UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES.

Centro de Ciencias Básicas.

Departamento de Estadística.

Licenciatura en Matemáticas Aplicadas.

Proyecto 1.

¿Qué Lenguaje de Programación se Ejecuta más Rápido?

Diseño de Experimentos.
Prof. Angélica Hernández Quintero.

Alumnos:
Juan Daniel Medina Jiménez.
Erick Ignacio Rodríguez Juárez.

Domingo 8 de Mayo, 2022.

CONTENIDOS.

1 INTRODUCCIÓN.

1.1 — Motivación —

El desarrollo de programas computacionales (y la construcción de nuevos lenguajes de programación) ha ofrecido poderosas herramientas y técnicas para la extensión de las profesiones mejorando su productividad. Con algunas tareas, para continuar con un ascenso en la eficiencia, se requieren crear y desarrollar tecnologías cuyo objetivo principal sea el de producir programas cada vez más rápidos. Sin embargo, la gran cantidad de lenguajes existentes (cada uno con diferentes implementaciones y funciones únicas) puede hacer que se desvíe la tarea por escribir el programa más eficiente en términos de recursos algorítmicos, y sea reemplazada por la búsqueda de paqueterías y lenguajes que integren las funciones deseadas, y los problemas resueltos. El enfoque inverso también representa una problemática: si el lenguaje en el que se programa para resolver una determinada tarea es ineficiente en términos de recursos computacionales para la implementación del algoritmo escrito, es conveniente migrar a otro lenguaje que realice tal proceso en una cantidad de tiempo menor.

Estos dos problemas son de interés para dos sectores. Primero, las empresas que requieran soluciones técnicas y específicas. Segundo, las personas que realicen investigación individual y el público que necesite implementar funciones para hacer su profesión (y/o su informática personal) mejor adaptada a sus necesidades. El presente trabajo está encaminado a resolver ésta problemática para las personas del segundo sector, dado que en una empresa pueden llegarse a obtener mejores recursos computacionales y solucionar su informática por fuerza bruta o algoritmos menos óptimos.

En la lista siguiente (obtenida de [1]) se incluyen los lenguajes de programación más conocidos y sencillos, cuyos compiladores (o programas de ejecución) son suficientemente ligeros para instalarse en un ordenador promedio casero.

1. `C`.
2. `Python`.
3. `R`.
4. `Ruby`.
5. `Matlab/Octave`.
6. `Rust`.
7. `Java`.
8. `Lua`.

1. `C++`.
2. `JavaScript`.
3. `Golang`
4. `Perl`.

Se seleccionarán para objeto de estudio, los primeros 4 lenguajes: `C`, `Python`, `R`, y `Ruby`, debido a que son de los más conocidos y simples en escritura para funciones del segundo sector demográfico mencionado, consulte [2].

El objetivo es determinar estadísticamente el tiempo promedio en que una misma secuencia de tareas (algoritmo) se logra ejecutar en cada uno de los lenguajes seleccionados, y medir en cuánto difieren en su tiempo promedio de ejecución. Para ello, se someterá a todos los programas escritos en los 4 lenguajes a seguir el mismo algoritmo, a pesar de que (posiblemente) existan formas más eficientes de lograr la tarea propuesta, ya sea por medio de paqueterías adicionales al lenguaje, o bien, por funciones que se ofrecen de forma nativa.

Se tiene conocimiento del estudio comparativo realizado en [3], cuyo objetivo es el de observar la cantidad de energía gastada por cada lenguaje. En tal análisis se realiza la recolección de 10 problemas de programación bien conocidos, escritos en 27 lenguajes de programación. No obstante, examinando el código en [4], se observa que se seleccionaron distintos algoritmos con diferentes órdenes de complejidad para resolver la misma tarea en algunos lenguajes, tales como en la obtención del $n-$ésimo número de Fibonacci en `C`, y `Python`.

El presente trabajo pretende exponer ejemplos más sencillos, pero estandarizados, para evaluar a los lenguajes de programación mencionados. Y para que el presente análisis pueda ser fácilmente replicado por cualquier persona del segundo sector que requiera apreciar diferencias significativas en los tiempos de ejecución. Se establece la licencia de software GPLv3 en todos los archivos de este documento y análisis en [5] con éste propósito.

1.2 — Diseño de Experimentos —

1.2.1 Definición de Diseño Experimental.

Un *experimento* es una sucesión de pasos que cambian las condiciones de operación de un sistema. Su objetivo en el planteamiento es el de determinar el efecto producido por las características variables en el estudio. Son escogidos un conjunto finito de condiciones a las que serán cometidos unos sujetos (u objetos) determinados para medir la respuesta producida por el cambio de condiciones.

Éstas con las características de un diseño experimental.

