TIEMPO DE VALOR ESPERADO EN ALGORITMOS DE ORDENACIÓN

Bollschweiler Vargas Ian Nicolas
Cabrejos Rafael Erika Epifania
Hidalgo Esquivel Jared Miguel
Del Rosario Sosa Joshua Jean Paul

Resumen

En este informe plantearemos la eficiencia de algunos algoritmos de ordenamiento, es decir, calcularemos el valor esperado de tiempo de ejecución para una entrada aleatoria de tamaño n. Este tema es importante ya que para diversas aplicaciones (como la industria de los videojuegos) los algoritmos de ordenamiento son muy utilizados y algunos aspectos (como la popularidad del juego) se verán afectados por la eficiencia del algoritmo (un juego que se "cuelga" mucho podría desanimar a sus seguidores).

1 INTRODUCCIÓN

Los algoritmos de ordenamiento, como bien indica su nombre, nos permiten ordenar, es decir darle a cada objeto(o dato) una posición determinada según cierto criterio específico.
En este caso, nos enfocaremos en ordenar un conjunto de datos, datos como: Vectores o matrices.
En este informe nos enfocaremos en el estudio de algunos de los algoritmos de ordenamiento más utilizados, analizando la cantidad de comparaciones que se dan, el tiempo que toma cada comparación y la extensión de su código, para cada algoritmo analizado. Este informe nos permitirá conocer y comprender más a fondo cada uno de los métodos de ordenamiento analizados, desde los más simples hasta los más complejos. Se realizaran comparaciones en tiempo de ejecucion, pre-requisitos de cada algoritmo, funcionalidad, alcance, entre otros. Gracias a los conocimientos obtenidos en clase se determinará la esperanza de cada algoritmo, se hará uso de criterios como la media y la mediana para este objetivo. a su vez, daremos una vista gráfica de los resultados obtenidos usando lenguaje R para graficar los tiempos y otros datos obtenidos de los algoritmos de ordenamineto.

2 ESTADO DEL ARTE

- **Performance analysis of Sorting Algorithms:**
  En esta tesis el autor realiza una vista general sobre los algoritmos de ordenación y un análisis de sus respectivos rendimientos .
- **Run-Time Analysis for sorting algorithms:**
  Este artículo trata sobre la evaluación de 3 algoritmos de ordenación en 3 distintos lenguajes de programación para aproximar su complejidad .

3 DISEÑO DEL EXPERIMENTO

Para encontrar el tiempo de valor esperado, implementaremos 5 algoritmos de ordenamiento, siendo estos: Merge sort, Bubble sort, Quick sort, Insertion sort y Bucket sort. Estos serán implementados en lenguaje R donde ordenarán un arreglo de números aleatorios de n elementos. Este proceso se realizará 1000 veces y se tomarán los tiempos de ejecución de cada algoritmo para hallar una distribución en intervalos para cierta cantidad n de elementos. Después, se procederá a encontrar la esperanza del tiempo de ejecución a través de un cálculo matemático de los datos para distintos valores de n. Finalmente, se procederá a realizar una gráfica con los tiempos esperados de cada algoritmo para diferentes tamaños de entrada n.
Los pseudocódigos de los algoritmos a utilizar son:

Algoritmo 1: Merge Sort

Entrada: Array A,p,r
Salida : Array A ordenado

```
1 if p < r then
2 |   q = ⌊(p+r)/2⌋;
3 |   MergeSort(A,p,q);
4 |   MergeSort(A,q+1,r);
5 |   Merge(A,p,q,r);
  |   // Merge junta los arreglos
  |   A[p..q] y A[q+1..r] que se
  |   encuentran ordenados
6 end
```

Algoritmo 2: Bubble Sort

Entrada: Array A con n elementos
Salida : Array A ordenado

```
1 for i ← 1 to n do
2 |   for j ← 1 to n do
3 |   |   if A[j] > A[j+1] then
4 |   |   |   Intercambiar(A[j], A[j+1]);
5 |   |   end
6 |   end
7 end
```

Algoritmo 3: Quick Sort

Entrada: Array A,p,r
Salida : Array A ordenado

```
1 if p<r then
2 |   q = Partición(A,p,r);
  |   // La función partición reordena
  |   el arreglo A[p..r]
  |   QuickSort(A,p,q-1);
3 |   QuickSort(A,q+1,r);
4 end
```

