# Práctica 7: -

**Hernández Castellanos César Uriel, Aguilar Garcia Mauricio**

Escuela Superior de Cómputo
Instituto Politécnico Nacional, México
*uuriel*12009*u*@*gmail.com*, *mauricio.aguilar.garcia*,90@*gmail.com*

**Resumen: Palabras Clave: Complejidad, Algoritmo, Ordenamiento, BucketSort, BogoSort y CocktailSort**

# 1. Introducción

En computación y matemáticas un algoritmo de ordenamiento es un algoritmo que pone elementos de una lista o un vector en una secuencia dada por una relación de orden, es decir, el resultado de salida ha de ser una permutación —o reordenamiento— de la entrada que satisfaga la relación de orden dada. Las relaciones de orden más usadas son el orden numérico y el orden lexicográfico. Ordenamientos eficientes son importantes para optimizar el uso de otros algoritmos (como los de búsqueda y fusión) que requieren listas ordenadas para una ejecución rápida. También es útil para poner datos en forma canónica y para generar resultados legibles por humanos.[4]

Desde los comienzos de la computación, el problema del ordenamiento ha atraído gran cantidad de investigación, tal vez debido a la complejidad de resolverlo eficientemente a pesar de su planteamiento simple y familiar. Por ejemplo, BubbleSort fue analizado desde 1956. Aunque muchos puedan considerarlo un problema resuelto, nuevos y útiles algoritmos de ordenamiento se siguen inventado hasta el día de hoy (por ejemplo, el ordenamiento de biblioteca se publicó por primera vez en el 2004). Los algoritmos de ordenamiento son comunes en las clases introductorias a la computación, donde la abundancia de algoritmos para el problema proporciona una gentil introducción a la variedad de conceptos núcleo de los algoritmos, como notación de O mayúscula, algoritmos divide y vencerás, estructuras de datos, análisis de los casos peor, mejor, y promedio, y límites inferiores.

# 2. Conceptos básicos

## Ordenamiento por casilleros

El ordenamiento por casilleros (bucket sort o bin sort, en inglés) es un algoritmo de ordenamiento que distribuye todos los elementos a ordenar entre un número finito de casilleros. Cada casillero sólo puede contener los elementos que cumplan unas determinadas condiciones.[1]

Después cada uno de esos casilleros se ordena individualmente con otro algoritmo de ordenación (que podría ser distinto según el casillero), o se aplica recursivamente este algoritmo

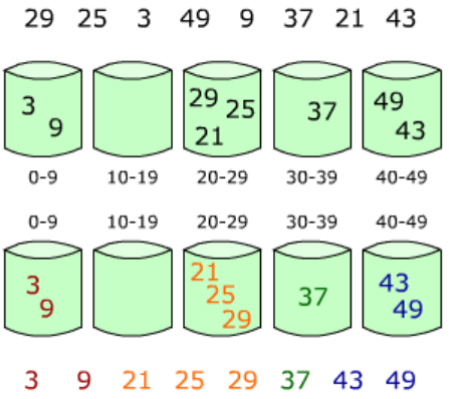


Figura 1: Ejemplo de ordenamiento por casilleros

## BogoSort

Stupid Sort, en inglés también conocido como BogoSort, es un algoritmo de búsqueda particularmente inefectivo basado en el paradigma de ensayo y error.
Consiste en posicionar los elementos de manera aleatoria hasta obtener el ordenamiento del arreglo en cuestión.[2]