1. *Unidad Experimental:* Objeto al que se registrará las mediciones de todo el experimento.
2. *Variable Respuesta:* Característica medible (con unidades) de la unidad experimental.
3. *Factores Controlables:* Características del experimento que siempre podrán ser manipulables y que podrán ser manipulables en cualquier otro experimento análogo.
4. *Niveles de Factor Controlable (tratamientos):* Categorías disponibles del factor controlable ($T_1, \ \ldots, \ T_k$).
5. *Factores de Ruido:* Características del experimento que interfieren con la unidad experimental. No necesariamente son las mismas en otros experimentos análogos.
6. *Número de Repeticiones por tratamiento:* Número de veces en que se realiza una medición en el mismo nivel de factor controlable, y en los mismos factores de ruido.
7. *Número de Corridas Experimentales:* Número total de mediciones.
8. *Balanceo:* Un diseño es balanceado si se realizó el mismo número de mediciones en cada nivel de factor controlable.

En algunos diseños se tiene adicionalmente el siguiente concepto.

9. *Factor Bloque:* Factor de ruido que es manipulable en el experimento.
10. *Niveles de Factor Bloque:* Categorías disponibles del factor bloque ($B_1, \ \ldots, \ B_n$).

Se tiene como principio el de determinar el orden de las corridas experimentales de forma *aleatoria*, esto es, que no se tenga elección sobre el orden de presentación de factores de ruido y controlables al momento de realizar la recolección de mediciones. Consulte [6].

1.2.2 Definición de un Diseño Cuadro Latino.

Un *Diseño Cuadro Latino* (DCL) es un diseño experimental lineal con un sólo factor controlable con k tratamientos ($T_1, \ \ldots, \ T_k$), dos factores de bloque, cada uno con k niveles ($A_1, \ \ldots, \ A_k$ y $B_1, \ \ldots, \ B_k$), de tal manera que por cada posible configuración de bloque sólo exista un medición de un factor controlable que no se repita en el mismo Bloque 1, o Bloque 2. Así, la Tabla 1 es un DCL 4×4. Usualmente los tratamientos se enumeran con las letras $A, B, \ \ldots \ Z$ y los niveles de bloque se establecen al inicio del experimento, indicando cuál será la fila y cuál la columna. De modo que la representación final se obtiene en la Tabla 2.

	A_1	A_2	A_3	A_4
B_1	T_4	T_2	T_3	T_1
B_2	T_2	T_4	T_1	T_3
B_3	T_1	T_3	T_4	T_2
B_4	T_3	T_1	T_2	T_4

Tabla 1: Ejemplo de DCL no–estándar.

D	B	C	A
B	D	A	C
A	C	D	B
C	A	B	D

Tabla 2: Representación de DCL del la Tabla 1.

Un DCL es *estándar*, en caso de que la primer fila conste de los tratamientos ordenados $T_1, \ \ldots, \ T_k$. Cualquiera de los DCL puede obtenerse a través de la permutación de filas y columnas de un DCL estándar. La formulación matemática es la siguiente. Sean

$$Y_{ij(h_{ij})} : \Omega \longrightarrow \mathbb{R}, \qquad Y_{ij(h_{ij})} \sim N(\mu + \tau_{h_{ij}} + \gamma_i + \delta_j, \sigma^2), \qquad i,j \leqslant k.$$

donde a μ se le llama *gran media*, $\tau_{h_{ij}}$ es el efecto del h_{ij}–ésimo tratamiento, γ_i es el efecto del i–ésimo nivel del bloque 1, y d_j es el efecto del j–ésimo nivel del bloque 2. Hay que estimar $\{\mu, \tau_{h_{ij}}, \gamma_i, \delta_j\}$ y además se asume que

$$Y_{ij(h_{ij})} = \mu + \tau_{h_{ij}} + \gamma_i + \delta_j + \varepsilon_{ij(h_{ij})}. \tag{1}$$

es decir, $\varepsilon_{ij(h_{ij})} \sim N(0, \sigma^2)$, y es una variable aleatoria llamada *residual*. Para esto, consulte [7]. Las preguntas principales a responder en un DCL son las siguientes. (Se les llama hipótesis porque no se conoce su veracidad y las proposiciones se podrán a prueba de los datos recolectados). Se le llama H_0 a la *hipótesis nula*, y H_a, *hipótesis alternativa*.

Hip. Principal

$$\begin{aligned} H_0 : &\quad \tau_1 = \ \cdots \ = \tau_k = 0, \\ H_a : &\quad \exists h \ : \ \tau_h \neq 0. \end{aligned} \tag{2}$$

La cual, es probada con el estimador $F_{trat} = CM_{trat}/CM_E$ de la tabla ANOVA de la sección 9.3. Y H_0 significa que no hay diferencia entre tratamientos.

Hip. Secundaria 1

$$\begin{aligned} H_0 : &\quad \gamma_1 = \ \cdots \ = \gamma_k = 0, \\ H_a : &\quad \exists i \ : \ \gamma_i \neq 0. \end{aligned} \tag{3}$$

La cual, es probada con el estimador $F_{bloq1} = CM_{bloq1}/CM_E$ de la tabla ANOVA. Entonces, H_0 significa que los niveles del Bloque 1 no afectan al experimento.

Hip. Secundaria 2

$$\begin{aligned} H_0 : &\quad \delta_1 = \ \cdots \ = \delta_k = 0, \\ H_a : &\quad \exists j \ : \ \delta_j \neq 0. \end{aligned} \tag{4}$$

La cual, es probada con el estimador $F_{bloq2} = CM_{bloq2}/CM_E$ de la tabla ANOVA. Además, H_0 significa que los niveles de Bloque 2 no afectan al experimento.

