# Método de la Ingeniería

**Aplicación para la Solución de un Problema**

**Miguel Angel Romero Rosas y Jonathan Arenas**

# Contexto Problemático

Una empresa de fabricación de microprocesadores está evaluando la posibilidad de implementar varios algoritmos de ordenamiento como instrucciones básicas de su próximo coprocesador matemático. la empresa ha decidido implementar tres (3) algoritmos diferentes de ordenamiento que permitan ordenar, muy rápidamente, números enteros de tamaño arbitrariamente grande y números en formato de coma flotante de cualquier tamaño.

# Desarrollo de la Solución

# Paso 1. Identificación del Problema

## Identificación de necesidades y síntomas

1. La empresa requieren implementar varios algoritmos de ordenamiento como instrucciones básicas de su próximo coprocesador matemático.
2. La solucion al problema debe garantizar un ordenamiento muy rápidamente de números enteros de tamaño arbitrariamente grande y números en formato de coma flotante de cualquier tamaño.
3. -La solucion del problema debe de ser eficiente ya que esto evitaria que el procesador principal tenga que realizar estas tareas de cómputo intensivo.
4. -El coprocesador matematico que se quiere implementar debe acelerar el rendimiento del Sistema por el hecho de esta descarga de trabajo en el procesador principal.
5. -Una empresa de fabricación de microprocesadores desea implementar tres algoritmos de ordenamientos como una operación nativa de su próximo coprocesador.
6. -Sus anteriores lineas de coprocesadores no implementaban o no soportaban varios algoritmos de ordenamiento.
7. -los algoritmos deben permitir ordenar muy rapidamente.
8. -La empresa desea un software para probar los algoritmos qué serán implementados en el hardware.
9. -No disponen de un software que permita evaluar la eficiencia y eficacia de los algoritmos en el hardware
10. -La empresa desea poder ingresar los valores que desean ordenar.
11. -La empresa desea que el programa genere aleatoriamente los valores.
12. -La empresa desea tener control sobre la cantidad total de valores generados y el intervalo en el cuál se generarán los mismos.
13. -La empresa, además, requiere poder elegir entre números repetidos o ausencia de los mismos en los números generados.
14. -La generación aleatoria debe permitir elegir entre unas configuraciones de los valores en la secuencia.
15. -La empresa requiere que el software ordene los valores ingresados utilizando el algoritmo apropiado deacuerdo con el tipo de número a ordenar y el intervalo de números generados.
16. -Finalmente, La empresa requiere que la interfaz gráfica muestre el tiempo que se demoró el algoritmo en ordenar las entradas.

## Definición del Problema

La empresa require la implementacion de varios algoritmos de ordenamiento como instrucciones básicas de su próximo coprocesador matemático, que permitan ordenar rápidamente, números enteros de tamaño arbitrariamente grande y números en formato de coma flotante de cualquier tamaño.

## **Paso 2 . Requerimientos Funcionales**

1. -La interfaz gráfica debe ofrecer una opción mediante la cuál el usuario pueda ingresar los valores que se desean ordenar.
2. -La interfaz gráfica debe ofrecer una opción mediante la cuál el usuario pueda configurar el tipo de dato a generar.
3. -La interfaz gráfica debe ofrecer una opción mediante la cuál el usuario pueda configurar la cantidad total de números a generar.
4. -La interfaz gráfica debe ofrecer una opción mediante la cuál el usuario pueda configurar el intervalo en el cuál se generarán los números.
5. -La interfaz gráfica debe ofrecer una opción mediante la cuál el usuario pueda decidir la ausencia o existencia de números repetidos.
6. -La interfaz gráfica debe ofrecer una opción mediante la cuál el usuario pueda elegir entre las siguientes configuraciones de los valores en la secuencia: (a)valores ya ordenados (b)valores ordenados inversamente (c)valores en orden completamente aleatorio (d)valores desordenados en un % dado por el usuario.
7. -La interfaz gráfica debe ofrecer una opción mediante la cuál el usuario pueda generar aleatoriamente valores.
8. -la interfaz gráfica debe dar al modelo del mundo los valores que se desean generar junto con la configuración deseada para la serie de datos de salida.
9. -El modelo del mundo debe generar la serie de datos con las especificaciones brindadas por la interfaz del usuario.
10. -el modelo del mundo debe brindar a la interfaz del usuario la serie de datos generados.
11. -La interfaz gráfica debe ofrecer una opción mediante la cuál el usuario pueda ordenar los valores.
12. -La interfaz gráfica debe dar al modelo del mundo los valores que se desean ordenar.
13. -El modelo del mundo debe brindar al programa 3 algoritmos de ordenamiento para entradas grandes enteras y de coma flotante.
14. -El modelo del mundo debe elegir un algoritmo mediante el cuál, dependiendo de la entrada, se llevará acabo la operación de ordenar.
15. -El modelo del mundo debe brindar a la interfaz gráfica los valores de la secuencia.
16. -El modelo del mundo debe brindar a la interfaz gráfica el tiempo empleado en el proceso de ordenamiento.
17. -La interfaz gráfica debe mostrar al usuario la serie de datos ordenados.
18. -La interfaz gráfica debe mostrar al usuario el tiempo empleado en el proceso de ordenar.

