$^{a}Hernandez\ Pachon, Diego\ Alejandro\ A00362976$;  $^{b}Díaz\ Parra, Alejandra\ A00358235$
$^{a}dialhepachon@gmail.com$;  $^{b}alejandradp19@gmail.com$
$^{a}Ingeniería\ de\ Sistemas$;  $^{b}Ingeniería\ Telemática\ e\ Ingeniería\ de\ Sistemas$

# TALLER DE DISEÑO DE EXPERIMENTOS

## Planeación y Realización

El objetivo del experimento se centra en averiguar de qué forma influyen ciertos factores en la velocidad de ejecución de un algoritmo de ordenamiento en 4 lenguajes de programación concretos, lo anterior con el fin de conocer el lenguaje de programación que brinda el mejor tiempo de ejecución.

Teniendo en cuenta el objetivo, se determinaron a continuación los aspectos claves que se tendrán en cuenta durante el experimento:

- Unidad experimental
    - El algoritmo de ordenamiento.

- Factores controlables:
    - Lenguaje de programación.
    - Tamaño del arreglo.
    - Estado de los valores en el arreglo.

- Factores no controlables:
    - Programas abiertos y procesos que se estén ejecutando en segundo plano.
    - Edad del hardware usado para llevar a cabo el experimento.
    - El tipo de procesador, que determina factores cómo
        - La velocidad del reloj del procesador.
        - El número de registros del procesador.
        - La arquitectura del procesador.
        - Número de núcleos e hilos.

Debido a que cada uno de los factores no controlados puede alterar las variables de respuesta, se decidió intentar controlarlas de la siguiente forma:

En el primer caso, se cerraron la mayor cantidad de programas y procesos en segundo plano que fuese posible para minimizar su efecto en los resultados.
En el segundo y tercer caso, se intentaron controlar estos factores realizando la toma de datos para el experimento en un sólo computador.

- Factores estudiados:
    - Lenguaje de programación.
    - Tamaño del arreglo.
    - Estado de los valores en el arreglo.

- Niveles de dichos factores:
    - Lenguaje de programación: C#, Kotlin, GO, Java.
    - Tamaño del arreglo: 500, 5000, 10000.
    - Estado de los valores en el arreglo: en orden aleatorio, orden ascendente, orden descendente.

- Tratamientos:

| Lenguaje | Tamaño del arreglo | Estado de los valores |
|---|---|---|
| C# | 500 | Orden aleatorio |
| C# | 500 | Orden ascendente |
| C# | 500 | Orden descendente |
| C# | 5000 | Orden aleatorio |
| C# | 5000 | Orden ascendente |
| C# | 5000 | Orden descendente |
| C# | 10000 | Orden aleatorio |
| C# | 10000 | Orden ascendente |
| C# | 10000 | Orden descendente |
| Kotlin | 500 | Orden aleatorio |
| Kotlin | 500 | Orden ascendente |
| Kotlin | 500 | Orden descendente |
| Kotlin | 5000 | Orden aleatorio |
| Kotlin | 5000 | Orden ascendente |
| Kotlin | 5000 | Orden descendente |
| Kotlin | 10000 | Orden aleatorio |
| Kotlin | 10000 | Orden ascendente |
| Kotlin | 10000 | Orden descendente |
| Go | 500 | Orden aleatorio |
| Go | 500 | Orden ascendente |
| Go | 500 | Orden descendente |

| | | |
|---|---|---|
| Go | 5000 | Orden aleatorio |
| Go | 5000 | Orden ascendente |
| Go | 5000 | Orden descendente |
| Go | 10000 | Orden aleatorio |
| Go | 10000 | Orden ascendente |
| Go | 10000 | Orden descendente |
| Java | 500 | Orden aleatorio |
| Java | 500 | Orden ascendente |
| Java | 500 | Orden descendente |
| Java | 5000 | Orden aleatorio |
| Java | 5000 | Orden ascendente |
| Java | 5000 | Orden descendente |
| Java | 10000 | Orden aleatorio |
| Java | 10000 | Orden ascendente |
| Java | 10000 | Orden descendente |

- Número de repeticiones:
    - 100 por cada tratamiento.
- Variables de respuesta:
    - Tiempos de ejecución de cada algoritmo.
    - Eficiencia de cada lenguaje.

