1. *Planeación y Realización*

Para cumplir con el objetivo del experimento, se determinará que algoritmo de ordenamiento es más eficiente en los diferentes lenguajes de programación para ordenar arreglos de diferentes tamaños.

*Fenómeno*

Bubble Sort: Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada

Insertion Sort: Su funcionamiento consiste en el recorrido por la lista seleccionando en cada iteración un valor como clave y compararlo con el resto insertándolo en el lugar correspondiente.

*Unidad Experimental*

La unidad experimental son los algoritmos de ordenamiento Bubblesort e Insertionsort utilizados en este experimento, desarrollados en C#, Goolang y Kotlin y con factores esenciales como el tamaño y orden.

*Variable(s) de Respuesta*

La variable de es el tiempo del ordenamiento para cada algoritmo en base a los diferentes factores que componen el experimento, los diferentes órdenes y tamaños que pueden generarse.

Factores controlables

* Lenguaje de programación (este factor es obligatorio y los niveles deben ser, al menos, los lenguajes utilizados en el curso hasta el momento)
* Algoritmo de ordenamiento (Burbuja, Selection, QuickSort, HeapSort, etc)
* Tamaño del arreglo (10^1, 10^2, 10^3, 10^4, etc)
* Estado de los valores en el arreglo (en orden aleatorio, ordenado ascendente, ordenado descendente)
* RAM del computador donde se ejecuta el algoritmo (2GB, 4GB, 8GB, 16GB, etc)
* Procesador del computador donde se ejecuta el algoritmo
* Sistema Operativo
* Cantidad de procesos que se están ejecutando en el computador mientras se ejecuta el algoritmo
* Nivel de fragmentación del disco duro del computador donde se ejecuta el algoritmo
* Tamaño del registro del procesador (8bits, 16bits, 32bits, 64bits)

*Factores no controlables*

* Temperatura del medio ambiente y del computador
* Memoria RAM al compilar el programa
* Tipo de compilador

*Niveles*

Lenguajes de Programación

* C#
* Goolang
* Kotlin
* Java

|  |  |
| --- | --- |
| *Lenguaje de Programación* | *Nivel del lenguaje* |
| C# | 1 |
| Goolang | 2 |
| Kotlin | 3 |
| Java | 4 |

Algoritmo de ordenamiento

* Bubble
* Insertion

|  |  |
| --- | --- |
| *Algoritmo de Ordenamiento* | *Nivel del algoritmo* |
| Bubble | 1 |
| Insertion | 2 |

Tamaño del arreglo:

* Pequeño
* Grande

|  |  |
| --- | --- |
| *Tamaño del arreglo* | *Nivel del tamaño* |
| Pequeño | 1 |
| Grande | 2 |

*Tratamientos*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Lenguaje de Programación* | *Algoritmo de ordenamiento* | *Tamaño del arreglo* | *Repetición* | *Tratamiento* | *Tiempo* |
| C# | Bubble | 10 | 100 | 1 |  |
| C# | Bubble | 10^4 | 100 | 2 |  |
| C# | Insertion | 10 | 100 | 3 |  |
| C# | Insertion | 10^4 | 100 | 4 |  |
| Goolang | Bubble | 10 | 100 | 5 |  |
| Goolang | Bubble | 10^4 | 100 | 6 |  |
| Goolang | Insertion | 10 | 100 | 7 |  |
| Goolang | Insertion | 10^4 | 100 | 8 |  |
| Kotlin | Bubble | 10 | 100 | 9 |  |
| Kotlin | Bubble | 10^4 | 100 | 10 |  |
| Kotlin | Insertion | 10 | 100 | 11 |  |
| Kotlin | Insertion | 10^4 | 100 | 12 |  |
| Java | Bubble | 10 | 100 | 13 |  |
| Java | Bubble | 10^4 | 100 | 14 |  |
| Java | Insertion | 10 | 100 | 15 |  |
| Java | Insertion | 10^4 | 100 | 16 |  |

(B) BUBBLE SORT: Complejidad Temporal

|  |  |
| --- | --- |
| *Línea* | *# de veces que se repite* |
| int n = arr.length; | 1 |
| for (int i = 0; i < n-1; i++) | n+1 |
| for (int j = 0; j < n-i-1; j++) | ((n(n+1))/2) + n |
| if (arr[j] > arr[j+1]) | ((n(n+1))/2) + 1 |
| int temp = arr[j]; | ((n(n+1))/2) + 1 |
| arr[j] = arr[j+1]; | ((n(n+1))/2) + 1 |
| arr[j+1] = temp; | ((n(n+1))/2) + 1 |

T(B) = 1 + (n+1) + ((n(n+1))/2) + n + ((n(n+1))/2) + 1 + ((n(n+1))/2) + 1 + ((n(n+1))/2) + 1 + ((n(n+1))/2) + 1

T(B) = 2n + ((5n^2 + 2n)/2) + 6

T(B)= O(n^2)

(B) BUBBLE SORT: Complejidad Espacial

|  |  |  |
| --- | --- | --- |
| *Tipo* | *Variable* | *Cantidad de valores atómicos* |
| Input | n | n |
| Auxilar | temp  I  j | 1  1  1 |
| Output | - | - |

T(B)= n + 3

T(B) = ө (n)

(I) INSERT SORT: Complejidad Temporal

|  |  |
| --- | --- |
| *Linea* | *# de veces que se repite* |
| int n = arr.length; | 1 |
| for (int i = 1; i < n; ++i) { | n+1 |
| int key = arr[i]; | n |
| int j = i - 1; | n |
| while (j >= 0 && arr[j] > key) | ((n(n+1))/2) + n |
| arr[j + 1] = arr[j]; | ((n(n+1))/2) + 1 |
| j = j - 1; | ((n(n+1))/2) + 1 |
| arr[j + 1] = key; | ((n(n+1))/2) + 1 |

T(I) = 1 + (n+1) + n + n + ((n(n+1))/2) + n + ((n(n+1))/2) + 1 + ((n(n+1))/2) + 1 + ((n(n+1))/2) + 1 +

T(I) =3 n + ((4n^2 + 2n)/2) + 5

T(I)= O(n^2)

|  |  |  |
| --- | --- | --- |
| *Tipo* | *Variable* | *Cantidad de valores atómicos* |
| Input | n | n |
| Auxilar | key  I  j | 1  1  1 |
| Output | - | - |

T(I)= n + 3

T(I) = ө (n)