## 

# Paso 3. Recopilación de Información

## Definiciones

**Numeros en formato coma flotante:**

es una forma de notación científica usada en los microprocesadores con la cual se pueden representar números racionales extremadamente grandes y pequeños de una manera muy eficiente y compacta, y con la que se pueden realizar operaciones aritméticas.

**Cooprocesador:** es un microprocesador de un ordenador utilizado como suplemento de las funciones del procesador principal (la CPU). Las operaciones ejecutadas por uno de estos coprocesadores pueden ser operaciones de aritmética en coma flotante, procesamiento gráfico, procesamiento de señales, procesado de texto, criptografía, etc.

# Operaciones con aritmetica:

son: suma, resta, multiplicación, división, potenciación, división entera. Los operadores lógicos son: "y", "o", "no" y "o exclusivo".

**Numero entero:**

es un elemento del conjunto numérico que contiene los números naturales N ={1,2,3,4….}, sus opuestos y el cero.1​ Los enteros negativos, como −1 o −3 (se leen «menos uno», «menos tres», etc.), son menores que cero y todos los enteros positivos.

# Paso 3. Búsqueda de Soluciones Creativas

**Alternativa 1:Metodo Merge Sort:**

El algoritmo de ordenación por combinación o Merge Sort, basado en la técnica Divide y Vencerás, ordena recursivamente un conjunto de elementos dividiéndolo en dos, ordenando cada una de estas partes en forma independiente y combinando los dos resultados.

Solucionar el problema por medio del operador %, el cual es de uso exclusivo entre enteros. 7%3 devuelve 1 ya que el resto de dividir 7 entre 3 es 1. Al valor obtenido lo denominamos módulo (en otros lenguajes en vez del símbolo % se usa la palabra clave mod) y a este operador a veces se le denomina “operador módulo”.

function esEntero(numero){

if (numero % 1 == 0) {

alert ("Es un numero entero");

} else {

alert ("Es un numero decimal");

}

}

# Alternativa 2:

Otra opción es usar Math.floor() para obtener la parte entera del numero y restársela al numero y si es 0 pues entonces el número es entero y sino pues obviamente no.

function esEntero(numero){

if (numero - Math.floor(numero) == 0) {

alert ("Es un numero entero");

} else {

alert ("Es un numero decimal");

}

}

**Alternativa 3:**

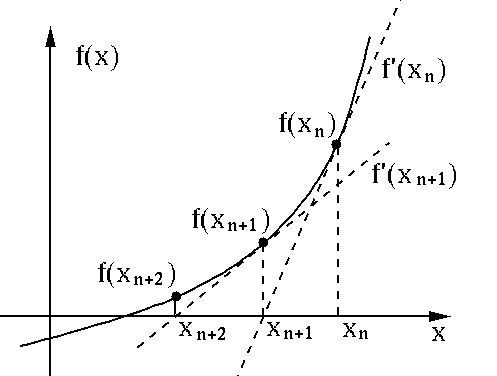
**Paso 4. Transición de las Ideas a los Diseños Preliminares** :

Fuente:

# https://es.wikipedia.org

Para este paso, aunque podemos pensar en soluciones propias, y como resulta que el problema es uno clásico en las matemáticas, buscamos en textos especializados diversas estrategias a través de los cuales puedan ser encontradas las raíces de una ecuación cuadrática. Los métodos encontrados son los siguientes:

## Alternativa 1. Método de Newton­Raphson

Es un método general para encontrar aproximaciones de las raíces de una función real.

