

Universidad Libre Seccional Cali

Métricas de Software

**“Investigación de herramientas de medición y pruebas de desempeño (profiling) en el lenguaje de programación Java”**

Juan Camilo Mamian Ruiz

Aldemar Bernal Hernández

Prof. Rafael Alberto Moreno

Facultad de Ingeniería

Programa Ingeniería de Sistemas

2021-1

**Contenido**

[**Instalación de herramienta para pruebas de desempeño Java** 3](#_Toc69008852)

[**Algoritmos de Ordenamiento Java** 5](#_Toc69008853)

[**¿Cómo utilizar la herramienta “Profiling” en Apache NetBeans?** 9](#_Toc69008854)

[**Mediciones con el Profiling de Apache NetBeans a el Programa de Algoritmos de Ordenamiento Java** 12](#_Toc69008855)

[**Gráficos Estadísticos de Medición por código “currentTimeMillis();”** 14](#_Toc69008856)

[**Relación entre las Medidas Arrojadas por la Herramienta “Profiling” vs el Medidor por Código “currentTimeMillis();”** 16](#_Toc69008857)

[**¿Hay una relación entre los resultados arrojados por el “profiler” y el medidor código? ¿hay similitud?** 21](#_Toc69008858)

# **Instalación de herramienta para pruebas de desempeño Java**

Las herramientas de medición y desempeño **“Profiling”** nos permiten realizar pruebas en nuestro código, como, por ejemplo: tiempos de ejecución, porcentajes de utilización de CPU, uso de memoria, entre otros. Para nuestro caso, la herramienta de pruebas de desempeño ya viene integrada en un entorno de desarrollo integrado libre, el nombre de este entorno de desarrollo es **Apache NetBeans** el cual es gratuito.

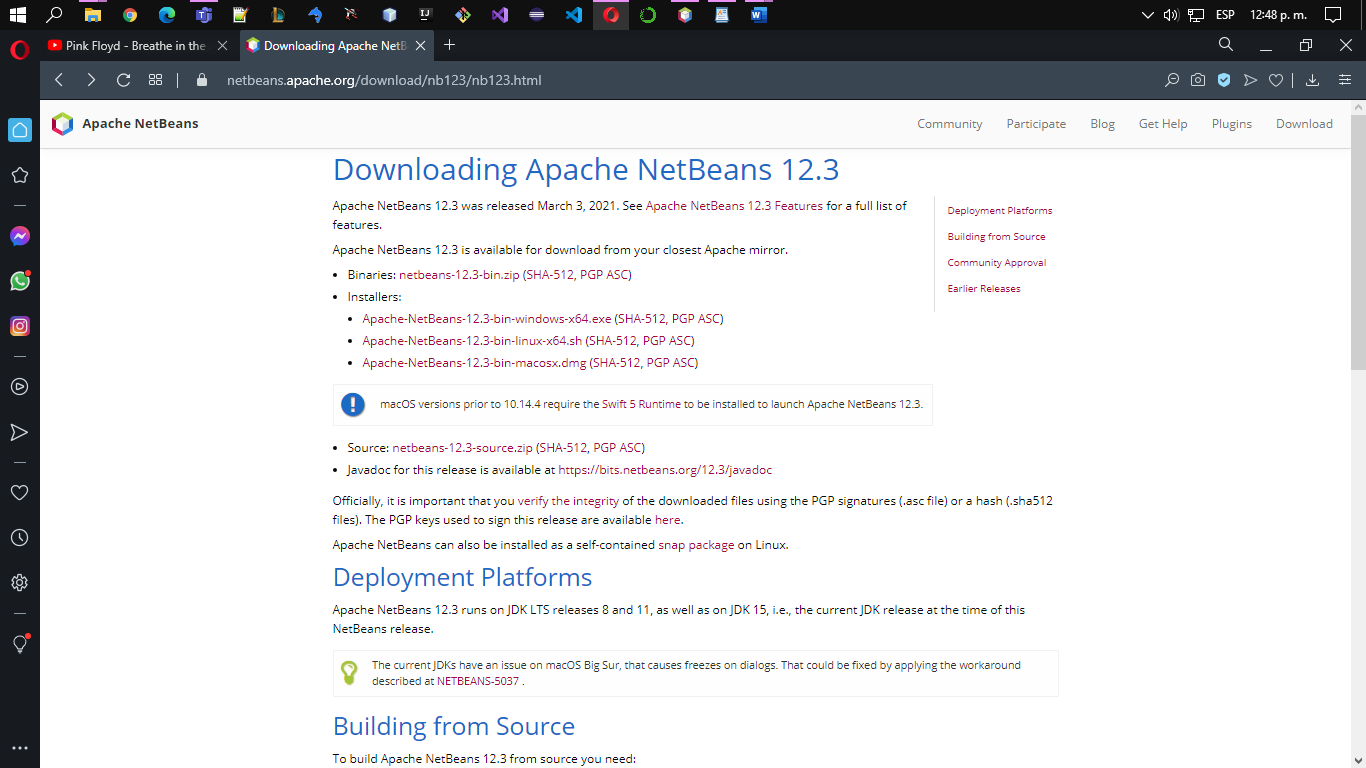
La instalación de **Apache NetBeans** es muy sencilla, primeramente, debemos tener el **Kit de desarrollo de Java JDK (Java Development Kit)**. Lo podemos encontrar y descargar en el siguiente enlace:

<https://www.oracle.com/java/technologies/javase-downloads.html>

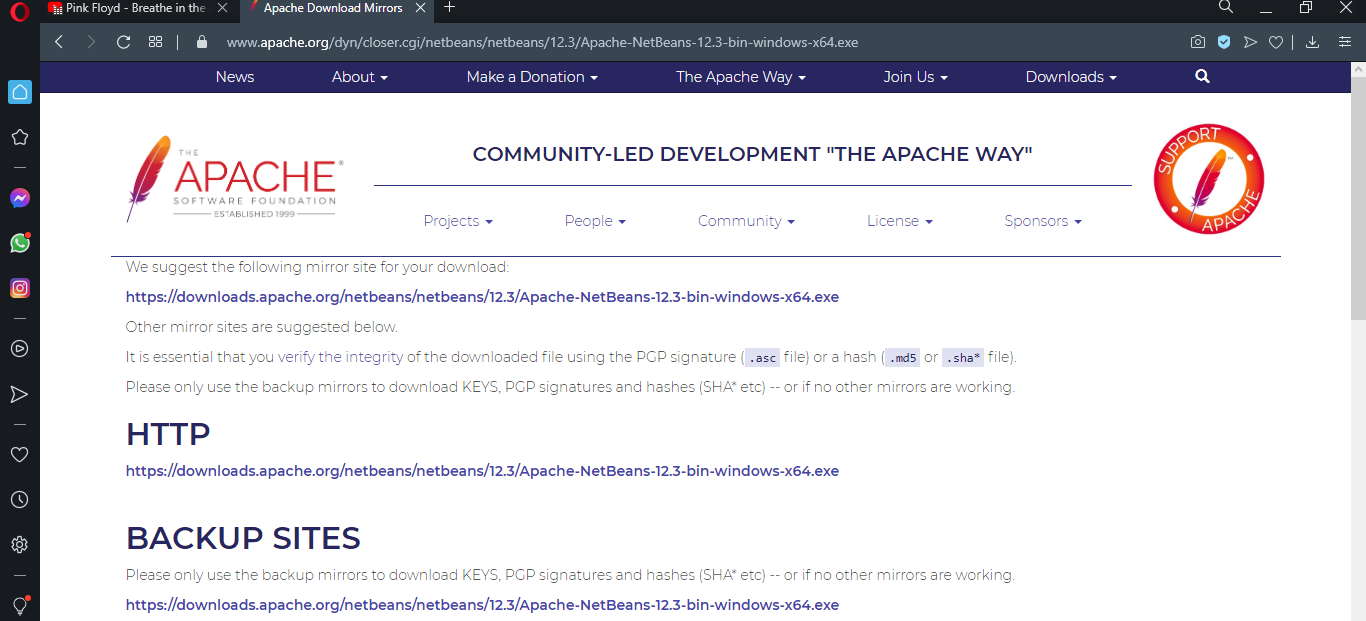
Una vez realizado la descargar del kit, procedemos con la de **Apache NetBeans**. Nos dirigimos al siguiente enlace:

<https://netbeans.apache.org/download/nb123/nb123.html>

Deberá aparecernos algo como esto:

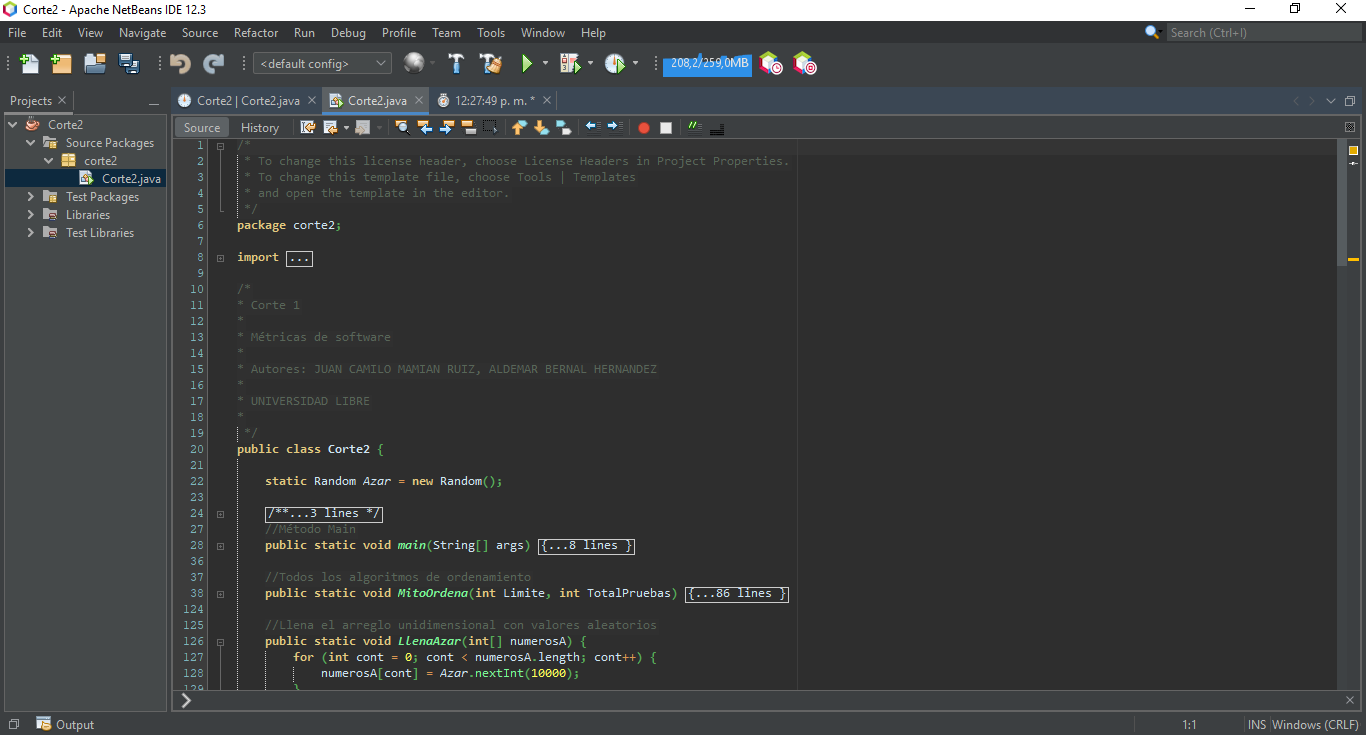


Procedemos a elegir el **.exe** que sea semejante a nuestra arquitectura y sistema operativo de nuestro ordenador. Se encuentra en la sección con título **Downloading Apache Netbeans 12.3**.

Una vez hecho esto, la página nos dirige al link de descargar, donde al presionarlo automáticamente se descargará él **.exe**.

Seleccionamos el link que está debajo del título **HTTP** y listo.

Al ya descargar él **.exe**, la instalación de escritorio es muy sencilla, no toca hacer ninguna configuración manual salvo a darle siguiente y aceptar términos y condiciones. Terminada la instalación, ejecutamos el entorno y aparecerá algo como esto:



Concluido, tenemos el **Apache NetBeans** junto con la herramienta de desempeño **“Profiling”**, la herramienta se encuentra en la parte superior con el siguiente icono: 

La utilización de la herramienta se explicará a continuación con un programa escrito en Java, que contiene algoritmos de ordenamiento conocidos por los desarrolladores **(“Burbuja, QuickSort, Selección, Inserción y Shell”)**.

# **Algoritmos de Ordenamiento Java**

Primeramente, se hizo una conversión, el programa originalmente estaba escrito en **C#** y fue trasladado a **Java**:

**Código en Java**

**package** corte2;

**import** **java.util.Random**;

/\*

\* Corte 2

\*

\* Métricas de software

\*

\* Autores: JUAN CAMILO MAMIAN RUIZ, ALDEMAR BERNAL HERNANDEZ

\*

\* UNIVERSIDAD LIBRE

\*

\*/

**public** **class** **Corte2** {

**static** Random Azar = **new** Random();

/\*\*

\* @param args the command line arguments

\*/

//Método Main

**public** **static** **void** **main**(String[] args) {

// TODO code application logic here

**int** numPruebas = **10**;

**for** (**int** numElemento = **1000**; numElemento <= **1000**; numElemento += **1000**) {

MitoOrdena(numElemento, numPruebas);

}

}

//Todos los algoritmos de ordenamiento

**public** **static** **void** **MitoOrdena**(**int** Limite, **int** TotalPruebas) {

//Arreglos

**int**[] numerosA = **new** **int**[Limite];

**int**[] numerosB = **new** **int**[Limite];

**long** TPShell = **0**, TPIns = **0**, TPSel = **0**, TPBur = **0**, TPQuick = **0**;

**long** valor = **0**;

**long** inicio = **0**, fin = **0**;

//Para disminuir oscilaciones en el tiempo, se hacen

//N pruebas con cada grupo de pruebas

**for** (**int** prueba = **1**; prueba <= **1**; prueba++) {

LlenaAzar(numerosA);

//Ordenación por método Shell

System.arraycopy(numerosA, **0**, numerosB, **0**, numerosA.length);

inicio = System.currentTimeMillis();

Shell(numerosB);

fin = System.currentTimeMillis();

TPShell += fin - inicio;

valor += numerosB[**0**];

//Reset

inicio = **0**;

fin = **0**;

//Ordenación por método Inserción

System.arraycopy(numerosA, **0**, numerosB, **0**, numerosA.length);

inicio = System.currentTimeMillis();

Insercion(numerosB);

fin = System.currentTimeMillis();

TPIns += fin - inicio;

valor += numerosB[**0**];

//Reset

inicio = **0**;

fin = **0**;

//Ordenación por método Selección

System.arraycopy(numerosA, **0**, numerosB, **0**, numerosA.length);

inicio = System.currentTimeMillis();

Seleccion(numerosB);

fin = System.currentTimeMillis();

TPSel += fin - inicio;

valor += numerosB[**0**];

//Reset

inicio = **0**;

fin = **0**;

//Ordenación por método Burbuja

System.arraycopy(numerosA, **0**, numerosB, **0**, numerosA.length);

inicio = System.currentTimeMillis();

Burbuja(numerosB);

fin = System.currentTimeMillis();

TPBur += fin - inicio;

valor += numerosB[**0**];

//Reset

inicio = **0**;

fin = **0**;

//Ordenación por método QuickSort

System.arraycopy(numerosA, **0**, numerosB, **0**, numerosA.length);

inicio = System.currentTimeMillis();

QuickSort(numerosB, **0**, numerosB.length - **1**);

fin = System.currentTimeMillis();

TPQuick += fin - inicio;

valor += numerosB[**0**];

}

**double** TS = (**double**) TPShell;

**double** TI = (**double**) TPIns;

**double** TL = (**double**) TPSel;

**double** TB = (**double**) TPBur;

**double** TQ = (**double**) TPQuick;

String CSV = "#Datos: " + Integer.toString(Limite) + " Shell: " + Double.toString(TS) + " Insert: " + Double.toString(TI) + " Select: " + Double.toString(TL) + " Burb: " + Double.toString(TB) + " Quick: " + Double.toString(TQ);

CSV = CSV.replace('.', ',');

System.out.print(CSV);

System.out.println(" val: " + Long.toString(valor));

}

//Llena el arreglo unidimensional con valores aleatorios

**public** **static** **void** **LlenaAzar**(**int**[] numerosA) {

**for** (**int** cont = **0**; cont < numerosA.length; cont++) {

numerosA[cont] = Azar.nextInt(**10000**);

}

}

//Ordenamientos Insert

**public** **static** **void** **Insercion**(**int**[] arreglo) {

**int** tmp;

**int** j;

**for** (**int** i = **1**; i < arreglo.length; i++) {

tmp = arreglo[i];

**for** (j = i - **1**; j >= **0** && arreglo[j] > tmp; j--) {

arreglo[j + **1**] = arreglo[j];

}

arreglo[j + **1**] = tmp;

}

}

//Ordenamiento por Selección

**public** **static** **void** **Seleccion**(**int**[] arreglo) {

**for** (**int** i = **0**; i < arreglo.length - **1**; i++) {

**int** min = i;

**for** (**int** j = i + **1**; j < arreglo.length; j++) {

**if** (arreglo[j] < arreglo[min]) {

min = j;