Algoritmo 4: Insertion Sort

Entrada: Array A con n elementos
Salida : Array A ordenado

```
1 for j ← 2 to n do
2 |   llave = A[j];
3 |   i = j − 1;
4 |   while i>0 y A[i] > llave do
5 |   |   A[i + 1] = A[i];
6 |   |   i = i − 1;
7 |   end
8 |   A[i + 1] = llave;
9 end
```

Algoritmo 5: Bucket Sort

Entrada: Array A con n elementos
Salida : Array A ordenado

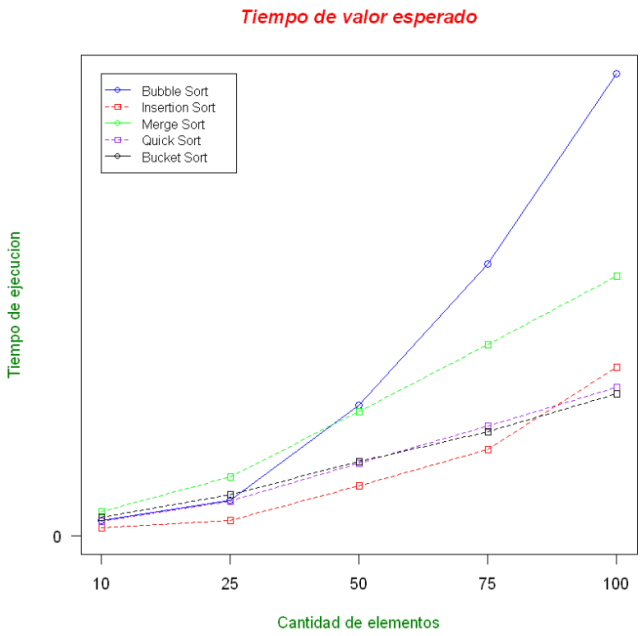
```
1  Sea B[0..n − 1] un nuevo array;
2  for j ← 0 to n − 1 do
3  |   Hacer B[i] una lista vacia;
4  end
5  for i ← 1 to n do
6  |   Insertar A[i] en la lista B[⌊nA[i]⌋];
7  end
8  for i ← 0 to n − 1 do
9  |   Ordenar la lista B[i] usando el Insertion
   |    Sort;
10 end
11 Concatenar las listas B[0], B[1]...B[n − 1] en
    orden;
```

4 EXPERIMENTOS Y RESULTADOS

Se realizó el ordenamiento para 10,25,50,75 y 100 elementos, obteniendo los siguientes tiempos de valor esperado:

	Bubble Sort	Insertion Sort	Merge Sort	Quick Sort	Bucket Sort
10	6.165755e-05	3.219223e-05	9.523218e-05	5.941045e-05	7.414262e-05
25	0.0001428318	6.084943e-05	0.0002352301	0.0001396892	0.0001627132
50	0.0005186403	6.084943e-05	0.000495109	0.0002891735	0.0002960197
75	0.001080035	0.0003438518	0.0007599532	0.0004373971	0.0004138448
100	0.001835479	0.0006710175	0.001031311	0.0005909096	0.0005635857

Luego con los datos obtenidos de la tabla, generamos la siguiente gráfica:



5 DISCUSIÓN

Podemos observar que el algoritmo más rápido es un empate entre el Quick Sort y el Bucket Sort, seguidos del Insertion Sort, el Merge Sort y finalmente el Bubble Sort. Notamos que el crecimiento del Insertion Sort es mayor que el del Merge Sort por lo que para una mayor cantidad de datos el Merge Sort sería más eficiente que el Insertion Sort.

6 CONCLUSIONES

Se obtuvo los valores esperados de los algoritmos de ordenamiento más conocidos, también hemos podido analizar la eficiencia de cada uno de ellos concluyendo que los mejores algoritmos serían Quick Sort y el Bucket Sort.

7 BIBLIOGRAFÍA O REFERENCIAS

$https://www.bowdoin.edu/\ Itoma/teaching/cs231/duke_cps130/Lectures/L06.pdf$
$https://www.ijcsmc.com/docs/papers/January2015/V4I1201557.pdf$
$https://pdfs.semanticscholar.org/186b/18407989d49e8e88fc2f7b46c7a2a376afd1.pdf$