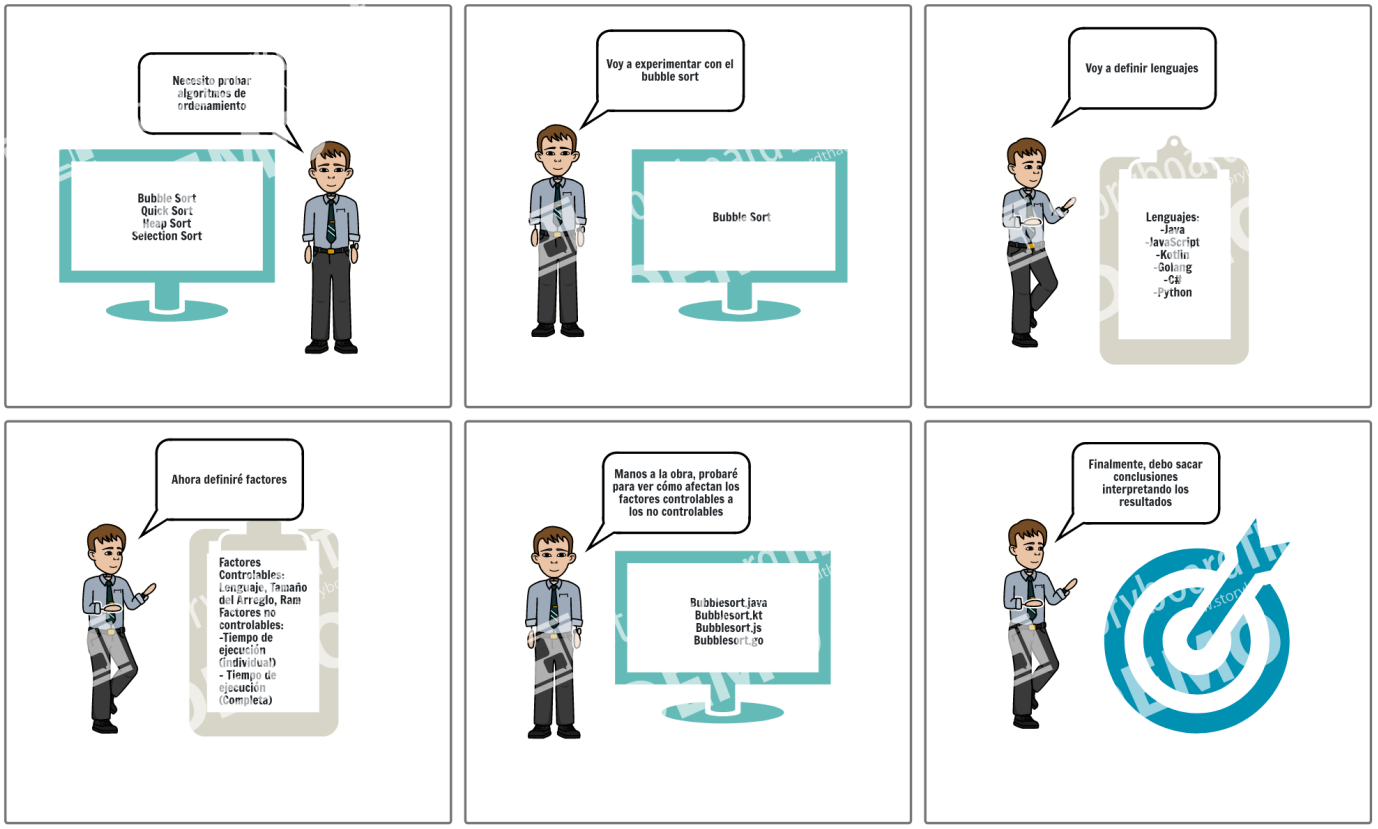
Experimento Bubble Sort

Giovanni Mosquera
Juan Sebastián Rodríguez
Juan Pablo Sanin A00296776
Juan Camilo Zorrilla

Planeación y Realización

Storyboard:



Cree sus los propios en Storyboard That

- **Unidad experimental:**
  Para nuestro experimento tomaremos en cuenta 3 muestras las cuales serán:
  - Lenguaje de programación: Lenguaje en el cual se desarrollará nuestro algoritmo de ordenamiento elegido.
  - Algoritmo de ordenamiento: Algoritmo que se utilizará para ordenar el arreglo.
  - Tamaño del arreglo: Cantidad de números a ordenar dentro de un arreglo.
  - RAM del computador: Memoria RAM del computador en el cual se ejecutará el algoritmo de ordenamiento.
- **Factores controlables:** Para nuestro experimento nuestras variables controlables serán el lenguaje de programación, el algoritmo de ordenamiento, el tamaño del arreglo y la memoria RAM que utilizaremos.