Debido a que sólo es posible medir una repetición del factor controlable en cada configuración de bloque, el DCL es trivialmente balanceado. Un diseño DBCA es un modelo con una sólo factor de bloque. Un DCA es un modelo sin factores bloque. El análisis estadístico es completamente análogo al DCL.

2 *DISEÑO EXPERIMENTAL.*

OBJETIVO: Determinar la existencia de un lenguaje de programación cuyos archivos de ejecución logren completar una tarea asignada, en un tiempo promedio diferente al resto de lenguajes. O en su defecto, encontrar si todos los programas se ejecutan estadísticamente en el mismo tiempo promedio.

Se tiene el siguiente planteamiento dadas las características del experimento.

1. *Unidad experimental:* Archivo de texto escrito en algún lenguaje de programación.
2. *Variable respuesta:* Tiempo de ejecución del programa en el compilador disponible del lenguaje.
3. *Factor controlable:* Lenguaje de programación.
4. *Niveles de factor controlable:*
   (a) `C`. (b) `Python`. (c) `R`. (d) `Ruby`.
5. *Factores de ruido:*
   (a) Tarea asignada a resolver, es decir, el algoritmo programado.
   (b) La cantidad de procesos en segundo plano que realiza el equipo de cómputo de pruebas para funcionar.
   (c) El equipo de cómputo en dónde se ejecutan los programas.
      i. Procesador. ii. Memoria RAM. iii. Tarjeta Gráfica.
   (d) Las versiones de los programas usados.
   (e) Programa de medición del tiempo en milisegundos del ordenador.
   (f) Temperatura del procesador al momento de las ejecuciones.
   (g) Emulador de terminal (aplicación) que despliega y ejecuta las pruebas.

De entre los factores de ruido mencionados anteriormente, se dispondrá del control en el experimento de los factores 5.(a) y 5.(b), dado que son usualmente los que los usuarios del segundo sector demográfico de la sección 1.1 puede controlar. Así, se tiene un Diseño Cuadro Latino.

6. *Niveles de Bloque 1 (algoritmo):*
   (a) Obtención del $n-$ésimo número de Fibonacci. Que es de Orden $O(n)$.
   (b) Máximo Común Divisor de dos números dados. Cuyo Orden de complejidad es de $O(n \log n)$.
   (c) Aproximación del cálculo de π. Su Orden de complejidad es de $O(n2^n)$.
   (d) Ordenación burbuja. El orden de complejidad es de $O(n^2)$.

   La justificación del orden de complejidad de éstos niveles se presenta en la sección 6.1.
7. *Niveles de Bloque 2 (Procesos en segundo plano):*
   (a) Sin procesos.
   (b) Editor de Texto.
   (a) Navegador abierto en una página web.
   (b) Navegador reproduciendo video en página web.

   La elección de éstos niveles se debe a que son los que usualmente se corren en segundo plano para investigar el cómo programar en lenguajes desconocidos.
8. *Número de repeticiones por tratamiento:* 1.
9. *Número de corridas experimentales:* 16.

El diagrama experimental es representado en la Figura 1.

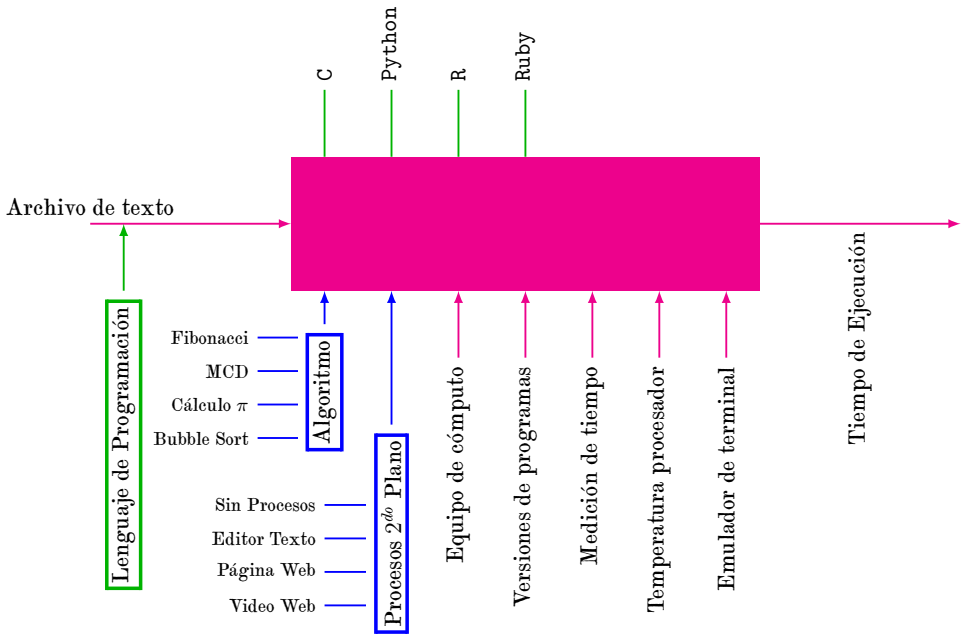


Figura 1: Diagrama que representa el experimento propuesto.