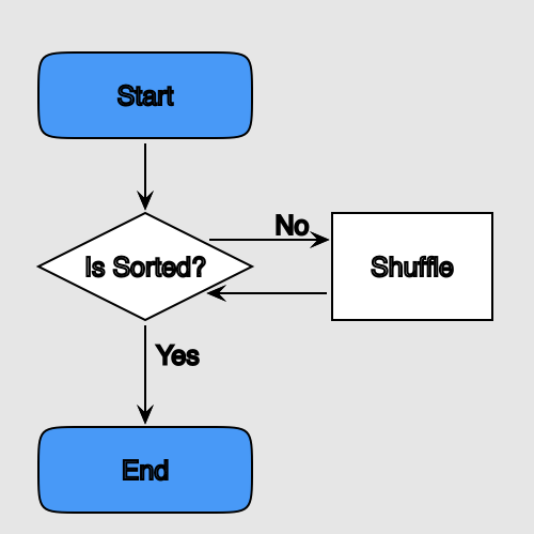


Figura 2: Diagrama de flujo de BogoSort

## 2.1. CocktailSort

El ordenamiento de burbuja bidireccional (cocktail sort en inglés) es un algoritmo de ordenamiento que surge como una mejora del algoritmo ordenamiento de burbuja.[3]

La manera de trabajar de este algoritmo es ir ordenando al mismo tiempo por los dos extremos del vector. De manera que tras la primera iteración, tanto el menor como el mayor elemento estarán en sus posiciones finales. De esta manera se reduce el número de comparaciones aunque la complejidad del algoritmo sigue siendo O($n^2$).

Hacemos un recorrido ascendente (del primer elemento al último), cogemos el primer elemento y lo comparamos con el siguiente, si el siguiente es menor lo pasamos al puesto anterior, de esta forma al final de la lista nos queda el mayor. Una vez terminada la serie ascendente, hacemos un recorrido descendente (del último elemento al primero) pero esta vez nos quedamos con los menores a los que vamos adelantando posiciones en vez de retrasarlas como hicimos en la serie ascendente. Repetimos las series alternativamente pero reduciendo el ámbito en sus extremos pues ya tendremos allí los valores más bajos y más altos de la lista, hasta que no queden elementos en la serie.

# 3. Experimentación y Resultados

1. Elija tres algoritmos de ordenación (no vistos en clase).

   i) Mediante ejemplos, muestre el funcionamiento de los algoritmos que se eligieron.
   ii) Compare el orden de complejidad en el mejor de los casos de manera experimental mediante gráficas de sus resultados.
   iii) Compare el orden de complejidad en el peor de los casos de manera experimental mediante gráficas de sus resultados.

### Funcionamiento de Bucket Sort

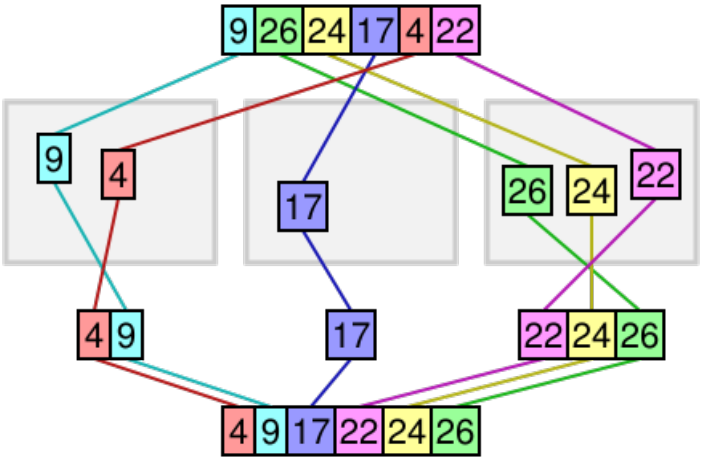


Figura 3: Ejemeplo de funcionamiento de BucketSort

Supongamos que se tiene el arreglo unidimensional 9,26,24,17,4,22, el algoritmo de ordenamiento por casilleros funciona de la siguiente manera:

Se determinan el tamaño del casillero de la siguiente manera:

$$tamanioDelCasillero = \frac{Superior - Inferior}{n}$$

Donde n es el número de casilleros que se desean crear, superior el número más grande y pequeño que puede tomar el arreglo (Se podría tomar el máximo número entero y el minimo como superior e inferior).
Posterior a que se determinan el número de casilleros a se crea una estructura de tipo casillero. Luego cada casillero almacenará a los datos con caracteristicas similares, como por ejemplo puede ser un rango de números u alguna otra caracteristicas si hablaramos de programación orientada a objetos.
Como penúltimo paso se ordenaran cada uno de los casilleros con algún otro algoritmo de ordenamiento o llamando recursivamente al algoritmo de ordenamiento por casileros.
Finalmente se integran todos los casilleros en el arreglo, obteniendo de esta manera nuestro arreglo ordenado.
En la figura 3 se muestra un ejemplo de BucketSort, donde se tienen tres casilleros en el que cada casillero contiene números que se encuentran en un cierto rango para cada casillero 9 y 4 para el primer casillero, 17 para el segundo y 26,24 y 22 para el último casillero, luego cada casillero se ordena para finalmente obtener el arreglo de interés ordenado. Ejemeplo de funcionamiento de BucketSort

## Ejemplo del funcionamiento de BogoSort

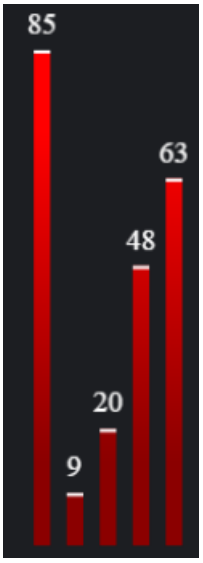


Figura 4: Arreglo desordenado

En la figura 4 se muestra un arreglo desordenado, el cual será ordenado por BogoSort que consiste básicamente en posicionar los elementos del arreglo de manera aleatoria hasta obtener el arreglo ordenado.

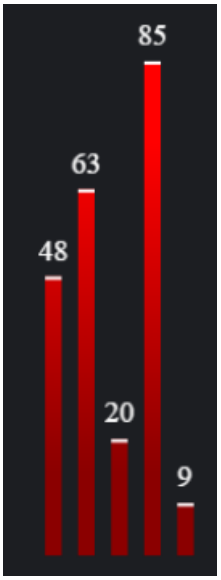


Figura 5: BogoSort en la primera iteración

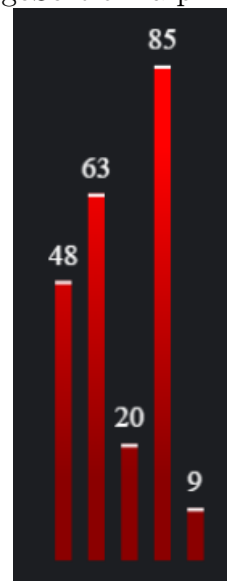


Figura 6: BogoSort en la segunda iteración

Como es posible observar el algoritmo de BogoSort es altamente ineficaz, ya que no existe certeza de cuando finalizará.

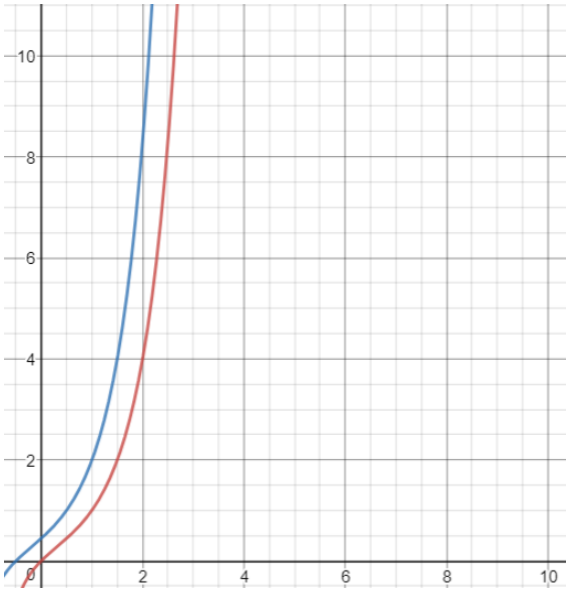


Figura 7: BogoSort en el peor de los casos $\theta$(n n!) (Función que se representa con el color rojo)