- **Factores no controlables:** Para nuestro experimento nuestras variables no controlables serán el tiempo de ejecución para cada arreglo y el tiempo de ejecución promedio

- **Niveles:**
  Nuestras variables poseen los siguientes niveles:
  lenguaje de programación (6 niveles): Los 6 niveles son Java, JS, Golang, Python, C#, Kotlin.
  Algoritmos de ordenamiento (1 nivel): Nuestro único nivel será el bubble sort.
  Tamaño del arreglo (4 niveles): Los 4 niveles para esta variable serán 10^1, 10^2, 10^3, 10^4
  Memoria RAM (3 niveles): 8 gigas, 12 gigas, 16 gigas.
- **Tratamientos:** En el experimento que estamos planteando tendremos 72 tratamientos, los cuales resultan de la multiplicación de todos nuestros niveles.
- **Variables de respuesta:** Para este caso la variable de respuesta será el tiempo que demore en ejecutar nuestro código

Recolección de Datos:

Computador de Juan Camilo Zorrilla (8 de RAM):

Java	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	8	0 ms (número muy pequeño)
	BubbleSort	10^2	8	0,22 ms
	BubbleSort	10^3	8	5,16 ms
	BubbleSort	10^4	8	543 ms

JS	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	8	0
	BubbleSort	10^2	8	2,28571E+14
	BubbleSort	10^3	8	7,26207E+15
	BubbleSort	10^4	8	7,21703E+15

Kotlin	Algoritmo de	Tamaño de arreglo	Ram	Tiempo (Respuesta

	ordenamiento			promedio en ms)
	BubbleSort	10^1	8	0,13
	BubbleSort	10^2	8	0,27
	BubbleSort	10^3	8	4,54
	BubbleSort	10^4	8	516,61

Golang	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en microsegundos)
	BubbleSort	10^1	8	0 microsegundos
	BubbleSort	10^2	8	9963 microsegundos
	BubbleSort	10^3	8	29632038349 microsegundos
	BubbleSort	10^4	8	9,15504E+13 microsegundos

C#	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	8	0,011174 ms
	BubbleSort	10^2	8	0,237942 ms
	BubbleSort	10^3	8	17,859162 ms
	BubbleSort	10^4	8	1909,531454 ms

Python	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	8	2,38E+10
	BubbleSort	10^2	8	0.0021122217178344725
	BubbleSort	10^3	8	0.27952577114105226
	BubbleSort	10^4	8	12.123.228.137.493.100

Computador de Giovanni Mosquera (12 de RAM):

Java	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	12	0
	BubbleSort	10^2	12	0,04
	BubbleSort	10^3	12	3,58
	BubbleSort	10^4	12	334,1

JS	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	12	0.003
	BubbleSort	10^2	12	5.95
	BubbleSort	10^3	12	20.02
	BubbleSort	10^4	12	282.27

Kotlin	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	12	0.16
	BubbleSort	10^2	12	0.17
	BubbleSort	10^3	12	5.38
	BubbleSort	10^4	12	363.33

Golang	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	12	0,000
	BubbleSort	10^2	12	17.291
	BubbleSort	10^3	12	63.490
	BubbleSort	10^4	12	69,374

C#	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	12	0,00001
	BubbleSort	10^2	12	0,00023
	BubbleSort	10^3	12	0,017
	BubbleSort	10^4	12	1,909

Python	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	12	0.0001
	BubbleSort	10^2	12	0.0019
	BubbleSort	10^3	12	0.1578
	BubbleSort	10^4	12	1.278

Computador de Juan Pablo Sanin (16 de RAM):

Java	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	16	0 ms
	BubbleSort	10^2	16	0.02 ms
	BubbleSort	10^3	16	1.13 ms
	BubbleSort	10^4	16	128.57 ms

JS	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	16	0.002 ms
	BubbleSort	10^2	16	0.027 ms
	BubbleSort	10^3	16	1.978 ms
	BubbleSort	10^4	16	205.024 ms

Kotlin	Algoritmo de	Tamaño de arreglo	Ram	Tiempo (Respuesta

	ordenamiento			promedio en ms)
	BubbleSort	10^1	16	0.06 ms
	BubbleSort	10^2	16	0.07 ms
	BubbleSort	10^3	16	1.12 ms
	BubbleSort	10^4	16	127.59 ms

Golang	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	16	0 microsegundos
	BubbleSort	10^2	16	0 microsegundos
	BubbleSort	10^3	16	277672 microsegundos
	BubbleSort	10^4	16	18109987 microsegundos

C#	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	16	0.000003781 ms
	BubbleSort	10^2	16	0.00004516 ms
	BubbleSort	10^3	16	0.003890 ms
	BubbleSort	10^4	16	0.45 ms

Python	Algoritmo de ordenamiento	Tamaño de arreglo	Ram	Tiempo (Respuesta promedio en ms)
	BubbleSort	10^1	16	0.00004985 ms
	BubbleSort	10^2	16	0.0005683 ms
	BubbleSort	10^3	16	0.02946 ms
	BubbleSort	10^4	16	2.8743 ms

