Reporte de practica: Algoritmos de Ordenamiento

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

Materia: Matemáticas Computacionales Grupo:001 **Fecha:** 01/09/17

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 la sintaxis adecuada para que su programación sea exitosa.

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.

Su sintaxis en Python es:

```
def ordenamientoBurbuja(lista,tam):
        for i in range(1,tam):
                for j in range(0,tam-i):
                        if(lista[j] > lista[j+1]):
                        k = lista[j+1]
                        lista[j+1] = lista[j]
                        lista[j] = k;

def imprimeLista(lista,tam):
        for i in range(0,tam):
                print lista[i]

def leeLista():
        lista=[]
        cn=int(raw_input("Cantidad de numeros a ingresar: "))
        for i in range(0,cn):
                lista.append(int(raw_input("Ingrese numero %d : " % i)))
        return lista

A=leeLista()
ordenamientoBurbuja(A,len(A))
imprimeLista(A,len(A))
```

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.

Su sintaxis en Python es:

```
def insercion(lista,tam):
        for i in range(1,tam):
                v=lista[i]
                j=i-1
                while j>=0 and lista[j]>v:
                        lista[j+1]=lista[j]
                        j=j-1
                lista[j+1]=v
def imprime(lista,tam):
        for i in range(0,tam)
                print lista[i]
def leerlista():
        lista=[]
        cn=int(raw_imput(‘’cantidad de números a ingresar: ‘’))
        for i in range(0,cn):
                lista.append(int(raw_imput(‘’Ingrese numero %d: ’’%i)))
                return lista
A=leelista()
insercion(A,len(A))
imprimelista(A,len(A))
```

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.

Su sintaxis en Python es:

```
def selection(lista,tam):
        for i in range (0,tam-1):
                min=i
                for j in rango(i+1,tam):
                        if (lista[min]>lista[j]:
                                min=j
                aux=lista[min]
                lista[min]=lista[i]
                lista[i]=aux

def imprimeLista(lista,tam):
        for i in range(0,tam):
                print lista[i]

def leeLista():
        lista=[]
        cn=int(raw_imput(‘’cantidad de números a ingresar: ‘’))
        for i in range(0,cn):
                lista.append(int(raw_imput(‘’Ingrese numero %d: ’’%i)))
                return lista

A=leeLista()
selection(A,len(A))
imprimeLista(A,len(A))
```

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.

Su sintaxis en Python es:

```
def quicksort(lista,izq,der):
    i=izq
    j=der
    x=lista[(izq + der)/2]

    while( i <= j ):
        while lista[i]<x and j<=der:
            i=i+1
        while x<lista[j] and j>izq:
            j=j-1
        if i<=j:
            aux = lista[i]; lista[i] = lista[j]; lista[j] = aux;
            i=i+1;  j=j-1;

        if izq < j:
        quicksort( lista, izq, j );
    if i < der:
        quicksort( lista, i, der );

def imprimeLista(lista,tam):
    for i in range(0,tam):
        print lista[i]

def leeLista():
    lista=[]
    cn=int(raw_input("Cantidad de numeros a ingresar: "))

    for i in range(0,cn):
        lista.append(int(raw_input("Ingrese numero %d : " % i)))
    return lista

A=leeLista()
quicksort(A,0,len(A)-1)
imprimeLista(A,len(A))
```

5. Conclusiones

Los distintos tipos de algoritmos de ordenamiento nos ayudan para efectuar una misma tarea de diferentes maneras, quizá si los comparamos en rapidez no notaríamos gran diferencia, ya que ejecutan rápidamente las instrucciones y es cuestión de segundos para visualizar su resultado. Sin embargo, considero que la diferencia entre estos cuatro son las líneas de código para cada uno, además de su grado de complejidad, que es lo que finalmente nos ayuda a deducir cual es más eficaz. No esta demás mencionar que cada uno ofrece un acomodo diferente de la información o los datos que se le proporcionan, pero al final el resultado es el mismo.

Aunque algunos algoritmos no son tan eficaces si los probamos con cantidades grandes, otros pueden ayudarnos fácilmente.

Estos 4 algoritmos básicos, nos ayudaran en un futuro a realizar tareas específicas, ya que el objetivo principal es programar nuestro propio bubble,insertion,Quicksort o selection, de manera que nos sea útil.