En la figura 7 es posible apreciar la complejidad de Bogosort en el peor de los casos que nos resultó ser $\theta$(n n!) y por último se indica la función que acota por arriba a la función f(n) = n n! la cual se propuso como f1(n) = (n+0.5) (n+0.5)! (función representada de color azul)

| n | k |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 4 |
| 3 | 18 |
| 4 | 96 |
| 5 | 600 |
| 6 | 4320 |
| 7 | 35280 |
| 8 | 322560 |

Figura 8: Pares de coordenadas obtenidos de manera experimental

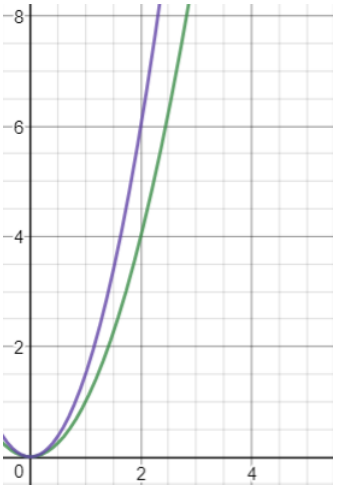


Figura 9: Bucketsort en el peor de los casos es $\theta(n^2)$

| n | k |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |
| 5 | 25 |
| 6 | 36 |
| 7 | 49 |
| 8 | 64 |
| 9 | 81 |
| 10 | 100 |

Figura 10: Bucketsort en el peor de los casos es $\theta(n^2)$

En la figura 9 se muestra la complejidad del algoritmo de BucketSort en el peor de los casos que nos resultó ser $\theta(n^2)$, se acotó por arriba con la función f(n)=$1.5n^2$, por último en la figura 10 se muestra los pares de coordenadas obtenidos experimentalmente.

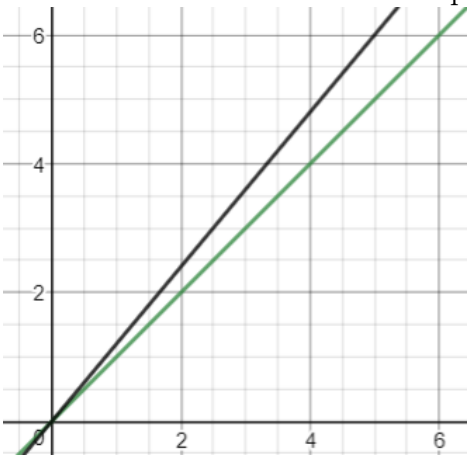


Figura 11: BogoSort en el mejor de los casos $\theta$(n) (Función que se representa de color rojo)

En la figura 11 se muestra la complejidad del algoritmo de Bogosort en su mejor caso que nos resultó lineal,

| n | k |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |
| 3 | 4 |
| 4 | 5 |
| 5 | 6 |
| 6 | 7 |
| 7 | 8 |
| 8 | 9 |

Figura 12: Tabla con los pares de coordenadas obtenidos de manera experimental

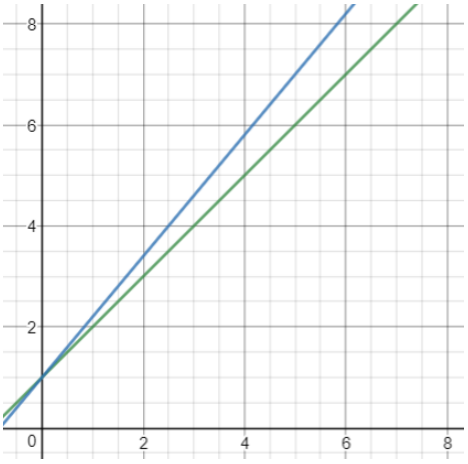


Figura 13: Bucket en el mejor de los casos $\theta$(n+k) (Función que se representa de color verde)