Análisis de Complejidad

```
function bubbleSort(array){
    for(var i = 0; i < array.length; i++){ // n+1
        for(var j = 0; j < array.length-1; j++){ //n(n)+n
            if(array[j]>array[j+1]){ // n(n-1)
                let temp = array[j+1]; // n(n-1)
                array[j+1] = array[j];// n(n-1)
                array[j] = temp;// n(n-1)
            }
        }
    }
}
```

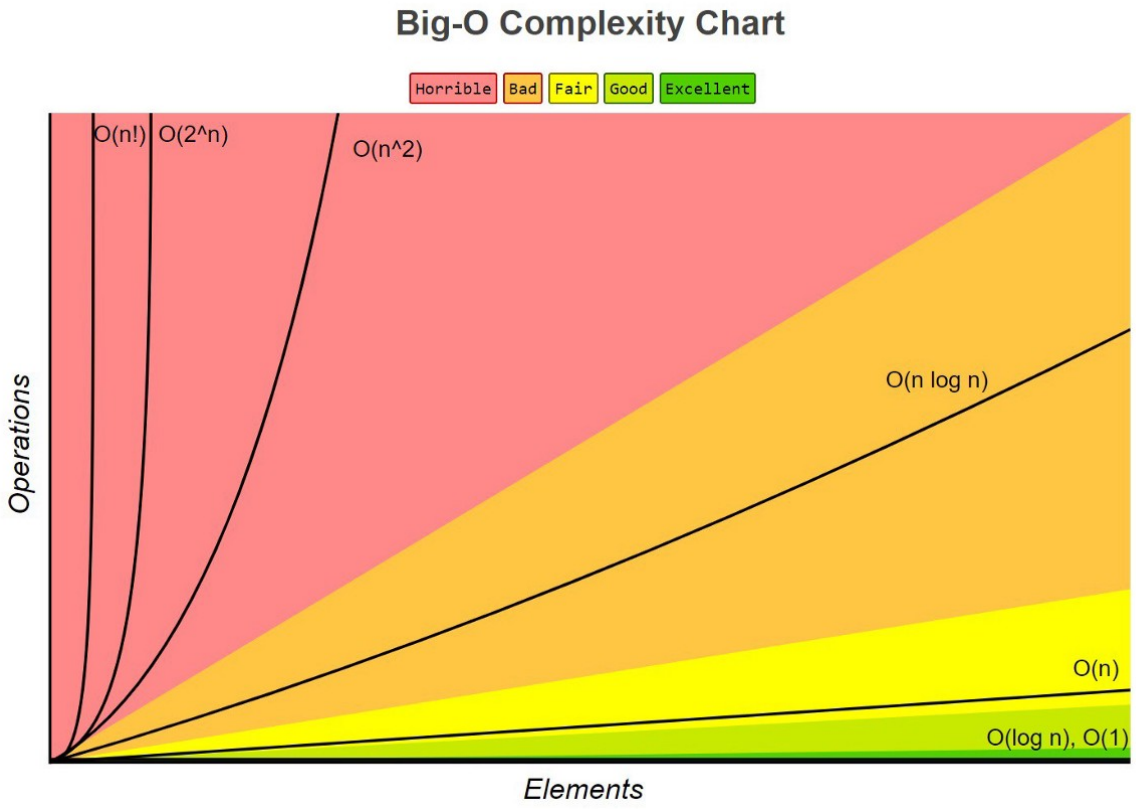
Total : 4(n(n-1)+n(n)+n+n+1

Total: 4(n^2-n)+n^2+2n+1

Total: 4n^2-4n+n^2+2n+1

Total: 5n^2-2n+1

O(n^2)



Análisis

Según los resultados obtenidos en los distintos computadores con distintas características, podemos ver que así como la RAM influye en el proceso de ordenamiento también influyen componentes como el procesador que posea el computador, la cantidad de procesos en segundo plano que tenga corriendo en ese momento, qué tan estresado está el procesador del computador e incluso, el nivel de temperatura a la que está el computador trabajando en ese momento, teniendo esto en mente se realizaron los experimentos intentando que todos los computadores estuvieran en las mejores condiciones posibles hablando de: temperatura, nivel de estrés y procesos en segundo plano. Ahora bien, teniendo en cuenta los resultados obtenidos se llegó a la conclusión de que el lenguaje con mayor rapidez o efectividad a la hora de ordenar los vectores de números fue **Golang**, le sigue el lenguaje **Java**, y en tercer lugar está **Kotlin**.

Control y conclusiones finales

A pesar de que Golang fue el lenguaje con mayor efectividad para ordenar un vector numérico con un método de alta complejidad temporal, este mismo fue el más tedioso y contraproducente a la hora de codificar ya que el lenguaje posee un **fuerte tipado** lo cual generó que la creación del código fuera tardía y llena de contratiempos repentinos. Uno de los mejores lenguajes para programación Web y cumplir con el requisito del software requerido decidimos que pueden ser **Java** utilizando su extensión de **JavaServer**, o **JavaScript** ya que estos últimos son de mayor efectividad a la hora de programar ya que son más popularmente utilizados para proyectos o sistemas de información en la Web

Lenguaje

RAM

Lenguaje>Ram diferencia no muy notable