Universidad ICESI

Integrantes:
Paola Osorio - Bryan Guapacha
Sean Quintero - John K. Landazuri

Taller Experimentos
(algoritmos de ordenamiento)

Resumen.

Durante el experimento se logró explorar el rendimiento de dos de los algoritmos de ordenamientos más conocidos por los desarrolladores, estos son: el algoritmo “burbuja” e “inserción” teniendo en cuenta diferentes factores que pueden o no afectar el rendimiento de los algoritmos mencionados.

Introducción.

Llevando a cabo este experimento se analizó el rendimiento de dos algoritmos de ordenamiento (bubble y Insertion) los cuales fueron implementados en diferentes lenguajes de programación y teniendo en cuenta otros factores como por ejemplo el sistema operativos donde se ejecutan los algoritmos y el tamaño del arreglo (10^2 y 10^3) y realizando 100 repeticiones en cuanto a ejecución del algoritmo se refiere y así poder analizar y se presentaban cambios en el factor tiempo de ejecución.

Experimento

Esos cambios sobre la operación del proceso se definen a través de los FACTORES. Factores cuyo cambio en sus valores pueden llegar a afectar el proceso anterior podrían ser:

- 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)

Desempeño algoritmos de ordenamiento.

1. Unidad experimental

- **Conjunto de algoritmos de ordenamiento:**
  - Burbuja.
  - Inserción.

2. Factores controlables

- **Lenguajes:**
  - C#
  - JavaScript
  - Kotlin
  - GoLang
- **Tamaño del arreglo:**
  - 10^2
  - 10^3

3. Factores no controlables

- Equipo en que se ejecuta.
- Hardware en que se modela la ejecución del programa.
- Temperatura del equipo en que se ejecuta.
- Las buenas prácticas en el código, que permitan ejecución veloz en los lenguajes a usar.
- Sistema operativo.

4. Niveles

- **Lenguajes:**
  - javaScript= Bryan
  - c#= Paola
  - Golang = John
  - Kotlin=Sean
- **Tamaño del arreglo**
  - 10^2
  - 10^.4
- **Sistema Operativo**
  - Windows
  - Mac OS

Tratamientos.

Lenguaje	Tamaño del arreglo	Sistema operativo	Tratamiento
JavaScript	10^2	Windows	1
JavaScript	10^2	MacOs	2
JavaScript	10^3	Windows	3
JavaScript	10^3	MacOs	4
C#	10^2	Windows	5
C#	10^2	MacOs	6
C#	10^3	Windows	7

C#	10^3	MacOs	8
GoLang	10^2	Windows	9
GoLang	10^2	MacOs	10
GoLang	10^3	Windows	11
GoLang	10^3	MacOs	12
Kotlin	10^2	Windows	13
Kotlin	10^2	MacOs	14
Kotlin	10^3	Windows	15
Kotlin	10^3	MacOs	16

Se harán 100 repeticiones de cada tratamiento.

6. Variable de respuesta
   - Duración ejecución en milisegundos

ANÁLISIS DE COMPLEJIDAD TEMPORAL

Insertion sort:

```
public static void insertionSortImperative(int[] input) {
 for (int i = 1; i < input.length; i++) {
 int key = input[i];
 int j = i - 1;
while (j >= 0 && input[j] > key) {
 input[j + 1] = input[j];
 j = j - 1;
}
 input[j + 1] = key; }
 }
```

Instrucción	Veces que se repite (Big O)
1. for (int i = 1; i < input.length; i++) {	n
2. int key = input[i];	n-1
3. int j = i - 1;	n-1
4. while (j >= 0 && input[j] > key) {	(n*(n-1))/2
5. input[j + 1] = input[j]	((n*(n-1))/2)-1
6. j = j - 1;	((n*(n-1))/2)-1

7. input[j + 1] = key;	n-1
Total:	n^2

Bubble sort:

```
void bubbleSort(int arr[]) {
      int n = arr.length;
      for (int i = 0; i < n-1; i++)
         for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
            {
               int temp = arr[j];
               arr[j] = arr[j+1];
               arr[j+1] = temp;
            }
  }
```

Instrucción	Veces que se repite (Big O)
int n = arr.length;	n
for (int i = 0; i < n-1; i++)	n
for (int j = 0; j < n-i-1; j++)	(n*(n-1))/2
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
Total:	n^2

ANÁLISIS COMPLEJIDAD ESPACIAL

InsertionSort

```
public static void insertionSortImperative(int[] input) {
 for (int i = 1; i < input.length; i++) {
 int key = input[i];
 int j = i - 1;
while (j >= 0 && input[j] > key) {
 input[j + 1] = input[j];
 j = j - 1;
}
 input[j + 1] = key; }
 }
```

Tipo	Variable	Cantidad de valores atómicos
Entrada	input	n
Auxiliar	key	1
	i	1
	j	1
Salida		

Sea n = input
Complejidad Espacial Total = Entrada + Auxiliar + Salida = n + 1 + 1 + 1 = n + 3 = θ(n)
Complejidad Espacial Auxiliar = 1 + 1 + 1 = θ(1)
Complejidad Espacial Auxiliar + Salida = 1 + 1 + 1 = θ(1)

Bubble sort:

```
void bubbleSort(int arr[]) {
      int n = arr.length;
      for (int i = 0; i < n-1; i++)
         for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
            {
               int temp = arr[j];
               arr[j] = arr[j+1];
               arr[j+1] = temp;
            }
   }
```

Tipo	Variable	Cantidad de valores atómicos
Entrada	arr	n
Auxiliar	temp	1
	i	1
	j	1
Salida		

Sea n = arr
Complejidad Espacial Total = Entrada + Auxiliar + Salida = n + 1 + 1 + 1 = n + 3 = θ(n)
Complejidad Espacial Auxiliar = 1 + 1 + 1 = θ(1)
Complejidad Espacial Auxiliar + Salida = 1 + 1 + 1 = θ(1)

Resultado de tratamientos.

Las implementaciones realizadas fueron puestas en el siguiente excel:

https://docs.google.com/spreadsheets/d/1OjY5G-kOxTxVy1AWs7106htVGJPVK99DdDxOgayLvZs/edit?usp=sharing

CONCLUSIONES

Basados en los promedios de ejecución en ms del arreglo de tamaño 10^2

1. Se recomienda usar golang en cualquiera de los sistemas operativos, ya que su promedio es de o ms en arreglos tamaño 10^2

Basados en los promedios de ejecución en ms del arreglo de tamaño 10^3

2. Se recomienda usar golang en cualquiera de los sistemas operativos, ya que su promedio es de o ms en arreglos tamaño 10^3

Basado en los promedios generales de ejecución por lenguaje encontramos:

3. Golang es el lenguaje con mayor rapidez de ejecución con un promedio de 0.33625ms