}

}

**if** (i != min) {

**int** aux = arreglo[i];

arreglo[i] = arreglo[min];

arreglo[min] = aux;

}

}

}

//Ordenamiento por Burbuja

**public** **static** **void** **Burbuja**(**int**[] Arreglo) {

**int** n = Arreglo.length;

**int** tmp;

**for** (**int** i = **0**; i < n - **1**; i++) {

**for** (**int** j = **0**; j < n - **1**; j++) {

**if** (Arreglo[j] > Arreglo[j + **1**]) {

tmp = Arreglo[j];

Arreglo[j] = Arreglo[j + **1**];

Arreglo[j + **1**] = tmp;

}

}

}

}

//Ordenamiento por Shell

**public** **static** **void** **Shell**(**int**[] arreglo) {

**int** N = arreglo.length;

**int** incremento = N;

**do** {

incremento = incremento / **2**;

**for** (**int** k = **0**; k < incremento; k++) {

**for** (**int** i = incremento + k; i < N; i += incremento) {

**int** j = i;

**while** (j - incremento >= **0** && arreglo[j] < arreglo[j - incremento]) {

**int** tmp = arreglo[j];

arreglo[j] = arreglo[j - incremento];

arreglo[j - incremento] = tmp;

j -= incremento;

}

}

}

} **while** (incremento > **1**);

}

//Ordenación por QuickSort

**public** **static** **void** **QuickSort**(**int**[] vector, **int** primero, **int** ultimo) {

**int** i, j, central;

**double** pivote;

central = (primero + ultimo) / **2**;

pivote = vector[central];

i = primero;

j = ultimo;

**do** {

**while** (vector[i] < pivote) {

i++;

}

**while** (vector[j] > pivote) {

j--;

}

**if** (i <= j) {

**int** temp;

temp = vector[i];

vector[i] = vector[j];

vector[j] = temp;

i++;

j--;

}

} **while** (i <= j);

**if** (primero < j) {

QuickSort(vector, primero, j);

}

**if** (i < ultimo) {

QuickSort(vector, i, ultimo);

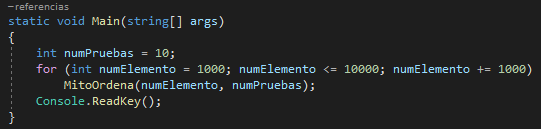
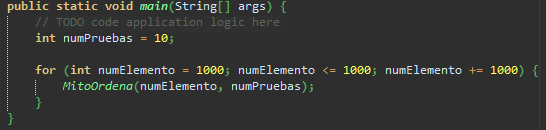
}

}

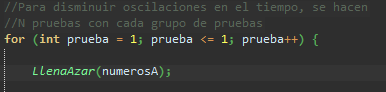
}

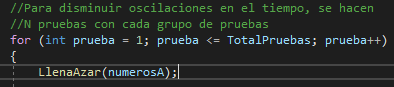
De igual manera el código fuente se encuentra guardado en la carpeta junto con este documento.

Luego, acondicionamos el código para poder realizar el seguimiento de desempeño de cada algoritmo de ordenamiento, para eso modificamos el código de la siguiente forma:

**Comparación C# con Java**

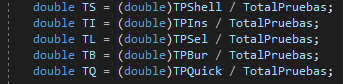
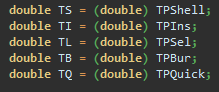
En el ciclo **for** de la imagen izquierda **“C#”**,se encuentra con **numElemento <= 10.000**, cosa que en el lado derecho **“Java”** no es así. Esto se debe porque solo queremos que la prueba de desempeño sea no más hasta **1000 datos**, por el motivo de que, si se aumentan más datos, no sabremos respectivamente cuanto fue el rendimiento con solamente **1000 datos**.

Seguido, se modificó otro ciclo **for** por el cual hacía que repitiera 10 veces la prueba de desempeño a través de código, **“Stopwatch”** en **C#** o **“currentTimeMillis”** en Java.



En la condición del ciclo **for** en la imagen izquierda **“C#”** se encuentra un **TotalPruebas**, este tiene un valor de 10, cosa que no queremos, ya que el objetivo es que los algoritmos de ordenamiento ordenen **solo una vez** para mirar su desempeño, como se muestra al lado derecho **“Java”**.

Por último, se modificaron las variables donde se almacenaban los tiempos de ejecución de cada algoritmo de ordenamiento.



Se hizo esto porque el programa originalmente **“C# imagen izquierda”** sacaba un promedio con 10 datos acumulados, por el ciclo mencionado anteriormente con la condición **prueba <=** **TotalPruebas**, donde **TotalPruebas** es igual a 10. Entonces, al quitar las divisiones dejándoles solo la variable de acumulación de tiempo, podemos saber cuánto se demoró con 1000 datos, **una sola vez**, los algoritmos de ordenamiento como se muestra en la imagen derecha **“Java”**.

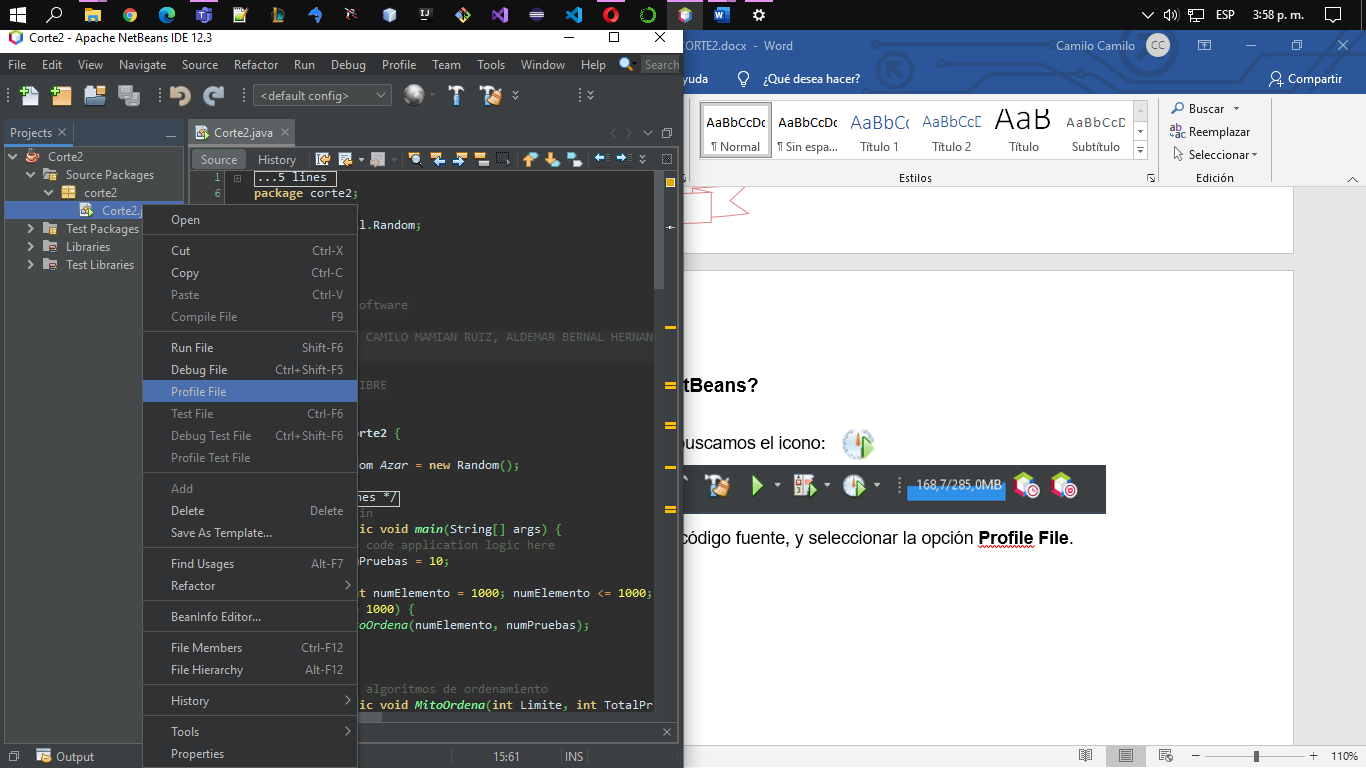
# **¿Cómo utilizar la herramienta “Profiling” en Apache NetBeans?**



