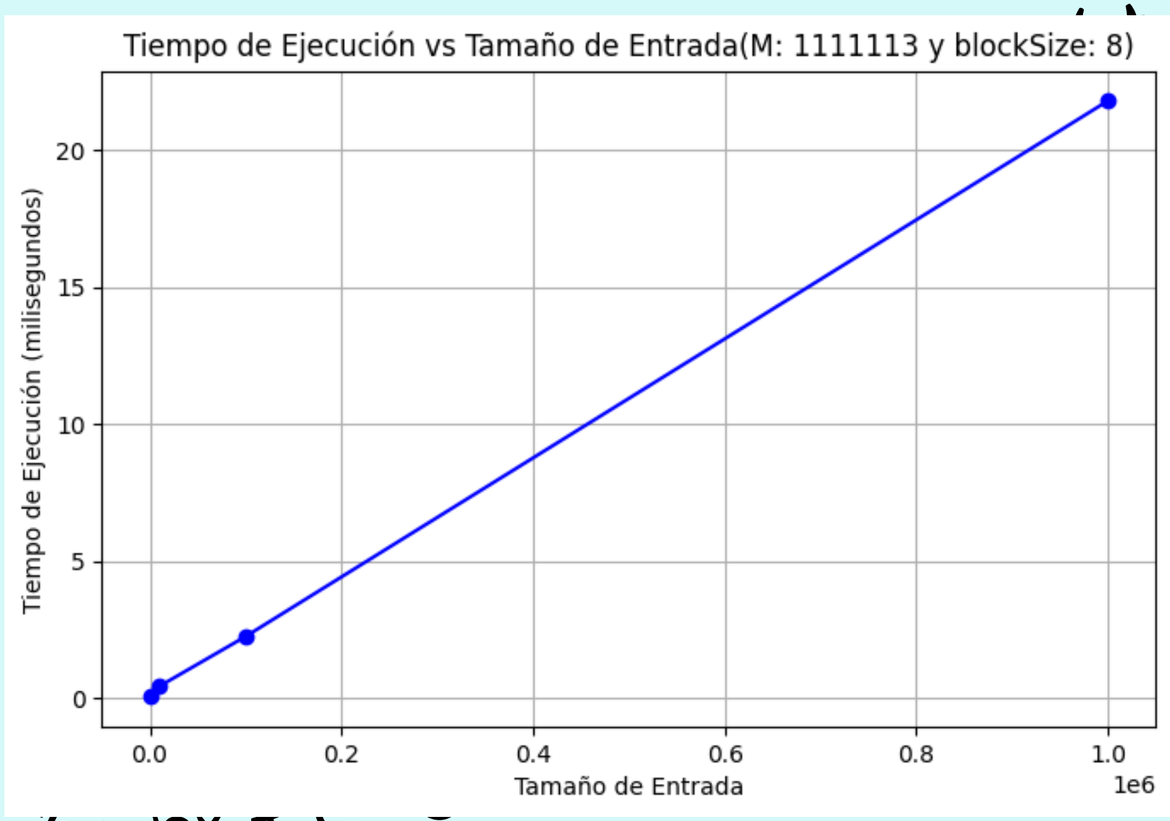
Tercer Informe

M:1111113 blockSize: 8

$Y = 18.61\,X + (0.07)$



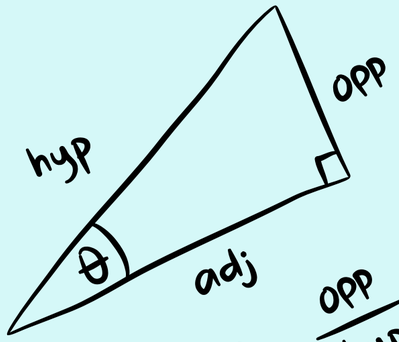
$Y = 21.68\,X + (0.13)$



CONCLUSION

- El checkSum es de suma importancia.
- Se pudo evidenciar a través de las gráficas que la complejidad teórica y real son similares.
- Al aumentar M y blockSize, se registra menos tiempo de ejecución

¡GRACIAS!



$\sin(\theta) = \frac{opp}{hyp}$

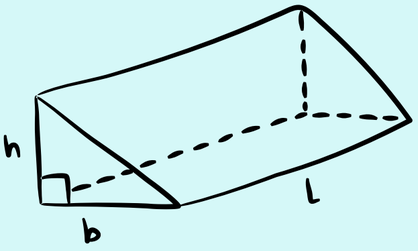
$V = lwh$

$V = \pi r^2 h$

$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

$a = \frac{V_f - V_i}{t}$

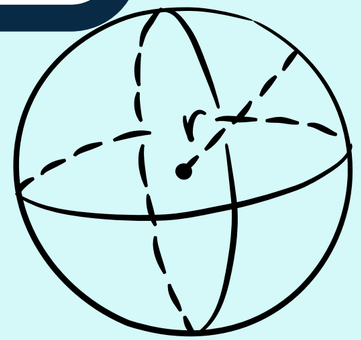
$y = mx + b$



$V = \frac{1}{2}bhl$

$\frac{x}{a} + \frac{y}{b} = 1$

$ax^2 + bx + c = 0$



$V = \frac{4}{3}\pi r^3$