Durante el experimento se utilizó la siguiente implementación de un bubblesort:

```java
int temp = 0;
for(int i=0; i < arr.length; i++){
        for(int j=1; j < (arr.length-1); j++){
                if(arr[j-1] > arr[j]){
                        //swap elements
                        temp = arr[j-1];
                        arr[j-1] = arr[j];
                        arr[j] = temp;
                }

        }
}
```

Ilustración 1 - Implementación de bubblesort en JAVA.

```csharp
int temp;
for (int j = 0; j <= arr.Length - 2; j++)
{
    for (int i = 0; i <= arr.Length - 2; i++)
    {
        if (arr[i] > arr[i + 1])
        {
            temp = arr[i + 1];
            arr[i + 1] = arr[i];
            arr[i] = temp;
        }
    }
}
```

Ilustración 2 - Implementación de bubblesort en C#.

```kotlin
for(j in 0 until arr.size - 1){
    for (i in 0 until arr.size - 1) {
        if (arr[i] > arr[i + 1]) {
            temp = arr[i+1]
            arr[i+1] = arr[i]
            arr[i] = temp
        }
    }
}
```

Ilustración 3 - Implementación de bubblesort en Kotlin.

```go
for j := 0; j <= len(arr)-2; j++ {
    for i := 0; i <= len(arr)-2; i++ {
        if arr[i] > arr[i+1] {
            temp = arr[i+1]
            arr[i+1] = arr[i]
            arr[i] = temp
        }
    }
}
```

Ilustración 4 - Implementación de bubblesort en Go.

Se sabe que el orden de ejecución del experimento debería ser aleatorio para evitar confundir una de las variables de estudio con una variable oculta (variable no controlada y no observada qué cambia en el experimento), pero por practicidad del experimento se tomó la decisión de utilizar el orden estándar.

El diseño del experimento fue factorial completo con 36 tratamientos diferentes; las ejecuciones se realizaron en orden estándar, cada tratamiento tuvo 100 repeticiones de las que se tomaron los datos. Para analizar estos resultados se utilizará la prueba de ANOVA, buscando determinar si hay una diferencia estadísticamente significativa entre las medias de

los tiempos de ejecución, y si llega a determinarse dicha diferencia se buscará identificar el lenguaje que produce el menor tiempo de ejecución promedio.

**Análisis**

Si se asume la longitud del arreglo a tratar en el experimento como un valor entero n, se tiene que el bubblesort ejecuta el siguiente número de operaciones

$$c(n) = \frac{n^2 - n}{2}$$

De lo cual se puede apreciar que, aunque suene contraintuitivo, la cantidad de operaciones que realiza el algoritmo depende de la cantidad de términos sobre los que se aplique, y no del orden en el que estos se encuentren. De esto se deriva que:

$$\Theta(c(n)) = n^2$$

Sin embargo, el número de intercambios que realiza el algoritmo para ordenar el arreglo sí que depende del orden de los elementos que este contiene. Para analizar la complejidad de esto, se tiene que el mejor caso se da cuando el arreglo está ordenado ascendentemente antes de aplicarle el bubblesort, y el peor se da cuando el arreglo está previamente ordenado de manera descendente.

Para el primero de estos casos, el número de comparaciones será

$$\Omega(c(n)) = n^2$$

A su vez, como en todas las comparaciones el orden es el que se busca lograr no se realiza ningún intercambio, por lo que estos serán

$$i(n) = 0$$

Nótese que la cantidad de intercambios en este caso no depende de n, y es de

$$\Omega(i(n)) = 1$$

Ahora, para el peor caso se tiene que la cantidad de comparaciones que realiza es la mencionada inicialmente, es decir

$$c(n) = \frac{n^2 - n}{2}$$

Como en todas esas comparaciones se tiene que los valores están ubicados inversamente con respecto al orden al que se desea llegar, se tendrá que realizar un intercambio por cada comparación, es decir:

$$i(n) = \frac{n^2 - n}{2}$$

Como se puede apreciar, en el peor caso la cantidad de comparaciones y de intercambios coinciden, ocasionando que sus complejidades también lo hagan, así:

$$O(c(n)) = (O(i(n)) = n^2$$

Aparte de tener esta parte teórica como supuestos, se plantearon las siguientes hipótesis que permitieron generar el experimento:

- El tamaño de la matriz sí influye en el aumento del tiempo de ejecución del algoritmo.
- El tipo de lenguaje de programación en el cual se implementa el algoritmo sí influye en el tiempo de ejecución de este.
- El orden en el que estén los elementos del arreglo sobre el cual se va a efectuar el algoritmo de ordenamiento elegido sí influye en los tiempos de ejecución de este algoritmo.

**Interpretación**

A continuación se presentan los resultados obtenidos del análisis de los datos utilizando la prueba ANOVA:

Para los análisis de ANOVA se trabajaron variaciones de las siguientes hipótesis:

- Ho: No hay diferencias en el tiempo promedio de ejecución del algoritmo de ordenamiento entre los diferentes lenguajes de programación con un tamaño de arreglo de n elementos.
- H1: Al menos un par de medias presentan diferencias en el tiempo promedio de ejecución del algoritmo de ordenamiento entre los diferentes lenguajes de programación con un tamaño de arreglo de n elementos.

Con eso se obtuvieron los siguientes resultados:

SUMMARY

| *Groups* | *Count* | *Sum* | *Average* | *Variance* |
|---|---|---|---|---|
| C# | 100 | 238.8441 | 2.388441 | 0.161882334 |
| Java | 100 | 34.7095 | 0.347095 | 0.751451579 |
| Kotlin | 100 | 11.5963 | 0.115963 | 0.00315765 |
| Go | 100 | 33.544 | 0.33544 | 0.217659186 |

ANOVA

| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
|---|---|---|---|---|---|---|
| Between Groups | 341.1947713 | 3 | 113.7315904 | 401.1163082 | 1E-119 | 2.62744077 |
| Within Groups | 112.2809242 | 396 | 0.283537687 | | | |
| Total | 453.4756955 | 399 | | | | |

Tabla 1- Análisis de varianza para arreglos de tamaño 500 organizados ascendentemente.

SUMMARY

| *Groups* | *Count* | *Sum* | *Average* | *Variance* |
|---|---|---|---|---|
| C# | 100 | 265.7597 | 2.657597 | 0.158165 |
| Java | 100 | 33.9135 | 0.339135 | 0.246829 |
| Kotlin | 100 | 31.0786 | 0.310786 | 0.001992 |
| Go | 100 | 34.0071 | 0.340071 | 0.226762 |

ANOVA

| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
|---|---|---|---|---|---|---|
| Between Groups | 406.3844 | 3 | 135.4615 | 854.9862 | 1.5E-172 | 2.627441 |
| Within Groups | 62.74107 | 396 | 0.158437 | | | |
| Total | 469.1255 | 399 | | | | |

Tabla 2 - Análisis de varianza para arreglos de tamaño 500 organizados descendentemente.

SUMMARY

| *Groups* | *Count* | *Sum* | *Average* | *Variance* |
|---|---|---|---|---|
| C# | 100 | 188.2359 | 1.882359 | 0.075208 |
| Java | 100 | 60.9911 | 0.609911 | 0.282154 |
| Kotlin | 100 | 41.3837 | 0.413837 | 0.284167 |
| Go | 100 | 30.7378 | 0.307378 | 0.101261 |

ANOVA

| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
|---|---|---|---|---|---|---|
| Between Groups | 159.9388 | 3 | 53.31292 | 287.0953 | 6E-99 | 2.627441 |
| Within Groups | 73.53627 | 396 | 0.185698 | | | |
| Total | 233.475 | 399 | | | | |

Tabla 3- Análisis de varianza para arreglos de tamaño 500 organizados aleatoriamente.

