Reporte de practica: Algoritmos de Ordenamiento

Nombre: Diana Carolina Lozoya Pérez **Matricula:**1725012

Materia: Matemáticas Computacionales Grupo:001 **Fecha:** 17/09/17

Resumen: Este reporte tiene como objetivo describir el funcionamiento de cuatro algoritmos de ordenamiento, en este caso unos de los más comunes, se mencionará su definición y características importantes de cada uno de ellos, ya que se tiene como propósito programar a futuro alguno de estos, por lo cual, en la definición de cada uno, se mostrará un código programado personalmente.

1. Bubble

El algoritmo de Ordenamiento Burbuja o Bubblesort es un algoritmo que trabaja sobre un arreglo de números, tomando el último elemento y comparándolo con su anterior y si hubiese cambio los ordena, se le llama burbuja por que hace las comparaciones de tal manera que asemejan a burbujas subiendo desde el fondo del agua.
También es conocido como el método del intercambio directo dado que solo usa comparaciones para operar elementos.
El esquema de la implementación es esta:
1° Ingresa Arreglo de números
2° La función Burbuja los ordena
3° Se Muestra el Arreglo de números
El algoritmo burbuja en su implementación usa una variable auxiliar para hacer el intercambio en caso el dato anterior sea mayor. Su complejidad es de O(n^2), donde O representa la complejidad asintótica.

Bubble en Python:

```
#Algoritmo de ordenacion Bubble

#declaracion de funciones
contador=0
def burbuja(A):
        global contador
        for i in range(1,len(A)):
                for j in range(0,len(A)-1):
                        contador+=1
                        if(A[j+1]<A[j]):
                                aux=A[j]
                                A[j]=A[j+1]
                                A[j+1]=aux
        print(A)

#programa principal
A=[6,5,3,1,8,7,2,4]
print(A)
burbuja(A)
print(contador)
```

2. Insertion

El ordenamiento por inserción es una manera muy natural de ordenar para el ser humano y puede usarse fácilmente para ordenar un mazo de cartas numeradas, secuencias de números, de forma arbitraria.

Su idea principal consiste en ir insertando un elemento de la lista o un arreglo en la parte ordenada de la misma, asumiendo que el primer elemento es la parte ordenada de la misma, el algoritmo ira comparando un elemento de la parte desordenada de la lista con los elementos de la parte ordenada, insertando el elemento en la posición correcta dentro de la parte ordenada. Y así sucesivamente hasta obtener la lista ordenada.

Su complejidad al igual que Bubble es de O(n^2) donde O es el grado de complejidad asintótica, además, a comparación de bubble es más eficaz en su método de ordenación.

Insertion en Python:

```
#insertion

#definirInsertion
contador=0
def insercion(arreglo):
        global contador
        for indice in range(1,len(arreglo)):
                valor=arreglo[indice] #valor es el elemento que vamos a comparar
                i=indice-1 #i es el valor anterior al elemento que estamos comparando
                while i>=0:
                        contador+=1
                        if valor<arreglo[i]: #compraramos valor con el elemento anterior
                                arreglo[i+1]=arreglo[i] #intercambiamos los valores
                                arreglo[i]=valor
                                i-=1 #decremento en 1 el valor de i
                        else:
                                break
        return arreglo


A=[20,34,53,12,11,3,45,76,89,100,24,201]
print(A)
insercion(A)
print(A)
print(contador)
```

3. Selection

Es un algoritmo que consiste en encontrar el menor de todos los elementos del arreglo o vector e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño y así sucesivamente hasta ordenarlo todo. Su implementación requiere de O(n^2) comparaciones e intercambios para ordenar una sentencia de elementos. En pocas palabras se basa en la selección sucesiva de los valores mínimos o máximos.

Este algoritmo mejora ligeramente a Bubble, sin embargo, se considera lento y poco eficiente cuando se usa en listas grandes o medianas ya que realiza numerosas comparaciones.

