# Tarea 3
# Simulación de Sistemas

Beatriz Alejandra García Ramos

A 28 de Agosto de 2017

## 1 Teoría de colas

La teoría de colas estudia el tiempo de espera y en este caso se debe observar si existe una diferencia entre ejecutar nuestro código utilizando de uno hasta los núcleos deseados (o existentes) en nuestra computadora con el fin de hacer un análisis para identificar si éste tiempo tiene una diferencia significativa o no.

## 2 Tareas

- Examinar las diferencias en los tiempos de ejecución variando el número de núcleos asignados.
- Apoyarse en visualizaciones para argumentar las posibles causas en las diferencias de los tiempos de ejecución.
- Aplicar pruebas estadísticas para verificar si las diferencias en los tiempos de ejecución son significativas.

## 3 Solución

De manera lógica se piensa que si nuestro código toma más núcleos para poder ejecutarse debe tener un tiempo de espera menor cada vez que los núcleos van aumentando, pero no necesariamente puede ocurrir así si las tareas que se le asignan pueden llegar a ser complicadas, incluso tener más núcleos en ocasiones podría afectar, como estamos viendo el tiempo que se tarda en encontrar los números primos decidí verificarlos desde el número mil hasta el número tres mil con diez réplicas ya que son datos suficientes para tener una buena muestra de nuestra población.

### 3.1 Tarea base

Para tener la posibilidad de visualizar lo que sucede cada vez que se aumentan los núcleos decidí crear un **for** que me permitiera recorrer el código una y otra vez hasta completar los núcleos disponibles en mi computadora, que son cuatro. Además, para ir guardando cada réplica de cada ordenamiento en cada núcleo creé nuevas matrices que me permitieran guardar la información una vez se completara un paso de **for**, dado que se guardan por ordenamiento lo que hice fue juntar los tres distintos con sus réplicas dependiendo de sus núcleos en una nueva matriz *obs.nucleos.v*, también decidí obtener los promedios de las réplicas de cada ordenamiento en los distintos núcleos dado que es mucho más fácil identificar la diferencia entre doce datos a notar la diferencia entre ciento veinte datos que se deben comparar entre sí si no se obtienen los promedios, el vector que creé es *tiempo.prom*.
Y para lograr una mejor visualización de las comparaciones y diferencias entre las distintas observaciones con sus respectivos núcleos construí dos matrices en donde se pueden identificar mejor los tiempos de ejecución, llamadas *Observaciones* y *Promedio*.

| Nucleos | Ordenamiento | TiempoPromedio |
|---|---|---|
| 1 | 1 | 1.354 |
| 1 | 2 | 1.356 |
| 1 | 3 | 1.368 |
| 2 | 1 | 1.013 |
| 2 | 2 | 1.002 |
| 2 | 3 | 0.977 |
| 3 | 1 | 1.003 |
| 3 | 2 | 0.996 |
| 3 | 3 | 0.987 |
| 4 | 1 | 1.013 |
| 4 | 2 | 0.999 |
| 4 | 3 | 0.992 |

Figure 1: Tiempo promedio de ordenamientos con sus respectivos núcleos

Como podemos observar en la matriz de la Figura 1 los tiempos varían de 0.9 a 1.4, lo cual pareciera no tener gran diferencia en tiempo, pero si se observan los tiempos para el núcleo 1 con sus distintos ordenamientos y los comparamos con el núcleo 3 con sus ordenamientos, podemos identificar un cambio de un poco más de tres dédimas, lo cual en tiempos de ejecución para un programa si tiene una diferencia importante.

## 3.2 Reto 1

Una vez creadas las distintas matrices se puede lograr ver que sí existen diferecias sigificativas comparando los núcleos que se utilizan con los tiempos promedio de ejecución y también con los tiempos de ejecución de todas las observaciones, lo cual podemos observar en la Figura 2.

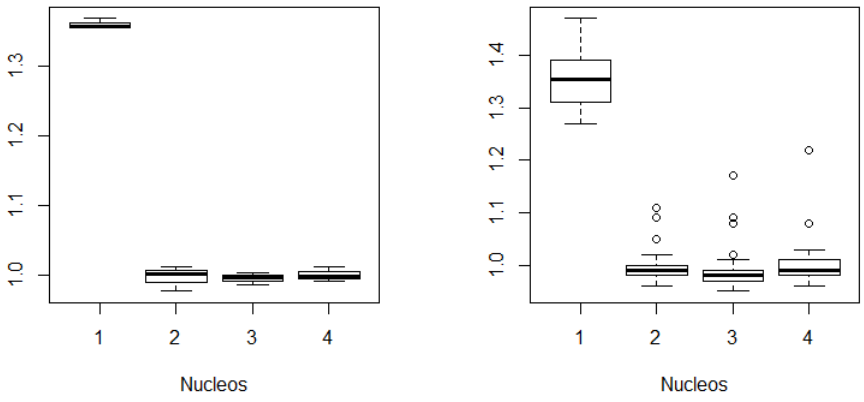


Figure 2: Tiempo promedio de ordenamientos y Tiempo de ejecución de las observaciones con sus respectivos núcleos

Pero si comparamos los tiempos con respecto a los distintos ordenamientos obtenemos los datos de gráfica de la Figura 3, los cuales nos hacen creer que no existe una gran diferencia sobre todo cuando el tiempo que se toma es el tiempo promedio. Lo cual nos indica que no importa el ordenamiento

que estemos utilizando, lo que va a lograr que haya una diferencia significativa va a ser el número de núcleos que se utiliza para la ejecución del código.