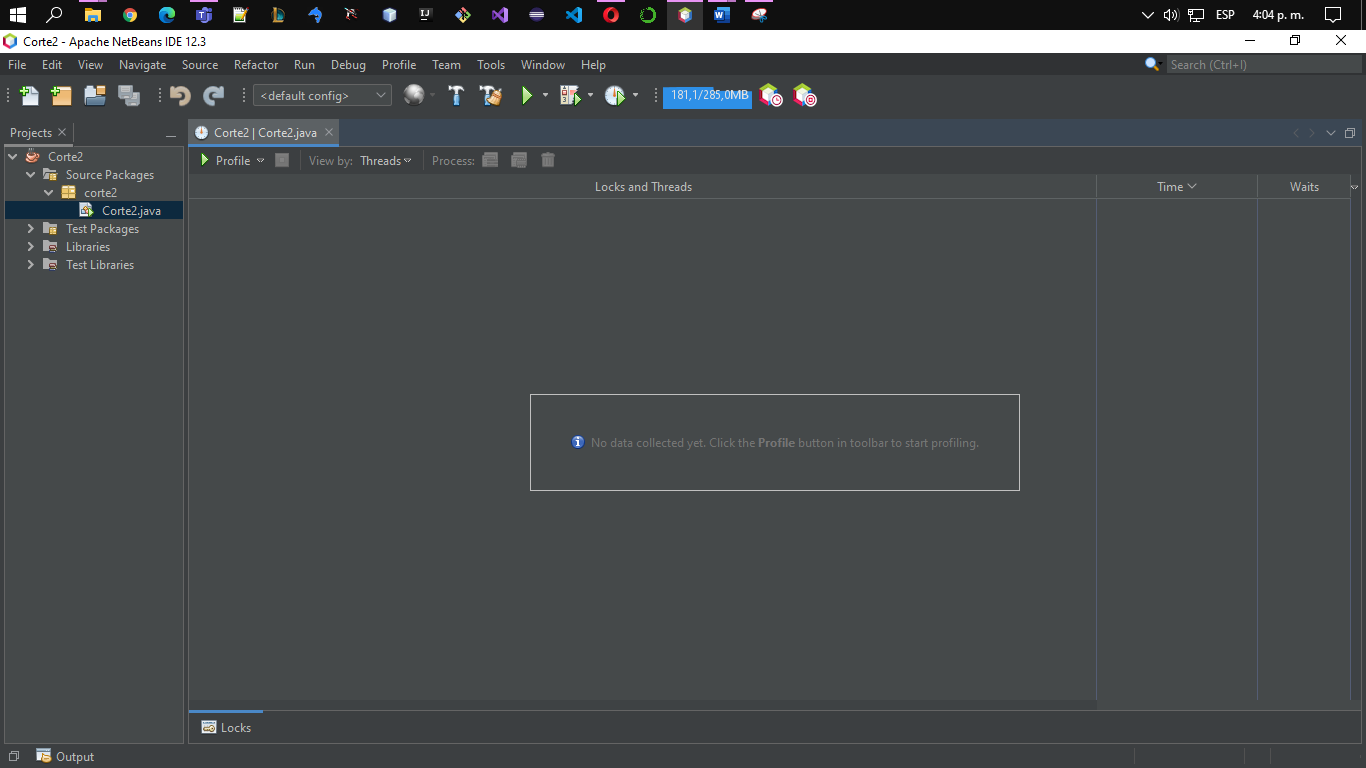
Nos dirigimos a la cinta de herramientas en la parte superior, y buscamos el icono:



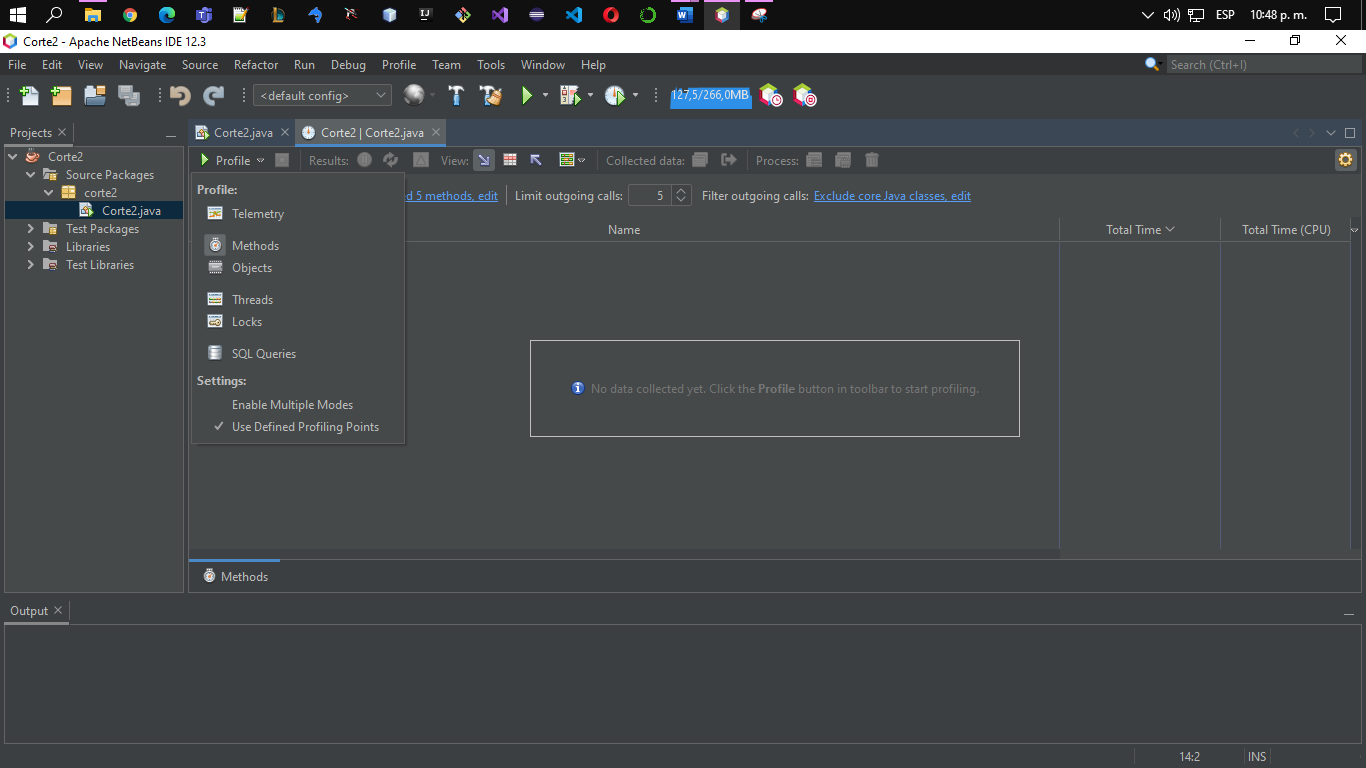
O también podemos dar clic derecho al archivo que contiene el código fuente, y seleccionar la opción **“Profile File”**.



Deberá aparecer algo como esto:



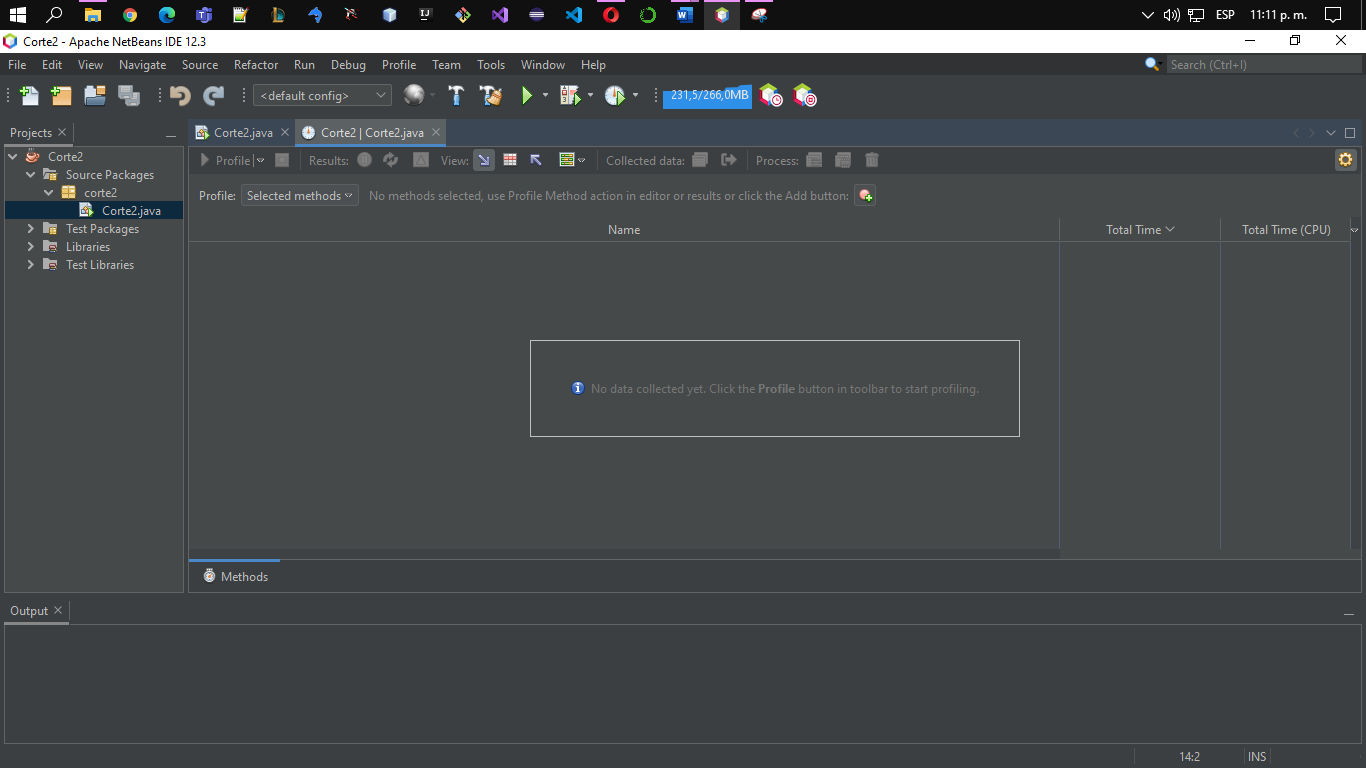
En la parte superior se encuentra un botón  . La pestañita desplegable  permite escoger a lo que le queremos hacer la respectiva medición (**“Métodos”, “Objetos”, “Telemetría general”, “Hilos”, “Queries” y “Cerraduras”**), como se muestra en la siguiente imagen:



El triángulo verde  nos permite ejecutar la prueba de métrica cuando ya ha sido seleccionada.

Donde nos vamos a centrar es en el item **“Methods”**, porque es el indicado en satisfacer lo que estamos buscando, lo cual es medir el tiempo, desempeño y rendimiento de cada uno de los métodos escritos en el programa **(“Burbuja, QuickSort, Selección, Inserción y Shell”)**.

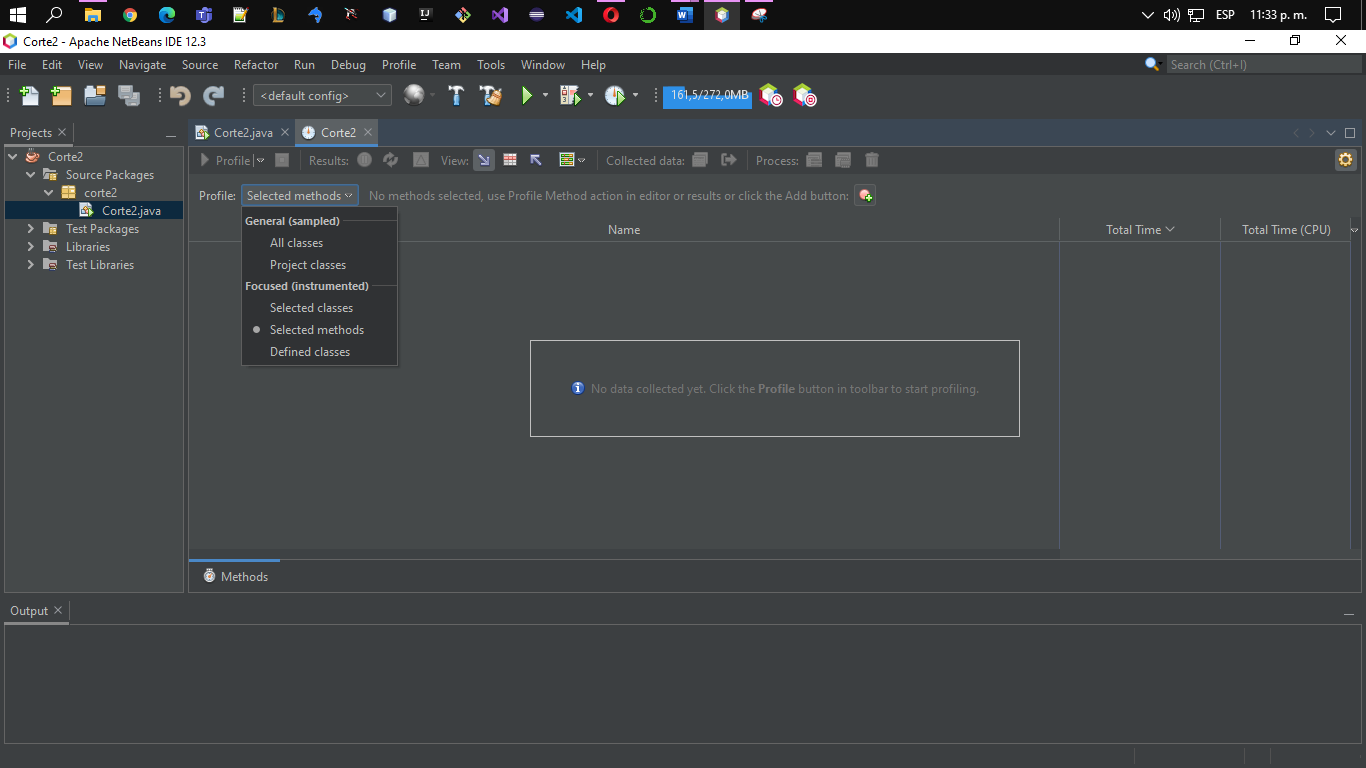
Entonces procedemos a seleccionar el item **“Methods”**, y deberá salir algo como esto:





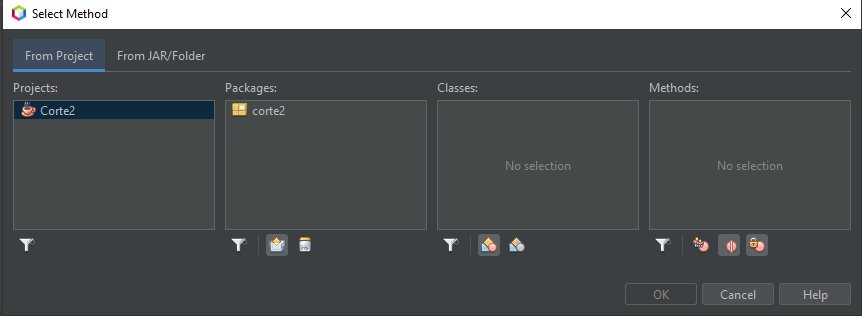
**Nota:** Si no te sale el mensaje o la opción de seleccionar los métodos, da clic en la tuerca .

Nos sale un mensaje indicando que todavía no hemos seleccionado métodos, que con este botón  podemos añadir los métodos para poderles hacer la métrica. Pero, también podemos seleccionar otras cosas con , con el cual tenemos la libertad de escoger clases, clases del proyecto a nivel general, o a nivel más específico de seleccionar los métodos manualmente, como, por ejemplo:

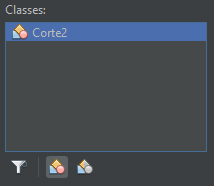
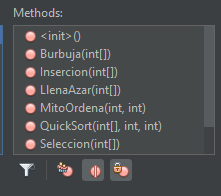


Nos vamos a la sección **“Focused (instrumental)”**, y seleccionamos **“Selected methods”**, para poder seleccionar los métodos manualmente (**“imagen de arriba”**).