Selection en Python:

```
#Algoritmo de ordenamiento selection
c=0 #declaracion de variable contador

#definir el algoritmo
def selection(lis):
        global c
        for i in range(0,len(lis)-1):
                v=i
                for j in range(i+1,len(lis)):
                        c= c+1
                        if lis[j]<lis[v]:
                                v=j
                if v !=-1:
                        aux=lis[i]
                        lis[i]=lis[v]
                        lis[v]=aux
        return c

#declaracion de una funcion para que nos de un arreglo de valores aleatorios del 0 la 100

import random
#definicion de la funcion
def ran_n(n,lim_i=0,lim_s=100):
        lis=[]
        for i in range(n):
                lis.append(random.randint(lim_i,lim_s))
        return lis


#programa principal
A=ran_n(12) #generamos un arreglo
print(A) #mostramos el arreglo generado
selection(A) #aplicamos el algoritmo al arreglo
print(A) #mostramos el arreglo ordenado
print(c) #mostramos el numero de iteraciones durante el cilo (cuantas veces se compraro)
```

4. QuickSort

El ordenamiento rápido (quicksort) es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo proporcional a n log n. En el caso promedio, el orden es O(n·log n). El algoritmo fundamental es el siguiente:

1°Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.

2°Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.

3°La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.

4°Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

Quicksort en Python:

```python
#Algoritmo de ordenamiento quicksort

c=0 #declaracion de variable contador
#def de quicksort
def quicksort(x):
    global c
    if len(x) == 1 or len(x) == 0:
        return x
    else:
        c+=1
        pivot = x[0]
        i = 0
        for j in range(len(x)-1):
            if x[j+1] < pivot:
                x[j+1],x[i+1] = x[i+1], x[j+1]
                i += 1
        x[0],x[i] = x[i],x[0]
        first_part = quicksort(x[:i])
        second_part = quicksort(x[i+1:])
        first_part.append(x[i])
        return first_part + second_part

#Programa Principal
alist = [54,26,100,17,77,31,44,55,20]
print(alist)#arreglo inicial
quicksort(alist)#implementando quicksort
print(alist)#arreglo ordenado
print(c) #numero de iteraciones
```

5. Conclusiones

Los distintos tipos de algoritmos de ordenamiento nos ayudan para efectuar una misma tarea de diferentes maneras.
De manera personal, al programar estos algoritmos he notado algunas características:

- Bubble me pareció el más fácil de implementar, su funcionamiento es ir comparando un elemento con el resto, de manera individual. Esto quiere decir que recorre el arreglo n veces por n veces (n^2), donde n es la longitud del arreglo. Además, bubble no requiere de almacenamiento temporal, reduciendo así el espacio a utilizar. La principal desventaja de bubble es que no es muy eficiente con arreglos grandes debido a su complejidad.
- Insertion analiza repetidamente la lista de elementos, cada vez insertando el elemento en secuencia desordenada, en su posición correcta. Al igual que bubble es fácil su implementación, aunque en código es un poco más extenso, similarmente ofrece un buen rendimiento cuando nuestro arreglo es pequeño, aunque también requiere un espacio mínimo, su desventaja es que no funciona tan bien como otros algoritmos, comparándolo con bubble en código es solo un poco más extenso, pero en complejidad son iguales. Es decir, con n^2 pasos requeridos para cada n elemento a ser ordenado.
- Selection, funciona pasando repetidamente por la lista de elementos, cada vez seleccionando un elemento según su ordenamiento y colocándolo en la posición correcta en el arreglo. Una de sus ventajas al igual que los dos algoritmos anteriores, es que funciona y da un rendimiento bueno con arreglos pequeños, no se necesita un almacenamiento temporal adicional mas allá, solo mantener la lista original, una de sus desventajas es su funcionamiento con arreglos enormes. Al igual que bubble e insertion, selection tiene su grado de complejidad definido como n^2. Lo que lo hace similar a los demás en cuanto a rendimiento, sin embargo, en estructura y forma de trabajar son diferentes.
- Quicksort a diferencia de estos tres que son muy parecidos, funciona según el famoso principio de ‘’divide y vencerás’’, primero se divide la lista en dos sublistas, se toma un elemento que es llamado pivote y a base de él, las dos sublistas generadas son comparadas y acomodadas. Este proceso se repite repentinamente en todas las sublistas que se generen. De manera personal creo que es el mejor algoritmo de los 4, ya que, si hablamos en términos de eficiencia, la complejidad de quicksort es de nlog(n), lo cual es evidentemente menor a n^2, generando buenos resultados si lo es utilizado con un arreglo enorme. Debido a que ordena en el lugar tampoco requiere un almacenamiento adicional.