SUMMARY

| *Groups* | *Count* | *Sum* | *Average* | *Variance* |
|---|---|---|---|---|
| C# | 100 | 13408.08 | 134.0808 | 1000.02 |
| Java | 100 | 621 | 6.21 | 37.19788 |
| Kotlin | 100 | 1026.429 | 10.26429 | 2.487402 |
| Go | 100 | 2805 | 28.05 | 1.704545 |

ANOVA

| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
|---|---|---|---|---|---|---|
| Between Groups | 1093344 | 3 | 364448 | 1399.825 | 2.4E-210 | 2.627441 |
| Within Groups | 103099.6 | 396 | 260.3526 | | | |
| Total | 1196444 | 399 | | | | |

Tabla 4 - Análisis de varianza para arreglos de tamaño 5000 organizados ascendentemente.

SUMMARY

| *Groups* | *Count* | *Sum* | *Average* | *Variance* |
|---|---|---|---|---|
| C# | 100 | 17129.06 | 171.2906 | 1523.174 |
| Java | 100 | 2038 | 20.38 | 32.25818 |
| Kotlin | 100 | 2923.72 | 29.2372 | 1.469398 |
| Go | 100 | 2826 | 28.26 | 3.325657 |

ANOVA

| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
|---|---|---|---|---|---|---|
| Between Groups | 1588810 | 3 | 529603.4 | 1357.759 | 6E-208 | 2.627441 |
| Within Groups | 154462.5 | 396 | 390.0568 | | | |
| Total | 1743273 | 399 | | | | |

Tabla 5 - Análisis de varianza para arreglos de tamaño 5000 organizados descendentemente.

SUMMARY

| *Groups* | *Count* | *Sum* | *Average* | *Variance* |
|---|---|---|---|---|
| C# | 100 | 18342.66 | 183.4266 | 910.3582 |
| Java | 100 | 3305 | 33.05 | 87.86616 |
| Kotlin | 100 | 2764.414 | 27.64414 | 4.462711 |
| Go | 100 | 2553 | 25.53 | 1.322323 |

ANOVA

| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
|---|---|---|---|---|---|---|
| Between Groups | 1797571 | 3 | 599190.3 | 2387.19 | 4.1E-253 | 2.627441 |
| Within Groups | 99396.93 | 396 | 251.0023 | | | |
| Total | 1896968 | 399 | | | | |

Tabla 6- Análisis de varianza para arreglos de tamaño 5000 organizados aleatoriamente.

SUMMARY

| *Groups* | *Count* | *Sum* | *Average* | *Variance* |
|---|---|---|---|---|
| C# | 100 | 43943.29 | 439.4329 | 5036.792 |
| Java | 100 | 2536 | 25.36 | 399.6065 |
| Kotlin | 100 | 3942.583 | 39.42583 | 22.72637 |
| Go | 100 | 11224 | 112.24 | 3.679192 |

ANOVA

| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
|---|---|---|---|---|---|---|
| Between Groups | 11289126 | 3 | 3763042 | 2755.392 | 7.7E-265 | 2.627441 |
| Within Groups | 540817.6 | 396 | 1365.701 | | | |
| Total | 11829943 | 399 | | | | |

Tabla 7 - Análisis de varianza para arreglos de tamaño 10000 organizados ascendentemente.

SUMMARY

| *Groups* | *Count* | *Sum* | *Average* | *Variance* |
|---|---|---|---|---|
| C# | 100 | 60346.96 | 603.4696 | 6742.173 |
| Java | 100 | 7719 | 77.19 | 377.9534 |
| Kotlin | 100 | 11651.11 | 116.5111 | 28.72518 |
| Go | 100 | 11246 | 112.46 | 4.109495 |

ANOVA

| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
|---|---|---|---|---|---|---|
| Between Groups | 18949898 | 3 | 6316633 | 3532.318 | 2.5E-285 | 2.627441 |
| Within Groups | 708143.1 | 396 | 1788.24 | | | |
| Total | 19658041 | 399 | | | | |

Tabla 8 - Análisis de varianza para arreglos de tamaño 10000 organizados descendentemente.