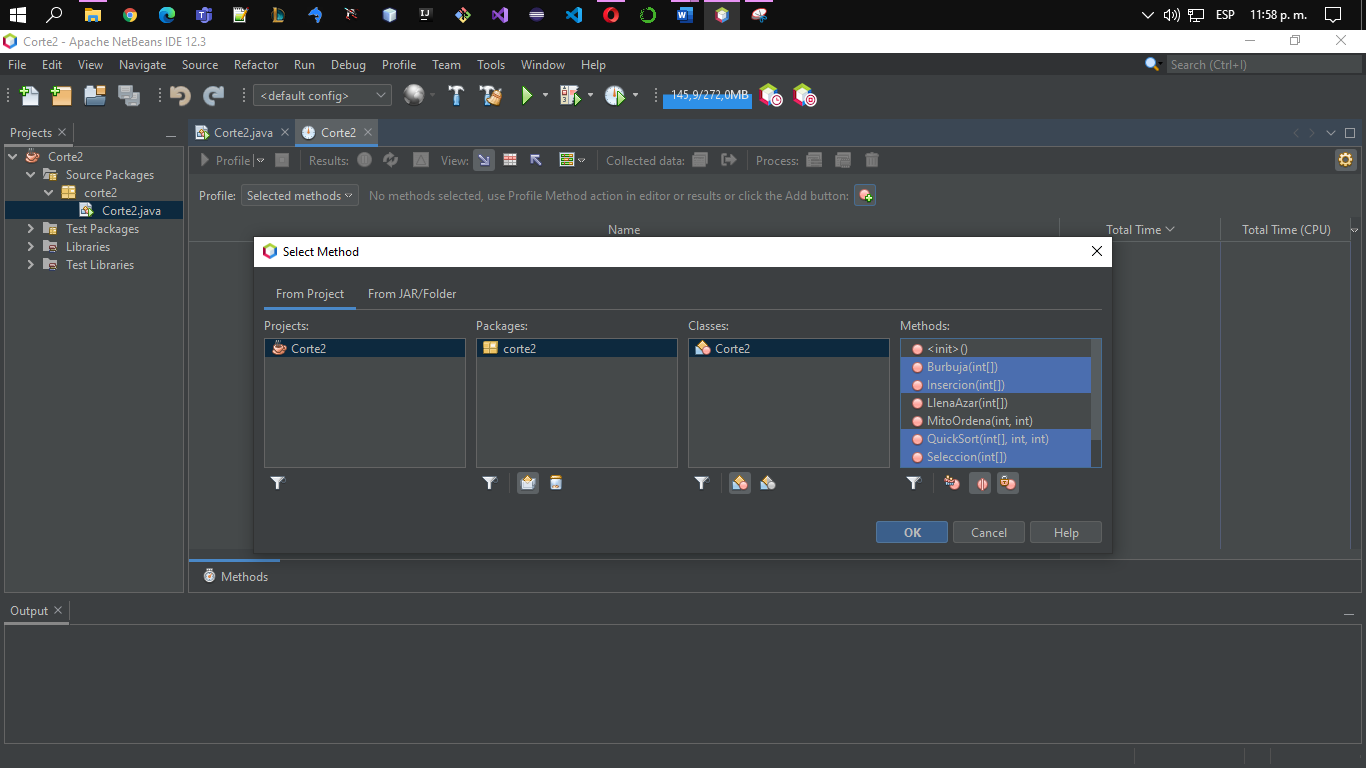
Continuando, le damos clic a , si ya hemos seleccionado **“Selected methods”**, y aparecerá algo como esto:



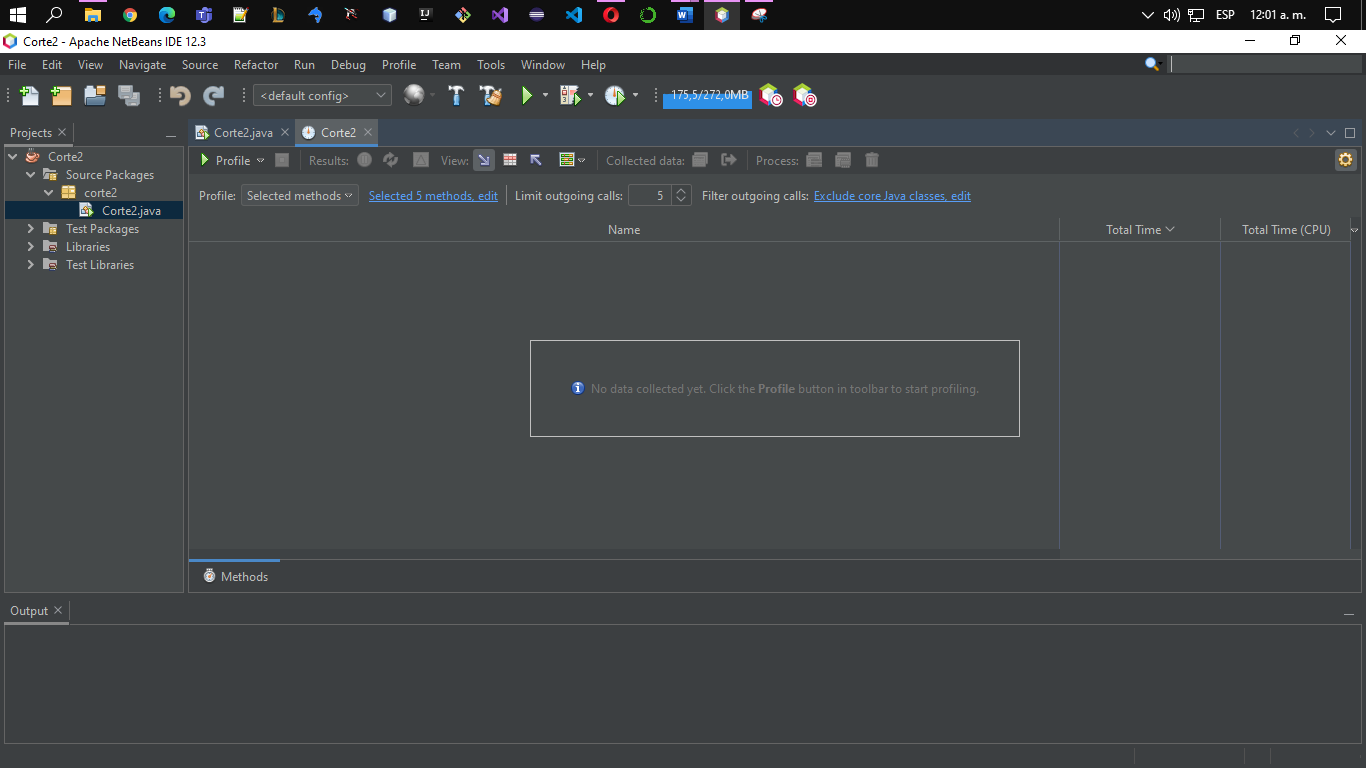
Buscamos los métodos dando doble clic al paquete; luego a la clase; y finalmente aparecerán los métodos:

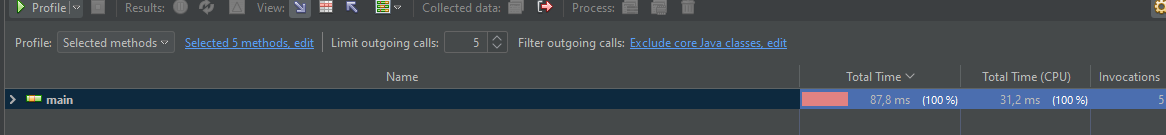
Los seleccionamos y le damos clic al botón **“OK”**:



Luego de darle **“OK”**,ya nos mostrara los métodos seleccionados:



Y listo, así queda configurado para hacerle la métrica a los métodos seleccionados, lo que se procede ahora es presionar botón  y ya nos saldrá la métrica.

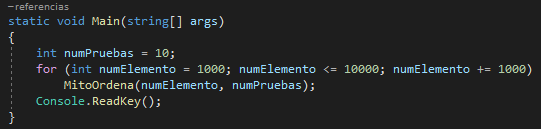
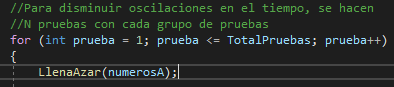
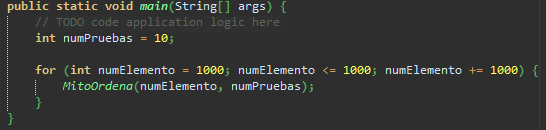
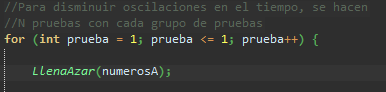


Para ver la métrica, clic en el desplegable  y nos mostrará los métodos.

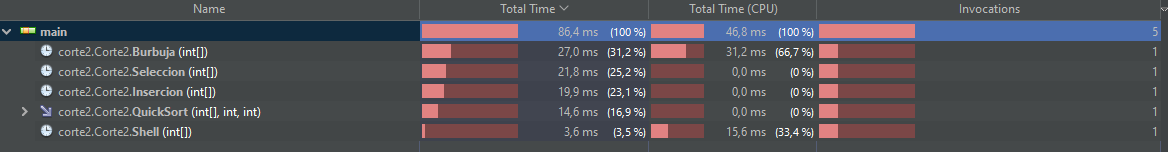
# **Mediciones con el Profiling de Apache NetBeans a el Programa de Algoritmos de Ordenamiento Java**

El paso a paso para realizar las mediciones a los respectivos métodos de ordenamiento, primero que todo, fue la realización anterior que se mostró en la modificación del código **C#** a **Java**. Recordando, lo que se hizo fue acomodar los ciclos **for** para hacer la prueba con ciertos datos, a cierta cantidad de veces. El motivo por el cual se hizo esto es para que la herramienta **“profiling”** nos muestre la métrica con estas condiciones mencionadas (**“1000 datos, 1 sola vez”**).