6. Gráfica

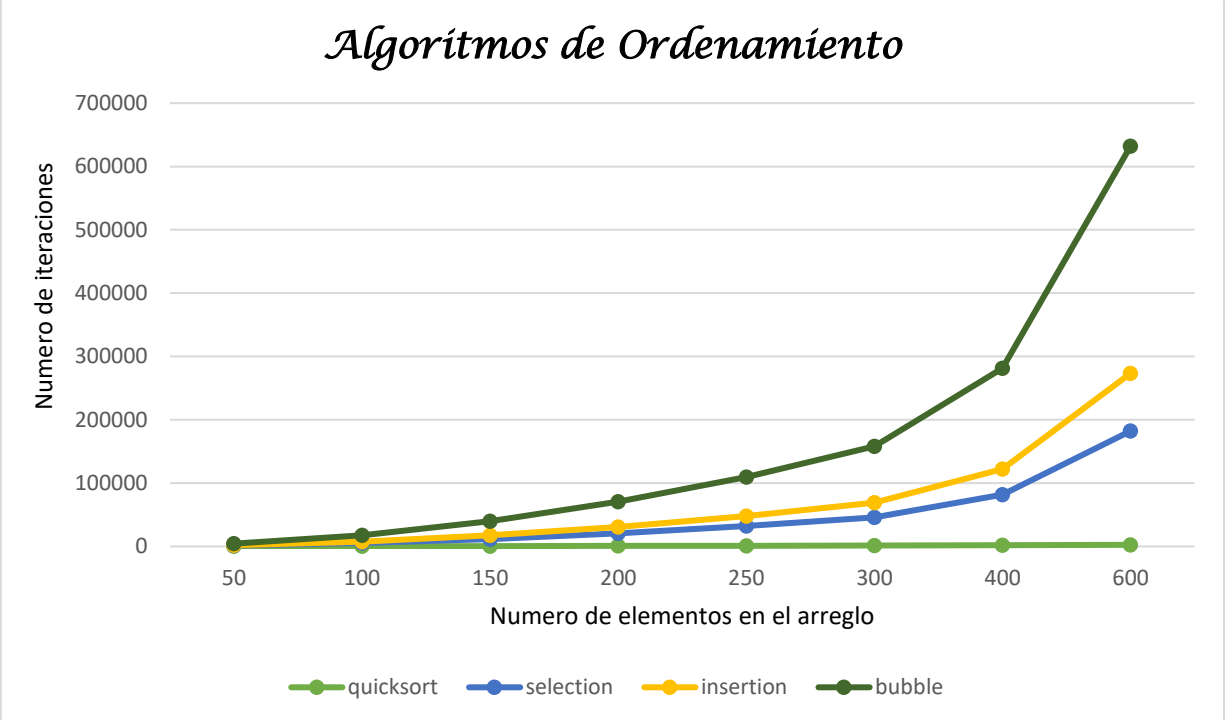
A continuación, se mostrará el rendimiento de cada uno de los algoritmos de manera gráfica con ayuda de Excel.

La tabla muestra resultados de 8 pruebas que se realizaron probando cada algoritmo de ordenamiento con el mismo arreglo en cada prueba, se compararon los elementos indicados en la siguiente tabla. Los números obtenidos representan a la variable contador que itera cada que se hace la comparación en cada uno de los algoritmos.

TABLA DE RESULTADOS EN 4 PRUEBAS

Algoritmos	Número de elementos en el arreglo							
	50	100	150	200	250	300	400	600
quicksort	149	336	500	834	969	1139	1774	2489
selection	1225	4950	11175	19900	31125	44850	79800	179700
insertion	566	2290	5697	10096	15584	22868	40590	91179
bubble	2401	9801	22201	39601	62001	89401	159201	358801

GRÁFICA DE COMPARACIÓN ENTRE LOS 4 ALGORITMOS Y SUS ITERACIONES



Las gráficas comprueban que Quicksort es más eficiente que los otros 3. Cabe destacar que los resultados obtenidos dependen del arreglo que genere el programa, ya que son arreglos con números aleatorios, la variable c (el contador) puede diferir en algunas cifras, por lo anteriormente dicho.