Sea *f* : [*a*,*b*] → *R* una función derivable definida en el intervalo real [a,b]. Empezamos con un valor inicial *x*0 y definimos para cada número natural n: *xn*+1 = *xn* − *ff*′((*xxnn*)) . Donde *f*′ denota la derivada de f con respecto a x.

2

## Alternativa 2. Normalización y Factorización

La ecuación cuadrática *ax*[[1]](#footnote-1) + *bx* + *c* = 0 puede ser normalizada, de tal manera que se reduzca a la ecuación *x*2 + *b*′*x* + *c*′ = 0 . Es decir, una ecuación cuadrática donde *a* = 1 : *b*′ = *ba* y *c*′ = *ac* .

Luego la expresión puede descompuesta en factores de la forma (*x* − *r*1) y (*x* − *r*2) tal que *ax*2 + *bx* + *c* = *x*2 + *b*′*x* + *c*′ = (*x* − *r*1)(*x* − *r*2) = 0 .

Esta factorización se consigue encontrando *r*1 + *r*2 = *b*′ al tiempo que *r*1 \* *r*2 = *c*′.

## Alternativa 3. Fórmula de la Ecuación Cuadrática

Utilizar la fórmula de la ecuación cuadrática: *r* = −*b*±√2*ba*2−4*ac*

**Paso 4. Transición de las Ideas a los Diseños Preliminares**

Lo​ primero que hacemos en este paso es descartar las ideas que no son factibles. En este sentido ​***descartamos******la***

***Alternativa 2 (Normalización y Factorización)*** debido a la dificultad que supone encontrar *r*1 y *r*2 que satisfagan *r*1 + *r*2 = *b*′ al tiempo que *r*1 \* *r*2 = *c*′, ya que resolver este par de ecuaciones de forma analítica nos pone un

problema similar al problema original de encontrar las raíces de una función cuadrática.

La revisión cuidadosa de las otras alternativas nos conduce a lo siguiente:

*Alternativa 1. Método de Newton­Raphson*.

­ Es un método abierto en el que no está garantizada su convergencia global.

­ El método puede no converger si la función presenta múltiples puntos de inflexión y pendientes pronunciadas cerca de la raíz. Sin embargo, no es el caso de la función cuadrática por lo que esta característica no sería una desventaja.

­ El método implica iterar desde un primer valor aproximado hasta llegar a un valor tan cercano como un valor de error mínimo indicado sea alcanzado.

­ El método requiere de un valor inicial que debe estar cerca de la raíz para aumentar la probabilidad y rapidez de convergencia.

## Alternativa 3. Fórmula de la Ecuación Cuadrática

­ No es posible calcular las raíces de cualquier función cuadrática directamente utilizando las operaciones aritméticas simples de un equipo de cómputo, porque el cálculo del discriminante *b*2 − 4*ac* puede llevar a valores negativos, lo cual produce números imaginarios al requerirse el cálculo de la raíz cuadrada del discriminante en la fórmula, en casos que éste sea menor a 0.

# Paso 5. Evaluación y Selección de la Mejor Solución

## Criterios

Deben definirse los criterios que permitirán evaluar las alternativas de solución y con base en este resultado elegir la solución que mejor satisface las necesidades del problema planteado. Los criterios que escogimos en este caso son los que enumeramos a continuación. Al lado de cada uno se ha establecido un valor numérico con el objetivo de establecer un peso que indique cuáles de los valores posibles de cada criterio tienen más peso (i.e., son más deseables).