Y se tiene la siguiente tabla del orden de factores bloque.

	Sin Procesos	Editor de Texto	Navegador en Pág. Web	Navegador Video Web.
Fibinacci				
MCD				
Cálculo de π				
Bubble Sort				

Tabla 3: Diseño de Bloques DCL.

3 *FACTORES DE RUIDO.*

3.1 — Ordenador de Ejecución —

3.1.1 Hardware.

`Modelo` `Dell Inspiron 11 3000 11-3147.`

`Procesador` `Intel®Pentium(R) CPU N3540 @ 2.16GHz × 4 (Quad-core).`

`Memoria RAM` `~4GB of 1333 MHz DDR3L.`

`Memoria` `500GB 5400 rpm 2.5" SATA II Hard Drive.`

`Tarjeta Gráfica` `Intel Atom Processor Z36xxx/Z37xxx Series Graphics & Display.`

El resto de características físicas del equipo de cómputo pueden encontrarse en [8] y [9].

3.1.2 Software.

`Sistema Operativo` `(GNU/Linux) Ubuntu 20.04.4 LTS.`

`Arquitectura` `x86-64 (i.e., 64bits).`

`Kernel` `5.13.0-40-generic.`

`Shell` `Bash 5.0.17.`

`Emulador de Terminal` `Kitty 0.22.2.`

El equipo de cómputo tenía `44min` encendido al realizar la primer prueba y las subsecuentes, como lo indica la Figura 2. Los procesos por defecto del sistema operativo son enunciados en la sección 6.2.

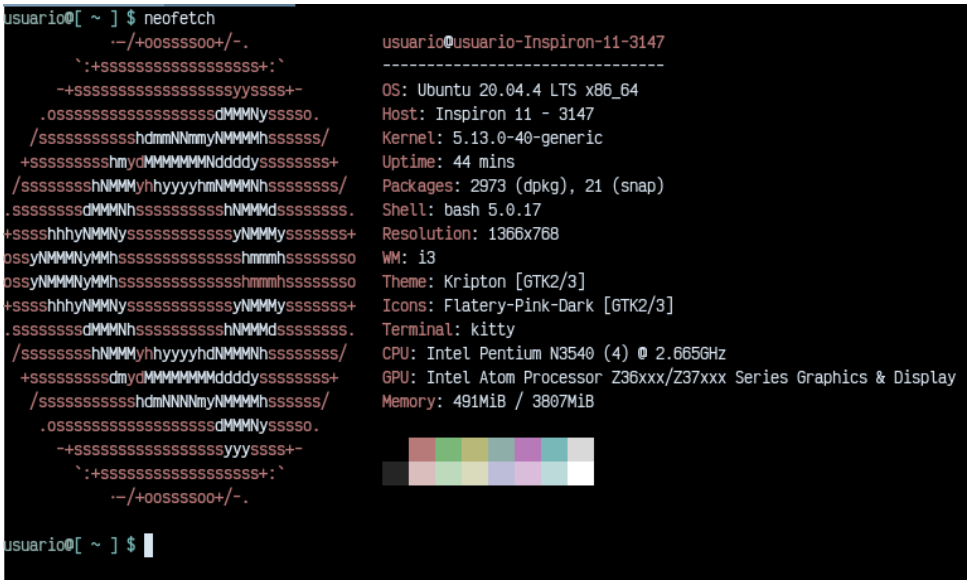


Figura 2: Captura de pantalla de las características del equipo.

3.2 — Medición del Tiempo —

Para el registro de tiempos de ejecución, se hará uso del comando `time`, del sistema operativo Ubuntu en la versión `1.7` (así como se puede comprobar en la Figura 3), cuya documentación se encuentra en [10].

Figura 3: Captura de pantalla desde la línea de comandos para ver la versión del comando `time`.

En cada proceso ejecutado por el sistema operativo, hay 3 eventos importantes del procesador que pueden medirse (en el tiempo) con el programa `time`.

1. `[Real Time]` Es la diferencia de registros obtenidos del reloj interno integrado del sistema operativo. El primer registro se realiza al empezar el proceso en terminal. El segundo registro se realiza cuando el proceso ha finalizado dentro del emulador de terminal, incluyendo así, el tiempo de posibles Input/Output desplegados por el programa.
2. `[User Time]` Cantidad de tiempo en que el CPU usa en el modo usuario en el proceso. También se le conoce como modo `no-privilegiado`. En este modo, ciertas funciones no están disponibles por condiciones de hardware.
3. `[System Time]` Cantidad de tiempo en que el CPU usa dentro del kernel del sistema operativo para ejecutar un proceso del programa. También se conoce como modo `privilegiado`. En este modo, se puede realizar cualquier instrucción que la arquitectura del sistema operativo permita. Sin embargo, no siempre es necesario acceder a éste modo.

Consulte [11] y [12].

Se centrará la atención del experimento únicamente para medir el tiempo Real, es decir, el que experimentará el usuario final con su programa, debido a que se desea medir el impacto del lenguaje de programación en el segundo sector demográfico descrito en la sección 1.1.

Se despliega un ejemplo del comando en a Figura 4, para observar que la incertidumbre de medición es de ± 0.5 milisegundos.