| SUMMARY | | | | | | |
|---|---|---|---|---|---|---|
| *Groups* | *Count* | *Sum* | *Average* | *Variance* | | |
| C# | 100 | 81260.83 | 812.6083 | 9070.907 | | |
| Java | 100 | 17498 | 174.98 | 1309.414 | | |
| Kotlin | 100 | 12107.69 | 121.0769 | 32.55019 | | |
| Go | 100 | 10990 | 109.9 | 2.838384 | | |
| | | | | | | |
| | | | | | | |
| ANOVA | | | | | | |
| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
| Between Groups | 34646260 | 3 | 11548753 | 4435.129 | 2.9E-304 | 2.627441 |
| Within Groups | 1031155 | 396 | 2603.927 | | | |
| | | | | | | |
| Total | 35677416 | 399 | | | | |

Tabla 9- Análisis de varianza para arreglos de tamaño 10000 organizados aleatoriamente.

En todas las tablas anteriores se puede observar que se rechaza la hipótesis nula para todos los factores que se escogieron. Esto implica que se acepta la hipótesis alternativa, y que al menos un par de las medias es diferente. Se sabe entonces que los factores seleccionados efectivamente influencian el tiempo de ejecución del algoritmo.

Aparte de esto, se realizaron diferentes análisis para analizar la eficiencia de cada una de las implementaciones del algoritmo, obteniendo lo siguiente:

| | | Difference | SE | q | |
|---|---|---|---|---|---|
| C# | Java | 2.04 | 0.05 | 38.34 | Java |
| C# | Kotlin | 2.27 | 0.05 | 42.68 | Kotlin |
| C# | Go | 2.05 | 0.05 | 38.56 | Go |
| Java | Kotlin | 0.23 | 0.05 | 4.34 | Kotlin |
| Java | Go | 0.01 | 0.05 | 0.22 | |
| Kotlin | Go | 0.22 | 0.05 | 4.12 | Kotlin |
| | | | q value = | 3.63 | |

Tabla 10 - Post-anova comparando las implementaciones de los algoritmos sobre arreglos de tamaño 500 ordenados ascendentemente.

De este análisis se obtuvo que la implementación en Kotlin presenta diferencias significativas con respecto a los demás lenguajes, siendo la que tuvo el menor tiempo de ejecución en promedio para este tamaño y orden de los elementos en el arreglo.

| | | Difference | SE | q | |
|---|---|---|---|---|---|
| C# | Java | 2.32 | 0.04 | 58.25 | Java |
| C# | Kotlin | 2.35 | 0.04 | 58.96 | Kotlin |
| C# | Go | 2.32 | 0.04 | 58.22 | Go |
| Java | Kotlin | 0.03 | 0.04 | 0.71 | |
| Java | Go | 0.00 | 0.04 | 0.02 | |
| Kotlin | Go | 0.03 | 0.04 | 0.74 | |
| | | | q value = | 3.63 | |

Tabla 11 - Post-anova comparando las implementaciones de los algoritmos sobre arreglos de tamaño 500 ordenados descendentemente.

De este análisis se obtuvo que las implementaciones en Java, Kotlin y Go presentan diferencias significativas con respecto a C#, presentando menores tiempos de ejecución en promedio para este tamaño y orden de los elementos en el arreglo. Sin embargo, no mostraron diferencia significativa entre ellos.

| | | Difference | SE | q | |
|---|---|---|---|---|---|
| C# | Java | 1.27 | 0.04 | 29.53 | Java |
| C# | Kotlin | 1.47 | 0.04 | 34.08 | Kotlin |
| C# | Go | 1.57 | 0.04 | 36.55 | Go |
| Java | Kotlin | 0.20 | 0.04 | 4.55 | Kotlin |
| Java | Go | 0.30 | 0.04 | 7.02 | Go |
| Kotlin | Go | 0.11 | 0.04 | 2.47 | |
| | | | q value = | 3.63 | |

Tabla 12 - Post-anova comparando las implementaciones de los algoritmos sobre arreglos de tamaño 500 ordenados aleatoriamente.