**Imágenes del Código C# Lado Izquierdo, Java Lado Derecho**



**Métrica con 1000 Datos 1 Sola Vez**



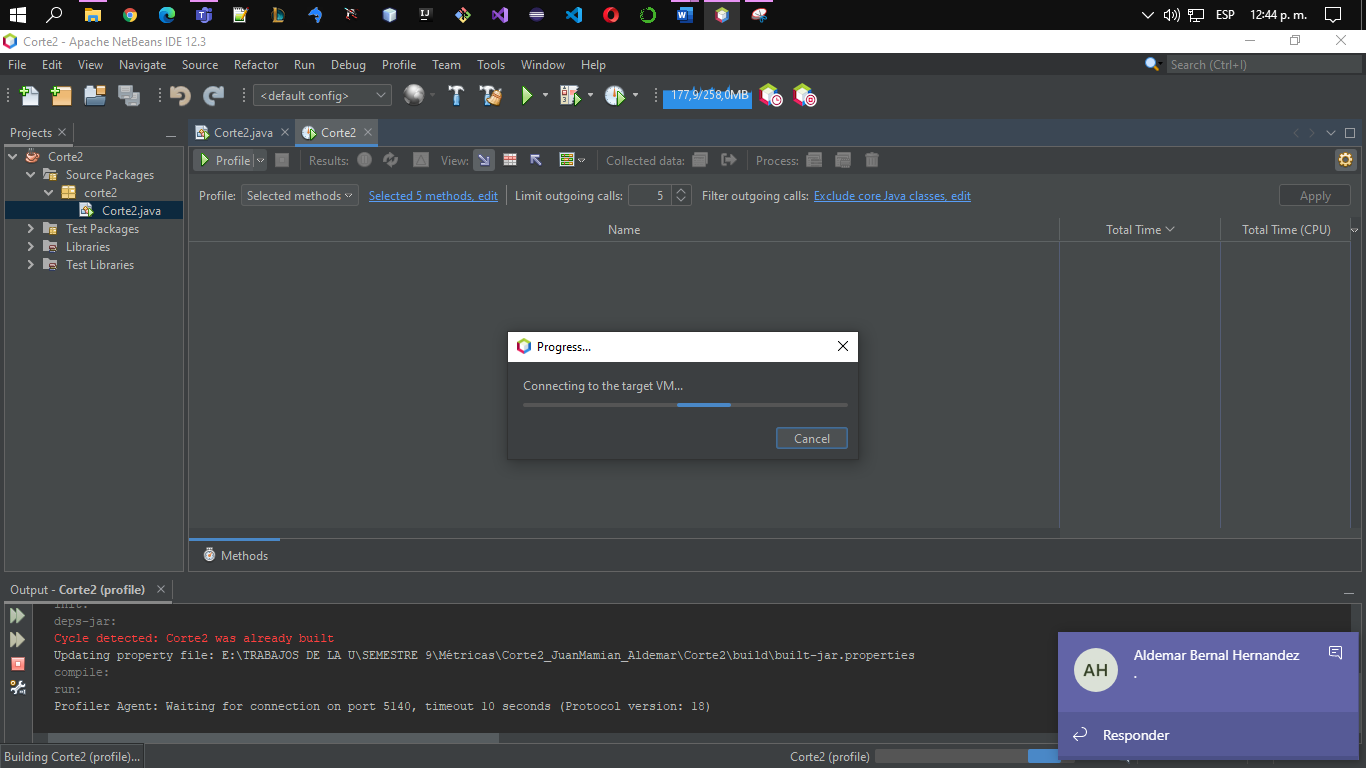
También, cuando ejecutamos el botón , nos ejecuta la consola indicándonos que ha sido satisfactoria la ejecución. Si tienes **“System.out.print();”**, los mostrará después del título  como en la siguiente imagen:



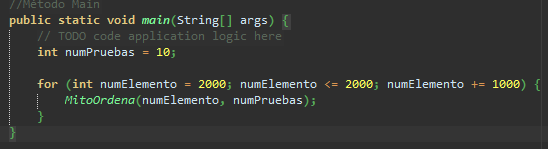
**Nota:** Búscalo con el Scroll del ratón porque arroja muchos mensajes

Entonces, siguiendo esta lógica, para mirar el comportamiento con más datos sería modificar la variable que contiene los 1000 datos, y subirla a 2000, además, que la condición solo llegue a 2000. Entonces, así podemos realizar **n** ejecuciones ejecutando la herramienta y modificando la variable y observar su comportamiento. Por lo que ya podemos comparar la medición por código **“currentTimeMillis();”** y la herramienta **“Profiling”**. por ejemplo, así sería con 2000 datos las siguientes imágenes:

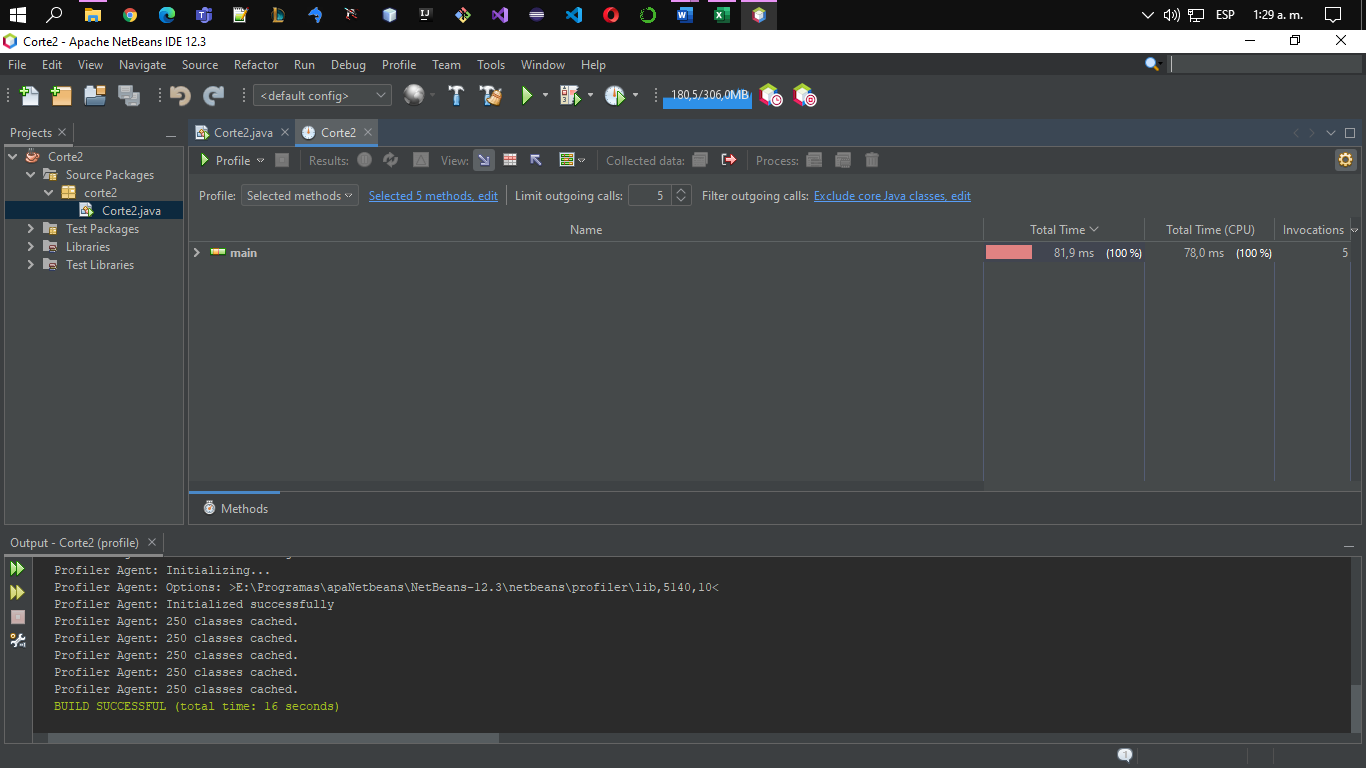
**Ejecutando la Herramienta**



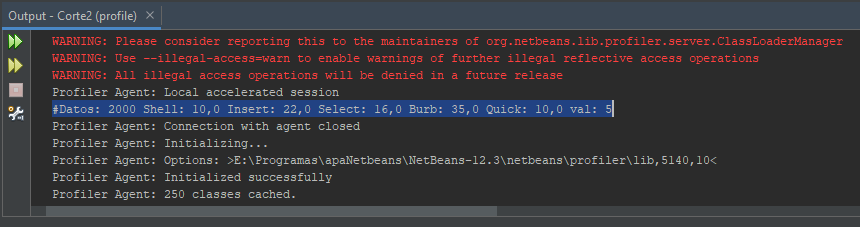
**Modificación de datos**



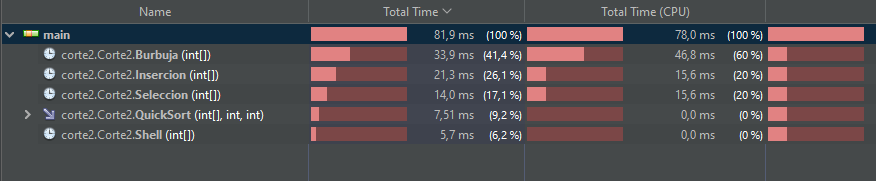
**Finalización de la Métrica**



**Salida por Consola**



**Gráficos “Profiler”**



# **Gráficos Estadísticos de Medición por código “currentTimeMillis();”**

A continuación, se muestran las gráficas y datos obtenidos con la medición por código manualmente, Las cuales son:

* Comportamiento de los algoritmos.
* Comparación entre algoritmos.
* Tabla de datos.
* Promedio de tiempos en el rango de 1000 a 10000 datos de los algoritmos.