­ *Criterio A.* Precisión de la solución. La alternativa entrega una solución:

­ [2] Exacta (se prefiere una solución exacta)

­ [1] Aproximada

­ *Criterio B.* Eficiencia. Se prefiere una solución con mejor eficiencia que las otras consideradas. La eficiencia puede ser:

­ [4] Constante

­ [3] Mayor a constante

­ [2] Logarítimca

­ [1] Líneal

­ *Criterio C.* Completitud. Se prefiere una solución que encuentre todas las soluciones. Cuántas soluciones entrega:

­ [3] Todas

­ [2] Mas de una si las hay, aunque no todas

­ [1] Solo una o ninguna

­ *Criterio D.* Facilidad en implementación algorítmica:

­ [2] Compatible con las operaciones aritméticas básicas de un equipo de cómputo moderno

­ [1] No compatible completamente con las operaciones aritméticas básicas de un equipo de cómputo moderno

## Evaluación

Evaluando los criterios anteriores en las alternativas que se mantienen, obtenemos la siguiente tabla:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Criterio A** | **Criterio B** | **Criterio C** | **Criterio D** | **Total** |
| Alternativa 1. Método de  Newton­Raphson | Aproximada 1 | Mayor a constante 3 | Solo una 1 | Compatible 2 | 7 |
| Alternativa 2. Fórmula de la Ecuación Cuadrática | Exacta 2 | Constante 4 | Todas  3 | No compatible  (directamente) 1 | 10 |

## Selección

De acuerdo con la evaluación anterior se debe seleccionar la Alternativa 2, ya que obtuvo la mayor puntuación de acuerdo con los criterios definidos. Se debe tener en cuenta que hay que hacer un manejo adecuado del criterio en el cual la alternativa fue peor evaluada que la otra alternativa.

# Paso 6. Preparación de Informes y Especificaciones

*Especificación del Problema* (en términos de entrada/salida)

*Problema:* Raíces de una función cuadrática

*Entradas:* coeficientes de la función *f*(*x*) = *ax*2 + *bx* + *c* solo es necesario conocer los valores de *a* , *b* y *c*.

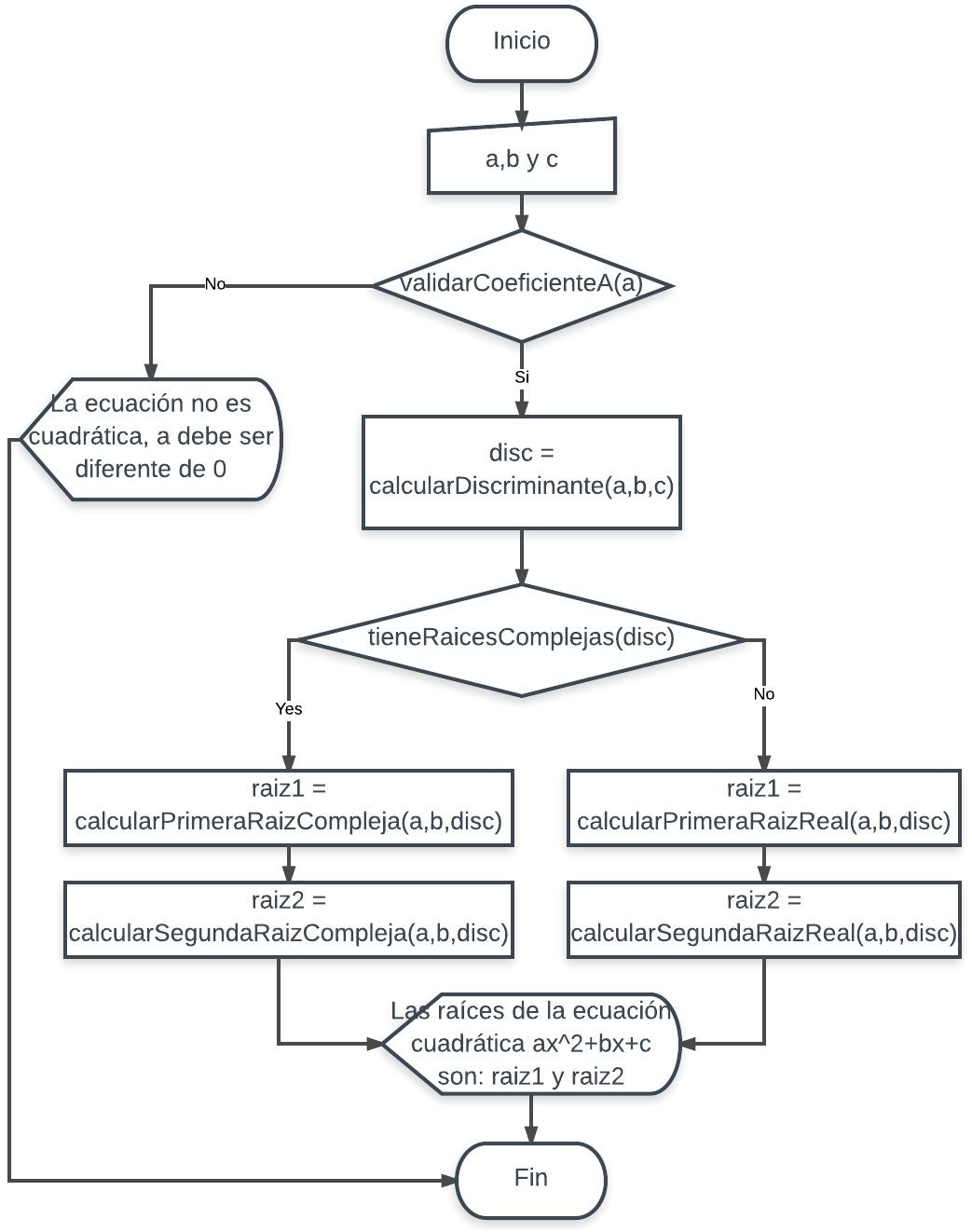
*Salida:* *r*1 y *r*2 tal que *ar*12 + *br*1 + *c* = 0 y *ar*22 + *br*2 + *c* = 0 . *r*1 y *r*2 pueden ser números reales o complejos.