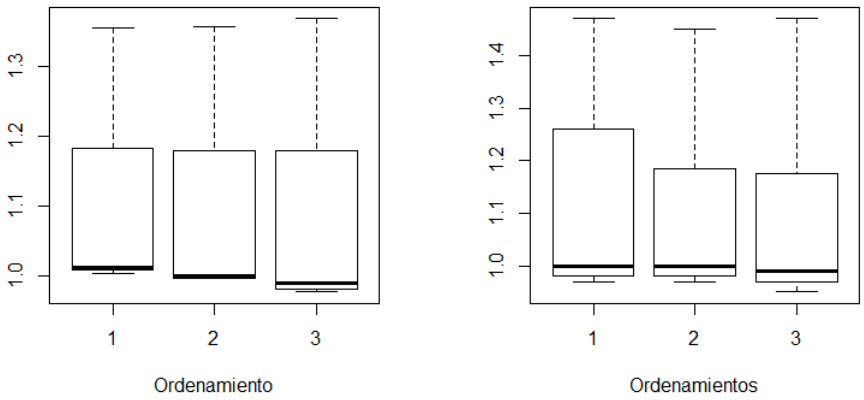


Figure 3: Tiempo promedio de ordenamientos y Tiempo de ejecución de las observaciones con sus respectivos ordenamientos

Además de la visualización de una *boxplot* de los tiempos con respecto a los núcleos podemos apoyarnos con una *barplot* donde se nos muestre una variación en el tiempo promedio de ejecución dada en la Figura 4, donde cada tres barras es un núcleo distinto.

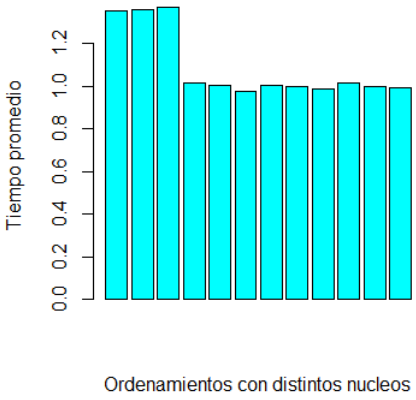


Figure 4: Tiempo promedio de ordenamientos

Tal vez no hay mucha diferencia entre los núcleos 2, 3 y 4 pero el núcleo 1 si tiene una gran variación con respecto a los otros tres.
Mi idea es que los núcleos son quienes afectan dado que son quienes pueden repartirse el trabajo de ejecución y si son más de uno el tiempo de ejecución sería menor, pero lo ideal es solamente tres de ellos ya que si se utilizaran más el tiempo de espera de alguno de ellos sería mayor y eso alentaría el paso del código, además se observa un crecimiento ligero cuando se toman cuatro de ellos y pienso que esto sucede debido a que un núcleo no puede comenzar a procesar hasta que los otros terminen su proceso.

### 3.3 Reto 2

Para este reto utilicé la prueba de Kruskal-Wallis ya que nuestro experimento cuenta con todos los requerimientos de ésta prueba y además está disponible en R.
Lo que se quiere lograr con ésta prueba es saber si las distribuciones de probabilidad entre los núcleos son iguales. Dado que nuestras observaciones no se distribuyen normalmente y además cumplen que son de tamaño mayor a cinco entonces podemos afirmar que la prueba es aceptable para nuestro caso.
Ahora, tendremos que nuestra hipótesis nula es $H_o$: "Las muestras tienen igual distribución de probabilidad" y nuestra hipótesis alternativa $H_1$: "sus distribuciones son diferentes".
Una vez que tenemos todo en orden haciendo la prueba de Kruskal-Wallis se obtienen los datos de la Figura 5.

```
        Kruskal-Wallis rank sum test
data:  nucleo1 by nucleo2
Kruskal-Wallis chi-squared = 8.9409, df = 11, p-value = 0.6274

        Kruskal-Wallis rank sum test
data:  nucleo1 by nucleo3
Kruskal-Wallis chi-squared = 12.354, df = 13, p-value = 0.4989

        Kruskal-Wallis rank sum test
data:  nucleo1 by nucleo4
Kruskal-Wallis chi-squared = 7.2543, df = 11, p-value = 0.7781

        Kruskal-Wallis rank sum test
data:  nucleo2 by nucleo3
Kruskal-Wallis chi-squared = 15.921, df = 13, p-value = 0.2534

        Kruskal-Wallis rank sum test
data:  nucleo2 by nucleo4
Kruskal-Wallis chi-squared = 19.969, df = 11, p-value = 0.04577

        Kruskal-Wallis rank sum test
data:  nucleo3 by nucleo4
Kruskal-Wallis chi-squared = 14.889, df = 11, p-value = 0.1876
```

Figure 5: Pruebas de Kruskal-Wallis en R

Dado que la prueba por lo general utiliza que las distribuciones sean iguales con un nivel de significancia del cinco por ciento utilizaré ésto para saber si se rechaza o no la hipótesis nula comparándolo con el p-valor. Si el p-valor es menor a 0.05 se rechazará, si no lo es entonces la tomaremos como verdadera.
Como podemos observar todos los p-valores son mayores a 0.05 excepto aquel en donde se comparan el nucleo2 con el nucleo4, por lo tanto podemos decir que esas dos muestras se distribuyen de manera diferente ya que rechazamos nuestra hipótesis nula, pero el resto de las pruebas nos indica que las muestras tienen igual distribución de probabilidad, es decir, que no hay diferencia entre los núcleos, lo que me parece extraño porque gráficamente no se muestra así pero tiene sentido ya que estamos trabajando con decimales y en realidad son valores tan pequeños que son parecidos aún cuando varían en décimas.