De esta prueba se tiene que en las implementaciones en Kotlin y en Go se presentan diferencias significativas con respecto a los demás lenguajes. A su vez, tienen los menores tiempos de ejecución en promedio para este tamaño y orden de los elementos en el arreglo.

| | | Difference | SE | q | |
|---|---|---|---|---|---|
| C# | Java | 127.87 | 1.61 | 79.25 | Java |
| C# | Kotlin | 123.82 | 1.61 | 76.74 | Kotlin |
| C# | Go | 106.03 | 1.61 | 65.71 | Go |
| Java | Kotlin | 4.05 | 1.61 | 2.51 | |
| Java | Go | 21.84 | 1.61 | 13.54 | Java |
| Kotlin | Go | 17.79 | 1.61 | 11.02 | Kotlin |
| | | | q value = | 3.63 | |

Tabla 13 - Post-anova comparando las implementaciones de los algoritmos sobre arreglos de tamaño 5000 ordenados ascendentemente.

De esta prueba se obtuvo que en las implementaciones en Kotlin y en Java se presentan diferencias significativas con respecto a los demás lenguajes. A su vez, tienen los menores tiempos de ejecución en promedio para este tamaño y orden de los elementos en el arreglo.

| | | Difference | SE | q | |
|---|---|---|---|---|---|
| C# | Java | 150.91 | 1.97 | 76.41 | Java |
| C# | Kotlin | 142.05 | 1.97 | 71.93 | Kotlin |
| C# | Go | 143.03 | 1.97 | 72.42 | Go |
| Java | Kotlin | 8.86 | 1.97 | 4.48 | Java |
| Java | Go | 7.88 | 1.97 | 3.99 | Java |
| Kotlin | Go | 0.98 | 1.97 | 0.49 | |
| | | | q value = | 3.63 | |

Tabla 14 - Post-anova comparando las implementaciones de los algoritmos sobre arreglos de tamaño 5000 ordenados descendentemente.

Observando estos resultados se observa que en la implementación en Java se presentan diferencias significativas con respecto a los demás lenguajes. A su vez, tiene el menor tiempo de ejecución en promedio para este tamaño y orden de los elementos en el arreglo.

| | | Difference | SE | q | |
|---|---|---|---|---|---|
| C# | Java | 150.38 | 1.58 | 94.92 | Java |
| C# | Kotlin | 155.78 | 1.58 | 98.33 | Kotlin |
| C# | Go | 157.90 | 1.58 | 99.66 | Go |
| Java | Kotlin | 5.41 | 1.58 | 3.41 | |
| Java | Go | 7.52 | 1.58 | 4.75 | Go |
| Kotlin | Go | 2.11 | 1.58 | 1.33 | |
| | | | q value = | 3.63 | |

Tabla 15 - Post-anova comparando las implementaciones de los algoritmos sobre arreglos de tamaño 5000 ordenados aleatoriamente.

Observando este resultado se tiene que la implementación en Go presentó diferencias con respecto a Java mostrando ser levemente más rápida, sin embargo con Kotlin no obtuvo ninguna diferencia significativa. Por otro lado, C# mantuvo el comportamiento del resto de los análisis obteniendo los mayores tiempos de ejecución.

| | | Difference | SE | q | |
|---|---|---|---|---|---|
| C# | Java | 414.07 | 3.70 | 112.05 | Java |
| C# | Kotlin | 400.01 | 3.70 | 108.24 | Kotlin |
| C# | Go | 327.19 | 3.70 | 88.54 | Go |
| Java | Kotlin | 14.07 | 3.70 | 3.81 | Java |
| Java | Go | 86.88 | 3.70 | 23.51 | Java |
| Kotlin | Go | 72.81 | 3.70 | 19.70 | Kotlin |
| | | | q value = | 3.63 | |

Tabla 16 - Post-anova comparando las implementaciones de los algoritmos sobre arreglos de tamaño 10000 ordenados ascendentemente.