**Tabla de Datos “Tiempos en (ms)”**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | DATOS | | | | | | | | | |  |
| Algoritmos | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 | 7000 | 8000 | 9000 | 10000 | Promedio |
| Shell | 4,0 | 4,0 | 3,0 | 5,0 | 3,0 | 17,0 | 8,0 | 9,0 | 6,0 | 7,0 | 6,6 |
| Insert | 4,0 | 7,0 | 10,0 | 9,0 | 8,0 | 13,0 | 12,0 | 15,0 | 12,0 | 16,0 | 10,6 |
| Select | 4,0 | 6,0 | 8,0 | 11,0 | 16,0 | 24,0 | 29,0 | 41,0 | 46,0 | 57,0 | 24,2 |
| Burb | 5,0 | 12,0 | 24,0 | 39,0 | 64,0 | 105,0 | 118,0 | 159,0 | 196,0 | 248,0 | 97,0 |
| Quick | 6,0 | 2,0 | 3,0 | 3,0 | 7,0 | 7,0 | 6,0 | 5,0 | 6,0 | 8,0 | 5,3 |
|  | Tiempo en ms | | | | | | | | | |  |

**Comportamiento de los Algoritmos**

**Promedio de tiempos en el rango de 1000 a 10000 datos de los algoritmos**

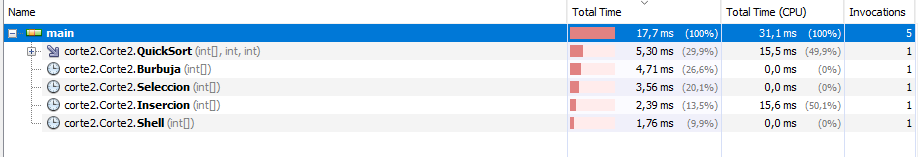
**Comparación entre Algoritmos**

# **Relación entre las Medidas Arrojadas por la Herramienta “Profiling” vs el Medidor por Código “currentTimeMillis();”**

A continuación, se muestra la comparación de gráficos entre la herramienta manual por código y el “**Profiler de Apache Neatbeans”**.

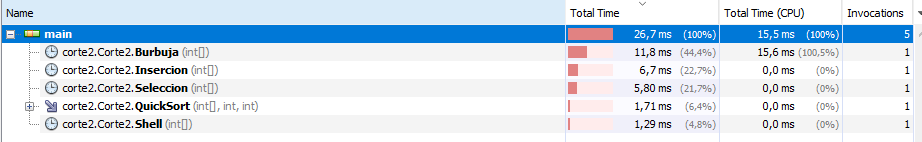
**Gráfico 1000 datos “Código” vs Gráfico 1000 datos “Profiler”**

**Vs**



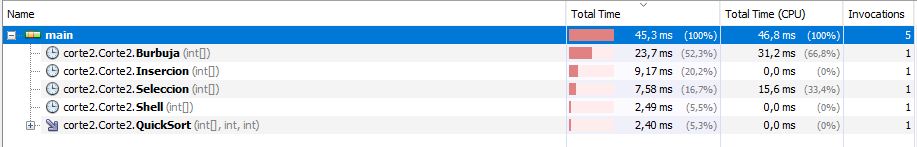
**Gráfico 2000 datos “Código” vs Gráfico 2000 datos “Profiler”**

**Vs**



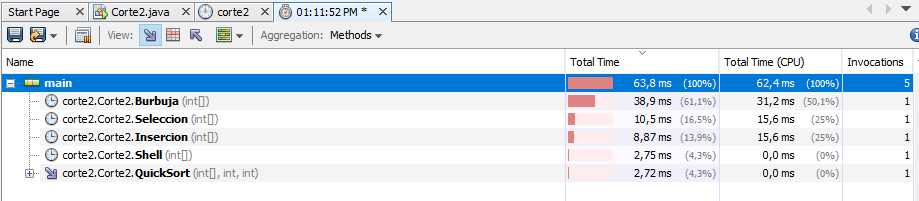
**Gráfico 3000 datos “Código” vs Gráfico 3000 datos “Profiler”**

**Vs**



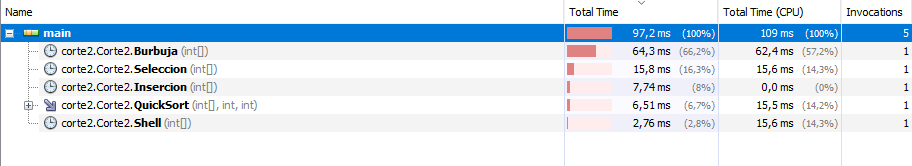
**Gráfico 4000 datos “Código” vs Gráfico 4000 datos “Profiler”**

**Vs**



**Gráfico 5000 datos “Código” vs Gráfico 5000 datos “Profiler”**

**Vs**



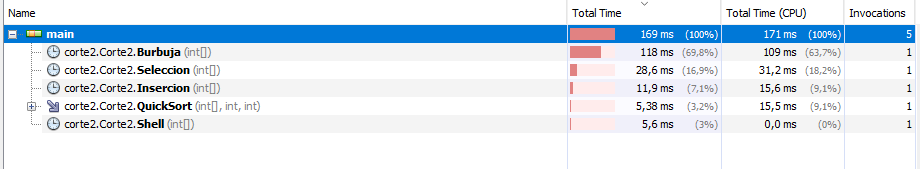
**Gráfico 6000 datos “Código” vs Gráfico 6000 datos “Profiler”**

**Vs**



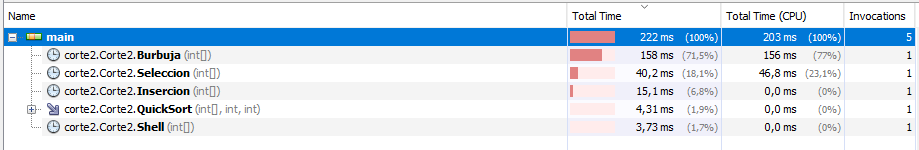
**Gráfico 7000 datos “Código” vs Gráfico 7000 datos “Profiler”**

**Vs**



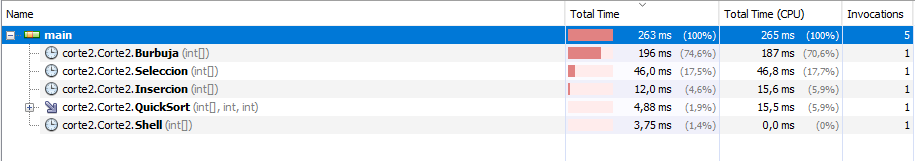
**Gráfico 8000 datos “Código” vs Gráfico 8000 datos “Profiler”**

**Vs**



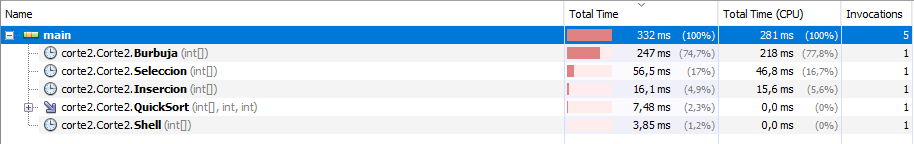
**Gráfico 9000 datos “Código” vs Gráfico 9000 datos “Profiler”**

**Vs**



**Gráfico 10000 datos “Código” vs Gráfico 10000 datos “Profiler”**

**Vs**



# **¿Hay una relación entre los resultados arrojados por el “profiler” y el medidor código? ¿hay similitud?**

**Conclusión**

Finalizando, la comparativa entre el **“Profiler”** y el medidor de código manual **“currentTimeMillis();”**, podemos hacer veracidad en que algunas ocasiones, pero no en mayoría, las mediciones de las dos herramientas arrojan datos equivalentes.

En mayoría, los datos entre estas dos herramientas difieren, pero no es mucho, podríamos decir que tratan de aproximarse lo suficiente para que sea creíble la medición.