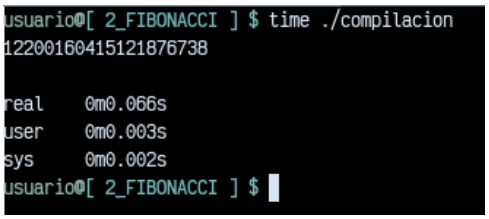


Figura 4: Ejemplo de ejecución del comando `time`, midiendo una compilación en el lenguaje `C`.

4 *ALEATORIZACIÓN.*

Como se indicó en la sección 1.2.1, se requiere tener una elección aleatoria sobre el orden de las corridas experimentales. Dado que, sólo se tendrá un factor controlable por cada una de las configuraciones de los bloques, entonces se asignarán los factores controlables aleatoriamente a cada celda de la matriz de la tabla de Diseño Cuadro Latino.

Se hace uso del programa escrito en [13] (adaptado en el archivo `ANALISIS_ESTADISTICO/1_ALEATORIZACION/GENERACION_DISENO.c`, en [5]), y se ejecuta con la siguiente instrucción de `Bash`.

```
g++ GENERACION_DISENO.c -o ./compilacion && ./compilacion > output.txt
```

Y se tiene el siguiente resultado

```
[Ruby, Python, R, C]
[R, Ruby, C, Python]
[Python, C, Ruby, R]
[C, R, Python, Ruby]
```

Así, se escribe el diseño en la Tabla 3, y se coloca en la base de datos en el archivo `ANALISIS_ESTADISTICO/2_DCL/dcl_2.txt` del repositorio [5].

	Fibonacci	MCD	Cálculo de π	Bubble Sort
Sin procesos	Ruby	Python	R	C
Editor de Texto	R	Ruby	C	Python
Navegador en Pág. Web	Python	C	Ruby	R
Navegador en Video Web	C	R	Python	Ruby

Tabla 4: Aleatorización del diseño.

Se tiene un DCL no–estándar.

5 *NIVELES DE FACTOR CONTROLABLE.*

Éstas, fueron las versiones de los compiladores, en que se ejecutaron los distintos lenguajes de programación.

`C` El programa en el sistema operativo es `g++ 9.4.0`.

`Python` `Python 3.8.10`.

`R` `Rscript front-end version 4.1.2 (2021-11-01) -- "Bird Hippie" x86_64-pc-linux-gnu`.

`Ruby` `ruby 2.7.0p0 (2019-12-25 revision 647ee6f091) [x86_64-linux-gnu]`

6 CONFIGURACIÓN DE BLOQUES.

6.1 — Algoritmos Seleccionados para Test de Rapidez —

Se esbozan los 4 algoritmos utilizados en el diseño experimental, cuyos archivos están licenciados con GPLv3, y pueden encontrarse en el repositorio [5], en la carpeta `PROGRAMAS/`. Estos algoritmos fueron escritos en cada lenguaje usando su documentación básica.

6.1.1 Números de Fibonacci.

```
Funcion Fibonacci (n) is
    Data: f_actual = 0, f_siguiente = 1, aux = 0
    for i ⩽ n do
        aux = f_actual + f_siguiente;
        f_actual = f_siguiente;
        f_siguiente = aux ;                                  /* orden de O(1) */
    end
end
Imprimir Fibonacci(46);
```

Este algoritmo es de orden $O(n)$, debido a la cantidad de ciclos requeridos de ejecución (n). Está inspirado en el repositorio [4].

6.1.2 Máximo Común Divisor.

```
Function Max_Com_Div (primer_num , segundo_num) is
    residuo = mod(primer_num,segundo_num);
    while residuo ≠ 0 do
        primer_num = segundo_num;
        segundo_num = residuo;
        residuo = mod(primer_num,segundo_num) ;        /* orden de O(log n) */
    end
end
Max_Com_Div (6983776800,5587021440);
```

Este algoritmo es de orden $O(n \log n)$, puesto que depende de lo largo que sea el número se efectuará el algoritmo de Euclides `mod` que es de complejidad $O(\log n)$. Está inspirado en el repositorio [4].

6.1.3 Cálculo de Pi.

```
Data: a = 0 , b = 1 , n = 100,000
suma = 0;
h = (b-a) /n;
for i ⩽ n do
    suma = suma + sqrt(1 - (a+i*h) * (a+i*h)) ;        /* orden de O(2^n) */
end
suma = h * [(b-a)/2 + suma];
Imprimir suma;
```

Este algoritmo es de orden $O(n2^n)$, puesto que la función `sqrt()` es de orden $O(2^n)$. Y consiste en la

obtención de la integral de la función $f(x) = \sqrt{1 - x^2}$, por el método numérico de la Regla del Trapecio.

6.1.4 *Bubble Sort.*

```
Data: a = [564, 550, 34, ..., 165]
;
Function Ordenamiento Burbuja (a) is
    for j ⩽ length(a) do
        for i ⩽ j do
            if a[i] < a[j] then
                aux = a[i];
                a[i] = a[i+1];
                a[i+1] = aux ;                          /* orden de O(1) */
            end
        end
    end
    return a;
end
```

Este algoritmo es de orden $O(n^2)$, por los dos procesos `for`. La cadena inicial `a` tiene los $1,000$ números naturales desordenados, y fue generada en python. Véase el código fuente en [5], para la revisión de la cadena introducida. Está inspirado en el repositorio [4].