## 3.1. CocktailSort

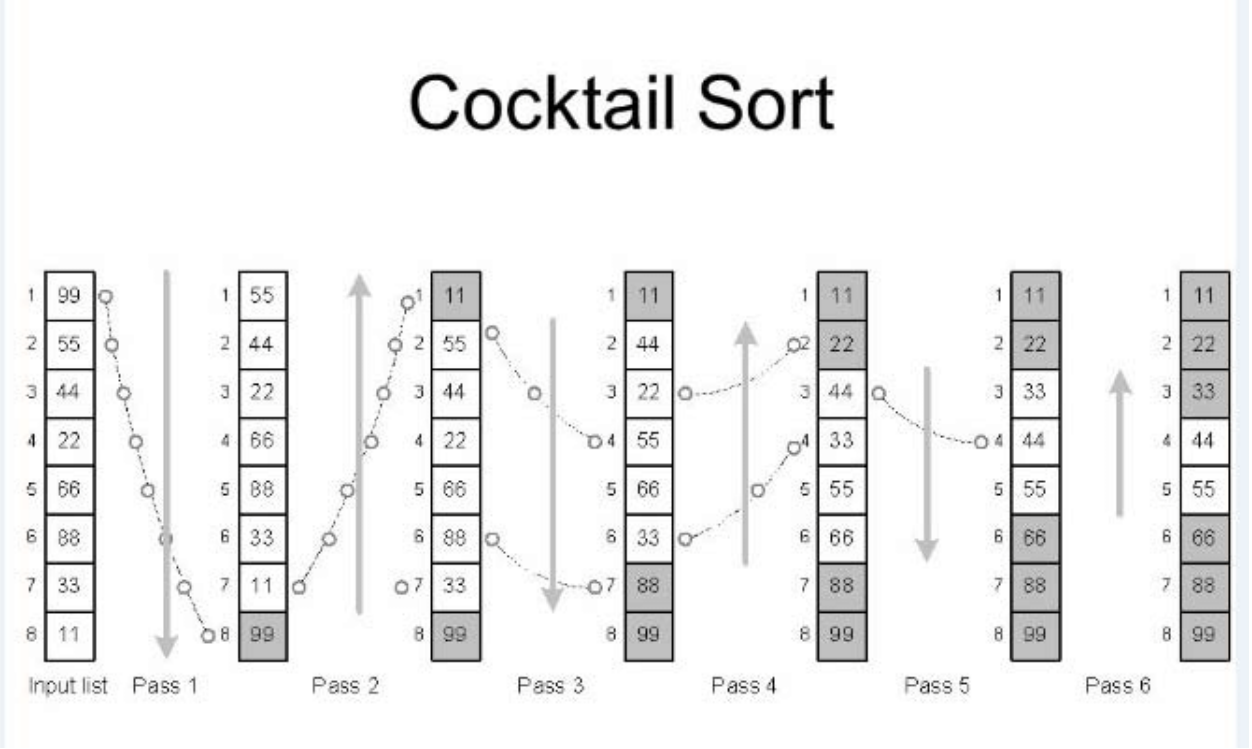


Figura 14: Ejemplo cocktailSort

En la figura 14 se puede apreciar el como va avanzando el cocktailsort siendo que empieza desde el indice uno y termina hasta el tamaño completo del arreglo y en el cual se va a reduciendo en uno cada vez que va a una direccion en especifico hacinedo el metodo del bubblesort en estos casos solo que no regresa al inicio si no que toma otras condiciones para ordenar.

| | |
|---|---|
| 5 | 21 |
| 10 | 65 |
| 15 | 167 |
| 20 | 228 |
| 40 | 1200 |
| 50 | 1768 |
| 75 | 4666 |
| 100 | 7435 |
| 50 | 1768 |
| 100 | 7435 |

Figura 15: Salida casos aleatorios del CocktailSort

En la figura 15 se puede tomar como promedio que tiende a una elevación cercana a cuadratica pero un poco menor

| | |
|---|---|
| 5 | 10 |
| 10 | 20 |
| 15 | 30 |
| 20 | 40 |
| 40 | 80 |
| 50 | 100 |
| 75 | 150 |
| 100 | 200 |