## Consideraciones

Se deben tener en cuenta los siguientes casos para calcular el valor de las raíces:

1. La función tiene dos soluciones reales, que ocurre cuando la parábola descrita por la función cuadrática corta el eje X en dos puntos diferentes. En este caso el discriminante es positivo.
2. La función tiene una solución real, que ocurre cuando la parábola es tangente al eje X. Aquí el discriminante es cero.
3. No hay solución real. La parábola no corta el eje X. El discriminante es menor que cero, y por tanto la función puede tener una o dos soluciones complejas.

## Diagrama de Flujo del Algoritmo Pseudocódigo del Algoritmo



if(!validarCoeficienteA())​

imprimir “No es una ecuación cuadrática”; else disc ← calcularDiscriminante(a, b, c); if(tieneRaicesComplejas(disc)){ raiz1 ← calcularPrimeraRaizCompleja(a, b, disc); raiz2 ← calcularSegundaRaizCompleja(a, b, disc); else raiz1 ← ""+calcularPrimeraRaizReal(a, b, disc); raiz2 ← ""+calcularSegundaRaizReal(a, b, disc);

imprimir "\nLas raíces de la ecuación cuadrática:\n"

+ a +"x^2 + "+b+"x + "+c+" = 0, son:\n"

+ "Raíz 1: "+raiz1+"\n"

+ "Raíz 2: "+raiz2+"\n";

# Paso 7. Implementación del Diseño

Implementación en un Lenguaje de Programación.

Lista de Tareas a implementar:

a. ​ Validar los coeficientes​

b. ​ Calcular el discriminante​

c. ​ Verificar si hay raíces complejas​

d. ​ Calcular la primera raíz real​

e. ​ Calcular la segunda raíz real​

f. ​ Calcular la primera raíz compleja​

g. Calcular la segunda raíz compleja

# Construcción

|  |  |
| --- | --- |
| **Especificación de Subrutinas**  Validar los coeficientes | Escritura del código en un Lenguaje de Programación  ­Java en este caso­ |

|  |  |
| --- | --- |
| Nombre: | validarCoeficienteA |
| Descripción: | Valida que el coeficiente ​*a* sea diferente de 0 |
| Entrada: | ­ ​*a*: double, es el coeficiente de la incognita al cuadrado |
| Retorno: | boolean, true si el coeficiente es válido y false si no lo es |

public boolean validarCoeficienteA(double a){ if(a == 0){

return false;

}else{

return true;

}

}

Calcular el discriminante

|  |  |
| --- | --- |
| Nombre: | calcularDiscriminante |
| Descripción: | Calcula el discriminante de la ecuación cuadrática con base en los coeficientes |
| Entrada: | ­ a: double, coeficiente de x^2  ­ b: double, coeficiente de x  ­ c: double, valor constante |
| Retorno: | double, el discriminante ya  calculado |

public double calcularDiscriminante(double a,

double b, double c){

double disc;

disc = b\*b – 4\*a\*c;

return disc;