6.2 — *Procesos en Segundo Plano* —

Las siguientes capturas de pantalla, junto con el resto de detalles de los programas en ejecución, pueden consultarse en [5], en la carpeta `DOCUMENTO/IMAGENES/6`.

Sin procesos Se tiene sólo la ejecución del programa más los procesos necesarios para que el sistema operativo funcione (en la Figura 5), junto con la terminal donde se ejecutan los "scripts" de la sección 6.1.

Figura 5: Procesadores trabajando únicamente con los procesos por defecto del SO.

Editor de Texto Se elige SublimeText (TM), por ser uno de los editores de texto más utilizados en ordenadores con relativos bajos recursos (consulte [14]). Los procesadores presentaron el siguiente comportamiento.

Figura 6: Procesadores trabajando con Sublime Text.

Página Web El ordenador de ejecución tenía las siguientes características en la ejecución de Firefox con la página de Internet `https://www.office.com/` abierta.

Figura 7: Procesadores con el navegador abierto en la página descrita.

Video Web Se elige la plataforma de Youtube en el video `https://www.youtube.com/watch?v=grBFMP3HDZA`. El cuál, en el ordenador de ejecución, tenía las siguientes características.

Figura 8: Procesadores ejecutando el video descrito en Firefox.

7 METODOLOGÍA DEL EXPERIMENTO.

Como indica la Figura 2, el ordenador estuvo $44min$ encendido al realizar la primer prueba de la Tabla 4. Se realiza cada uno de los registros de tiempos de ejecución de forma aleatoria.

Para minimizar el nivel de imprecisión del tiempo registrado computacionalmente, se realizan 6 repeticiones para cada una de las 16 mediciones, y se considerará como valor real su promedio aritmético. Los datos completos pueden consultarse en [5], en las carpetas `PROGRAMAS/*/EJECUCION_DE_PROGRAMAS/`. A continuación se escriben los comandos utilizados en cada programa.

7.1 — *C* —

Se realiza la compilación de cada archivo fuente `archivo.c`, mediante el programa `g++`.

```
g++ archivo.c -o ./compilacion
```

Dado que el experimento consiste en medir el tiempo que toma el archivo ejecutable `./compilacion` en completar la tarea deseada, se usa el programa `time`. Y se registra el tiempo generado por el siguiente comando.

```
time ./compilacion
```

El resto de lenguajes de programación no poseen archivos ejecutables.

7.2 — *Python* —

Se realiza el registro con el comando `time`, de la ejecución del programa `python3`.

```
time python3 archivo.py
```

7.3 — *R* —

El comando que registra los tiempos de ejecución es el siguiente.

```
time Rscript archivo.r
```

7.4 — *Ruby* —

Se registran los tiempos de medición con el siguiente comando.

```
time ruby archivo.rb
```

8 *RESULTADOS.*

Se registra en la Tabla 5, los valores registrados con los promedios aritméticos de 6 tiempos de ejecución de cada programa en las condiciones de cada bloque. La base de datos completa puede consultarse en [5], dentro del archivo `ANALISIS_ESTADISTICO/2_DCL/dcl_2.txt`.

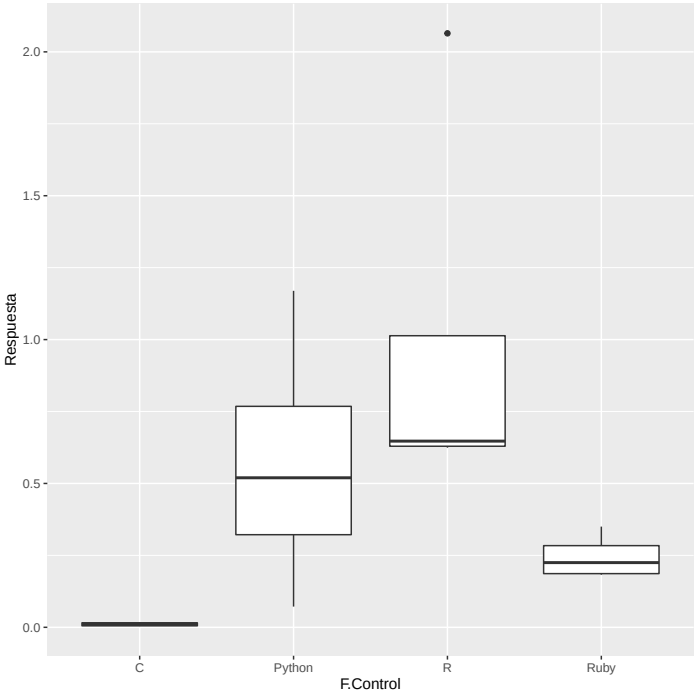
	Sin Procesos	Editor de Texto	Navegador en Pág. Web	Navegador Video Web.
Fibinacci	Ruby(0.1878333)	Python(0.072)	R(0.631)	C(0.004166667)
MCD	R(0.6243333)	Ruby(0.1828333)	C(0.0055)	Python(0.6341667)
Cálculo de `Aprox_Pi`	Python(1.1695)	C(0.015)	Ruby(0.2618333)	R(0.6631667)
Bubble Sort	C(0.01783333)	R(2.064333)	Python(0.4051667)	Ruby(0.35)

Tabla 5: Aleatorizaión del diseño.