Figura 16: Salida mejor caso CocktailSort

En la figura 16 se puede notar que la salida de CokctailSort tiene a ser del tipo *theta*(2*n) en el cual su mejor caso es cuando esta completamente ordenado

| | |
|---|---|
| 5 | 25 |
| 10 | 100 |
| 15 | 225 |
| 20 | 400 |
| 40 | 1600 |
| 50 | 2500 |
| 75 | 5625 |
| 100 | 10000 |

Figura 17: Salida peor caso CocktailSort

En la figura 17 se puede apreciar que tiene fines cuadraticos la salida en el cual el peor caso es que esten ordenados inversamente ya sea de mayor a menor o viceversa.

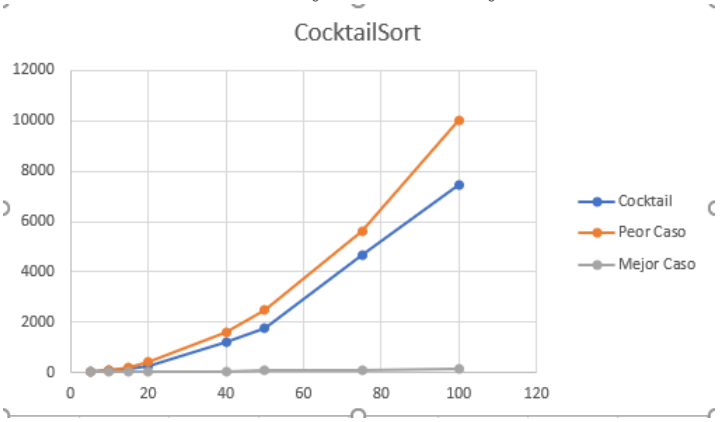


Figura 18: Comparación salidas delCocktailSort

En la figura 18 se puede apreciar el como regularmente se acerca mas al peor caso ya que al tener numeros aleatorios en el arreglo es dificil que esten ordenados completamente.

# 4. Conclusiones

## Conclusión general

Los métodos de ordenamiento de datos son muy útiles, ya que la forma de arreglar los registros de una tabla en algún orden secuencial de acuerdo a un criterio de ordenamiento, el cual puede ser numérico, alfabético o alfanumérico, ascendente o descendente. Nos facilita las búsquedas de cantidades de registros en un moderado tiempo, en modelo de eficiencia. Mediante sus técnicas podemos colocar listas detrás de otras y luego ordenarlas, como también podemos comparar pares de valores de llaves, e intercambiarlos si no se encuentran en sus posiciones correctas.
Existen múltiples algoritmos de ordenamiento con diferente complejidad, siendo el ganador hasta el momento QuickSort

## Aguilar Garcia Mauricio

Descubrimos que existen muchos algoritmos para ordenar arreglos, y se descubrio que algunos no cambian su complejidad en los casos o es muy parecida a su peor caso la mayoria de las veces, y en el ultimo algoritmo se uso como base uno ya existente para la creación de otro el caso del CocktailSort.

## Hernández Castellanos César Uriel

Existen múltiples algoritmos para ordenar un arreglo de datos, particularmente se revisaron tres algortimos (Bucket Sort, BogoSort y CocktailSort) de los cuales se obtuvo de manera experimental la complejidad de cada uno de ellos, siendo el de peor rendimiento el BogoSort.

# 5. Bibliografía

[1]GeeksforGeeks. (2018). Bucket Sort - GeeksforGeeks. [online] Available at: https://www.geeksforge
sort-2/ [Accessed 18 Oct. 2018].

[2].En.wikipedia.org. (2018). Bogosort. [online] Available at: https://en.wikipedia.org/wiki/Bogosort
[Accessed 18 Oct. 2018].

[3]GeeksforGeeks. (2018). Cocktail Sort - GeeksforGeeks. [online] Available at: https://www.geeksforg
sort/ [Accessed 18 Oct. 2018].

[4]Es.wikipedia.org. (2018). Algoritmo de ordenamiento. [online] Available at: https://es.wikipedia.org