}

Verificar si hay raíces complejas

|  |  |
| --- | --- |
| Nombre: | sonRaicesComplejas |
| Descripción: | Verifica el discriminante para  determinar si las raíces son complejas o no |
| Entrada: | ­ discrim: double, es el discriminante de la ecuación |
| Retorno: | boolean, true si las raíces son complejas y false si son reales |

public boolean sonRaicesComplejas(

double discrim){

if(discrim<0){

return true;

}else{

return false;

} }

Calcular la primera raíz real

|  |  |
| --- | --- |
| Nombre: | calcularSolucionRealUno |
| Descripción: | Calcula la primera raíz real |
| Entrada: | ­ a: double, el coeficiente de la incognita al cuadrado  ­ b: double, el coeficiente de la incognita  ­ d: double, el discriminante |
| Retorno: | double, el valor de la primera raíz real |

public double calcularSolucionRealUno(double a

double b, double d){

double raizUno;

raizUno = (­b + Math.sqrt(d))/(2\*a);

return raizUno;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Calcular la segunda raíz real   |  |  | | --- | --- | | Nombre: | calcularSolucionRealDos | | Descripción: | Calcula la segunda raíz real | | Entrada: | ­ a: double, el coeficiente de la incognita al cuadrado  ­ b: double, el coeficiente de la incognita  ­ d: double, el discriminante | | Retorno: | double, el valor de la segunda raíz real |   public double calcularSolucionRealDos(double a  double b, double d){  double raizDos;    raizDos = (­b ­ Math.sqrt(d))/(2\*a);    return raizDos;  }   |  |  | | --- | --- | |  |  | | Calcular la primera raíz compleja | public String calcularSolucionComlejaDos(double a |  |  |  | | --- | --- | | Nombre: | calcularSolucionComplejaUno | | Descripción: | Calcula la primera raíz compleja | | Entrada: | ­ a: double, el coeficiente de la incognita al cuadrado  ­ b: double, el coeficiente de la incognita  ­ d: double, el discriminante | | Retorno: | String, el número complejo que representa la primera solución compleja |   double b, double d){  String solComplejaDos;  double pReal, pImag;    pReal = ­b/(2\*a);  pImag = Math.sqrt(­d)/(2\*a);    solComplejaDos = pReal+” ­ ”+pImag+”i”;    return solComplejaDos; } |

}

Calcular la segunda raíz compleja

|  |  |
| --- | --- |
| Nombre: | calcularSolucionComplejaDos |
| Descripción: | Calcula la segunda raíz compleja |
| Entrada: | ­ a: double, el coeficiente de la incognita al cuadrado  ­ b: double, el coeficiente de la incognita  ­ d: double, el discriminante |
| Retorno: | String, el número complejo que representa la segunda solución compleja |

public String calcularSolucionComlejaUno(double a

double b, double d){

String solComplejaUno;

double pReal, pImag;

pReal = ­b/(2\*a);

pImag = Math.sqrt(­d)/(2\*a);

solComplejaUno = pReal+” + ”+pImag+”i”;

return solComplejaUno;

}

public String encontrarRaicesFuncionCuadratica(double a, double b, double c){ if(!validarCoeficienteA()){

return “No es una ecuación cuadrática”;

}

double disc;

String raiz1, raiz2;

//Se invoca el método calcularDiscriminante

disc = calcularDiscriminante(a, b, c);

//Se verifica cúal es la naturaleza de las raíces y se calculanCofe

if(tieneRaicesComplejas(disc)){

raiz1 = calcularPrimeraRaizCompleja(a, b, disc);

raiz2 = calcularSegundaRaizCompleja(a, b, disc);

}else{

raiz1 = ""+calcularPrimeraRaizReal(a, b, disc);

raiz2 = ""+calcularSegundaRaizReal(a, b, disc);

}

//Se retorna el resultado

return "\nLas raíces de la ecuación cuadrática:\n"

+ a +"x^2 + "+b+"x + "+c+" = 0, son:\n"

+ "Raíz 1: "+raiz1+"\n"

+ "Raíz 2: "+raiz2+"\n";

}

1. ​http://fourier.eng.hmc.edu/e176/lectures/NM/node20.html [↑](#footnote-ref-1)