Como se puede apreciar en lo obtenido de esta prueba, las implementaciones de cada lenguaje manifiestan diferencias significativas entre sí. Aparte de esto, se aprecia que la implementación en Java fue la que presentó el menor tiempo de ejecución en promedio para este tamaño y orden de los elementos en el arreglo.

| | | Difference | SE | q | |
|---|---|---|---|---|---|
| C# | Java | 526.28 | 4.23 | 124.45 | Java |
| C# | Kotlin | 486.96 | 4.23 | 115.15 | Kotlin |
| C# | Go | 491.01 | 4.23 | 116.11 | Go |
| Java | Kotlin | 39.32 | 4.23 | 9.30 | Java |
| Java | Go | 35.27 | 4.23 | 8.34 | Java |
| Kotlin | Go | 4.05 | 4.23 | 0.96 | |
| | | | q value = | 3.63 | |

Tabla 17 - Post-anova comparando las implementaciones de los algoritmos sobre arreglos de tamaño 10000 ordenados descendentemente.

En este resultado se puede observar que la implementación en Java presenta diferencias significativas con respecto a los demás lenguajes. A su vez, tiene el menor tiempo de ejecución en promedio para este tamaño y orden de los elementos en el arreglo.

| | | Difference | SE | q | |
|---|---|---|---|---|---|
| C# | Java | 637.63 | 5.10 | 124.95 | Java |
| C# | Kotlin | 691.53 | 5.10 | 135.52 | Kotlin |
| C# | Go | 702.71 | 5.10 | 137.71 | Go |
| Java | Kotlin | 53.90 | 5.10 | 10.56 | Kotlin |
| Java | Go | 65.08 | 5.10 | 12.75 | Go |
| Kotlin | Go | 11.18 | 5.10 | 2.19 | |
| | | | q value = | 3.63 | |

Tabla 18 - Post-anova comparando las implementaciones de los algoritmos sobre arreglos de tamaño 10000 ordenados aleatoriamente.

Finalmente, los resultados de esta última prueba demuestran que las implementaciones en Kotlin y en Go manifiestan diferencias significativas con las implementaciones en los demás lenguajes. Además, presentan los menores tiempos de ejecución en promedio para este tamaño y orden de los elementos en el arreglo.

**Control y conclusiones finales**

Gracias a los análisis efectuados en este experimento se pudieron concluir las siguientes afirmaciones:

- Tanto el lenguaje de programación, el tamaño y el estado de los valores del arreglo sobre el cual se aplique el algoritmo de ordenamiento influencian el tiempo de ejecución del algoritmo.
- Independientemente del tamaño o del estado de los valores en el arreglo sobre el cual se aplicó la implementación de C#, ésta presentó los tiempos mayores de ejecución con respecto a los demás lenguajes.
- Para el peor caso de ejecución el algoritmo, es decir, para cuando los valores del arreglo sobre el cual se aplica están ordenados descendentemente, se tiene que la implementación en Java presenta los menores tiempos de ejecución para cualquier tamaño del arreglo mencionado.
- Para el caso promedio de ejecución del algoritmo, es decir, para cuando los elementos que conforman el arreglo sobre el cual se va a aplicar están ordenados aleatoriamente se evidenció que las implementaciones en Kotlin y en Go presentan los menores tiempos de ejecución para tamaños de arreglo grandes y pequeños, sin embargo, para el tamaño de arreglo intermedio no se encontró diferencia significativa en los tiempos entre los lenguajes de Java, Kotlin y Go.

- Cuando los valores en el arreglo sobre el cual se va a aplicar el algoritmo de ordenamiento están ordenados ascendentemente, se pudo apreciar que la implementación en Kotlin presenta los menores tiempos de ejecución para tamaños de arreglo pequeños. A su vez, la implementación en Java presenta el menor tiempo de ejecución para tamaños de arreglo grandes.

**Referencias**

*Sorting algorithms/Bubble sort*. Rosetta Code. (n.d.). Retrieved September 27, 2021, from http://rosettacode.org/wiki/Sorting_algorithms/Bubble_sort.

*Ordenamiento de Burbuja*. EcuRed. (n.d.). Retrieved September 27, 2021, from https://www.ecured.cu/Ordenamiento_de_burbuja#An.C3.A1lisis_del_Costo_Computacional.

BitCuco. (2020, July 25). *Algoritmo de ordenamiento en Javascript: BubbleSort*. BitCuco. Retrieved September 27, 2021, from https://bitcu.co/algoritmo-de-ordenamiento-en-javascript-bubblesort/.