9 *ANÁLISIS ESTADÍSTICO.*

Los resultados se evaluarán a un nivel de confianza del $1 - \alpha = 95\%$, es decir, se toma $\alpha = 0.05$. El análisis completo puede consultarse en [5] en la carpeta `ANALISIS_ESTADISTICO/*`.

9.1 — Diagrama de Cajas Simultáneas —



Se concluye que

1. `Python`, `R` y `Ruby` presentan sesgo a la derecha.
2. `R` y `Ruby` comparten más del 75% de sus datos con `Python`.
3. `Python` presenta una mayor variabilidad que el resto de lenguajes.

9.2 — Supuestos del Modelo de Regresión Lineal —

Supuesto	Test	Valor–p	Conclusión
Normalidad	Shapiro–Wilk	0.3323	Se cumple el supuesto de normalidad
Homocedasticidad	Levene	0.4367	Se cumple el supuesto de homocedasticidad
Independencia	Durbin–Watson	0.6498	Se cumple el supuesto de independencia

9.3 — ANOVA —

Dada la verificación de los supuestos, se realiza la Tabla ANOVA.

F. Variabilidad	Grados de Libertad	Suma de Cuadrados	Cuadrados Medios	F	Valor–p
F.Control(Lenguaje)	3	2.18778	0.72926	2.9085	0.1231
Bloque1(Algoritmo)	3	0.52841	0.17614	0.7025	0.5843
Bloque2(Procesos 2^{do} plano)	3	0.14793	0.04931	0.1967	0.8950
Residuals	6	1.50440	0.25073		
Total	12	4.36852			

Se observa que la columna `[Valor-p]` contiene los percentiles de las pruebas de hipótesis (2), (3) y (4), respectivamente. Las cuales adaptadas al modelo estadístico son las siguientes.

Hip. Principal Se re-escribe (2).

H_0 : Todos los lenguajes de programación estudiados tienen el mismo tiempo de ejecución promedio.
H_a : Hay al menos un lenguaje analizado que tiene un tiempo promedio de ejecución diferente.

Hip. Secundaria 1 Se re-escribe (3).

H_0 : El tiempo de ejecución promedio no es afectado por el algoritmo programado.
H_a : Hay al menos un algoritmo que produce un tiempo de ejecución promedio diferente.

Se tiene que *valor–p*$= 0.5843 > \alpha$, entonces no se rechaza H_0 (es decir, el tiempo de ejecución promedio no es afectado por la complejidad algorítmica del programa). Se remueve ésta fuente de variabilidad y se realiza nuevamente el análisis en la sección 9.4.

Hip. Secundaria 2 Se re-escribe (4).

H_0 : El tiempo de ejecución promedio no es afectado por la cantidad de procesos en segundo plano.
H_a : Hay al menos un proceso en segundo plano que produce un tiempo de ejecución promedio diferente.

9.4 — Reducción de Modelo a un DBCA —

Se redujo el modelo a la siguiente formulación de la ecuación (1) $Y_{ijh} = \mu + \tau_{h_j} + \delta_j + \varepsilon_{ij(h_{ij})}$, que es un DBCA. Y por tanto, se construye de nuevo la Tabla ANOVA.

F. Variabilidad	Grados de Libertad	Suma de Cuadrados	Cuadrados Medios	F	Valor–p
F.Control(Lenguaje)	3	2.18778	0.72926	3.9722	0.04677
Bloque2(Procesos 2^{do} Plano)	3	0.14793	0.04931	0.2183	0.88126
Residuals	9	2.03281	0.22587		
Total	12	4.36852			

Hip. Principal

H_0 : Todos los lenguajes de programación estudiados tienen el mismo tiempo de ejecución promedio.
H_a : Hay al menos un lenguaje analizado que tiene un tiempo promedio de ejecución diferente.

Hip. Secundaria 2

H_0 : El tiempo de ejecución promedio no es afectado por la cantidad de procesos en segundo plano.
H_a : Hay al menos un proceso en segundo plano que produce un tiempo de ejecución promedio diferente.

Se tiene que $valor - p = 0.88126 > \alpha$, entonces no se rechaza H_0. Por tanto, los procesos en segundo plano no afectan a los tiempos promedio de ejecución. Se reduce el modelo a un DCA en la sección 9.5.

9.5 — Reducción a un DCA —

Se construye la tabla ANOVA. Finalmente, se contesta la Hipótesis Principal.

F. Variabilidad	Grados de Libertad	Suma de Cuadrados	Cuadrados Medios	F	Valor$-p$
F.Control(Lenguaje)	3	2.1878	0.72926	4.0129	0.03428
Residuals	12	2.1807	0.18173		
Total	12	4.36852			

Hip. Principal

H_0 : Todos los lenguajes de programación estudiados tienen el mismo tiempo de ejecución promedio.
H_a : Hay al menos un lenguaje analizado que tiene un tiempo promedio de ejecución diferente.

