Universidad Autónoma de Nuevo León

Facultad de Ciencias Físico Matemáticas

Alumna: **Mayra Alejandra Rivera Rodríguez**

Matricula: **1742899**

Matemáticas Computacionales

Reporte de Algoritmos de ordenamiento

En este reporte se explicara brevemente cuatro de los principales algoritmos de ordenación en arreglos que se utilizan en python y además se dará un ejemplo de el código en cada uno de ellos.

1.-**Bubble**

El proceso que realiza este algoritmo es más que nada el comenzar con una lista desordena de elementos e ir comparando por pares de elementos; va preguntando si es mayor(o menor es su defecto, dependiendo del orden que le quiera dar al arreglo) que el número anterior y si lo es, intercambia las posiciones de ambos; y así continua repitiendo el proceso hasta que el arreglo quede completamente ordenado.

Es un algoritmo no muy eficiente ya que el número de operaciones que realiza es de n^2 sea tanto en el peo como el mejor de los casos.

Un ejemplo de código es:

```
cnt = 0


def burbuja(arreglo):
    aux=arreglo[:]
    global cnt
    for a in range(len(arr)):
        for b in range(0,len(arreglo)-a-1):
            if(aux[b]>aux[b+1]):
                aux[b],aux[b+1]=aux[b+1],aux[b]
                cnt+=1
    return aux
```

```
import random

m = random.sample(range(0,300),100)

print(m)

rsorted=burbuja(m)

print(cnt)

print(msorted)
```

2.- **Insertion**

En este algoritmo para iniciar a ordenar se toma un elemento como base y se va a ir comparando con cada uno de los siguientes elementos del arreglo si se encuentra uno que es menor que ese(o mayor en su defecto, dependiendo del orden en que queremos nuestro arreglo)se toma el elemento y se coloca en la primera posición y ahora ese será nuestra nueva base que comparemos de igual manera que la base anterior hasta que encuentre alguna otra base y el proceso seguirá repitiéndose hasta que se termine de ordenar el arreglo.

Este algoritmo para ordenar el arreglo realiza un numero de n^2 de operaciones aproximadamente en el peor de los casos que es cuando este desordenado el arreglo por lo cual no es muy eficiente, aunque si esta ordenado totalmente su número de operaciones se reduce a n.

Un ejemplo del código seria el siguente:

```
cnt=0
def metodo_insertion(arreglo):
        global cnt
        for num in range(1,len(arreglo)):
                valor=arreglo[num]
                i=num-1
                while i>=0:
                        cnt+=1
                        if valor<arreglo[i]:
                                arreglo[i+1]=arreglo[i]
                                arreglo[i]=valor
                                i-=1
                        else:
                                break
        return arreglo


import random
m = random.sample(range(0,300),100)
```

```
print(m)
msorted=metodo_insertion(m)
print(cnt)
print(msorted)
```

3.-**Selection**

En este tipo de algoritmo como su nombre lo dice primero se selecciona el elemento mas chico del arreglo(o el más grande, según si el orden es de mayor a menor o viceversa) y lo intercambia con el que hay en la primera posición, después busca el segundo más chico (o en su defecto más grande) y lo intercambia con el de la segunda posición; de esta manera se repite el proceso hasta que se halla ordenado por completo el arreglo.

El número de operaciones realizado por este algoritmo en la ordenación de arreglos de es de n^2 sea el peor o el mejor de los casos por lo que podemos deducir que no es muy eficiente o por lo menso no es el mejor.

El siguiente es un ejemplo de código común en python :

```
cnt=0

def seleccion(arreglo):
        aux=arreglo[:]
        global cnt
        for a in range(0,len(arreglo)-1):
                valor=a
                for b in range(a+1,len(arreglo)):
                        cnt=cnt+1
                        if arreglo[b]<arreglo[valor]:
                                valor=b
                        if valor!=a:
                                aux=arreglo[a]
                                arreglo[a]=arreglo[valor]
                                arreglo[valor]=aux
        return cnt

import random
m = random.sample(range(0,300),100)
print(m)
msorted=seleccion(m)
print(cnt)
print(msorted)
```

4.- **Quicksort**

En este algoritmo lo que se hace primero es que se selecciona un numero de lista el cual llevara el nombre de pivote es que dividirá la lista en pequeñas sublistas tal que los menores queden del lado izquierdo y los mayores del lado derecho (o viceversa dependiendo del orden en el que se quiere dejar la lista).
En el mejor de los casos, el pivote es el número del centro que solo la dividirá en dos listas y en este caso el número de operaciones realizadas de nlog(n).

Y en el peor de los casos seria que el pivote se encontrara en uno de los extremos realizando así un número de operaciones de n^2 : aun que aun así sigue siendo uno de los algoritmos más eficaces.

Un ejemplo de un código de este algoritmo es el siguiente :

```
cnt = 0

def quicksort(arr):
    global cnt
    if arr==[]:
        return []
    m=arr[0]
    left=[]
    right=[]
    for k in arr[1:]:
        if k<m:
            left.append(k)
        else:
            right.append(k)
        cnt+=1
    return quicksort(left)+[m]+quicksort(right)

import random
a = random.sample(range(0,1000),200)
print(a)
asorted=quicksort(a)
print(cnt)
print(asorted)
```