Y en este caso tenemos que $valor - p = 0.03 < \alpha$, por lo que se rechaza H_0. Así, existe un lenguaje de programación que tiene un tiempo de ejecución promedio diferente al resto. Se observará cuál es diferente en la sección 9.6.

9.6 — Tratamientos Diferentes —

Se realiza el test de Fisher Least Significant Difference (LSD), y se observa lo siguiente, (véase [5]).

```
least Significant Difference: 0.8663824
Treatments with the same letter are not significantly different.
       Respuesta groups
R      0.9957082      a
Python 0.5702084     ab
Ruby   0.2456250     ab
C      0.0106250      b
```

10 CONCLUSIONES.

1. Los algoritmos elegidos no influyeron de forma significativa en los tiempos promedio de ejecución.
2. Los distintos procesos en segundo plano en el ordenador no influyeron de forma significativa en los tiempos promedio de ejecución.
3. `R`, `Python` y `Ruby` no son significativamente diferentes en su tiempo promedio de ejecución.
4. `C`, `Python` y `Ruby` no son significativamente diferentes en su tiempo promedio de ejecución.
5. `C` y `R` son significativamente diferentes en su tiempo de ejecución promedio.

   Observamos que en el código fuente de `R`, (de la página [15]), `R` mayormente está escrito en `C`. Indicando que, el proceso de traducción de instrucciones a `C` puede volver significativamente más lento los procesos de ejecución, que directamente, podrían realizarse en nativamente `C`.

11 FUTURA INVESTIGACIÓN.

1. Se alienta a la implementación futura de un diseño experimental que permita realizar la comparación entre algoritmos programados, y que se pueda realizar la ejecución de algoritmos a distintos "inputs", sin que se pierda la interpretación de que las ejecuciones provienen del mismo algoritmo. Es decir, que sea posible diseñar varios niveles, dentro de los mismos niveles de factor. Note que en la sección 6.1 se establecieron las ejecuciones para un parámetro fijo. En el caso de `Fibonacci`, $n = 46$. En el de `MCD`,

se tiene $n, m = 6983776800, 5587021440$. En el `Cálculo de Pi`, se tiene $n = 100,000$. Y en el `Ordenamiento Burbuja` se escogió aleatoriamente un "array" con los primeros $1,000$ números naturales desordenados.

2. Incluir un experimento con algoritmos que requieran bibliotecas de aleatorización (*i.e.* medir el tiempo promedio de ejecución de creación de números pseudo-aleatorios en cada lenguaje).

12 REFERENCIAS.

1 Brian Eastwood. The 10 most popular programming languages, 2020. https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/.

2 Taylor Karl. 7 programming languages for beginner developers, 2021. https://unitedtraining.com/resources/blog/7-programming-languages-for-beginner-developers.

3 Pereira, Couto, Ribeiro, Rua, Cunha, Fernandes, and Saraiva. Ranking programming languages by energy efficiency. *Elsevier*, 2021.

4 Marco Couto and Rui Pereira. Rosseta examples, 30 Nov 2020. https://github.com/greensoftwarelab/RosettaExamples.

5 ZeraujKcire (Erick Juárez). Festest executable language, 2022. https://github.com/ZeraujKcire/FASTEST_EXECUTABLE_LANGUAGE-.

6 De la Vara Salazar Román Gutiérrez Pulido Humberto. *Análisis y diseño de experimentos*. McGraw Hill, 2a edition, 2008. Págs. 7-9.

7 Nancy Reid D.R. Cox. *The Theory of the Design of Experiments*. Monographs on statistics and applied probability 86. Chapman Hall CRC, 1 edition, 2000. Capítulo 4.1.

8 BH Photo Video Audio. Dell inspiron 11 3000 11 - 3147 11.6" multi-touch convertible laptop computer, 2022. https://www.bhphotovideo.com/c/product/1123115-REG/dell_11_3147_inspiron_11_n3530_4gb_500gb_windows_8_1_11_6_silver.html.

9 Dell Technologies. Soporte para inspiron 3147, 2015. https://dl.dell.com/manuals/all-products/esuprt_laptop/esuprt_inspiron_laptop/inspiron-11-3147-laptop_reference%20guide_es-mx.pdf.

10 David MacKenzie. Manual page time command. https://www.commandlinux.com/man-page/man1/time.1.html.

11 GCeasy Team. User, sys, and real, which time to use?, 2016. https://blog.gceasy.io/2016/04/06/gc-logging-user-sys-real-which-time-to-use/.

12 Wikipedia. Modos de operación de la unidad central de procesamiento, 2021. https://es.wikipedia.org/wiki/Modos_de_operaci%C3%B3n_de_la_unidad_central_de_procesamiento.

13 Rosseta Code. Random latin squares, 2022. https://rosettacode.org/wiki/Random_Latin_squares#C.

14 Brian Jackson. 13 best text editors to speed up your workflow, 2022. https://kinsta.com/blog/best-text-editors/.

15 wch. r-source, 1997. https://github.com/wch/r-source/tree/trunk/m4.