REPORTE: ALGORITMOS DE ORDENAMIENTO

Luis Daniel Honorato Hernández

Licenciatura en Matemáticas

1729448

Introducción

En este presente reporte hablare acerca de los cuatro principales algoritmos de ordenamiento que existe en la programación de diversos lenguajes de programación pero centrándose en Python los cuales sirven como su nombre lo indican acomodar un conjunto de elementos en un arreglo determinado ya sea mediante lectura de datos o que ya este dado por default y además se abordara acerca de la principal función que hace cada uno con sus respectivos pseudocodigo y un ejemplo práctico ya aplicado con todo lo visto en cada algoritmo después de esto se realizara una tabla y grafica comparativa para saber quién hace más o menos tiempo y para finalizar una pequeña conclusión que englobe todo lo aprendido de este reporte

ALGORITMOS DE ORDENAMIENTO:

Algoritmo Inserción

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

La idea de este algoritmo de ordenación consiste en ir insertando un elemento de la lista ó un arreglo en la parte ordenada de la misma, asumiendo que el primer elemento es la parte ordenada, 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.

PSEUDOCODIGO

```
def ordenar_por_insercion(L):
    for i in range (1, len(L)):
        K = L[i]
        j = i-1
        while j >= 0 and L[j] > K:
            L[j+1]=L[j]
            j=j-1
        L[j+1]=K
    return(L)
```

CODIGO

```
import random

cnt=0

def ins (array): #array es la cantidad del número de elementos

    for índice in range (1, len(array)): #se empieza con el elemento 1 ya que no tiene caso iniciarlo en 0 porque no se puede comparar con un elemento antes de este

        valor=array[índice]#es el valor ocupado en el índice 2 a comparar con el valor de la izquierda

        i=indice-1 # es el índice que se encuentra antes que el anterior para realizar la comparación

        while (i>=0): # se debe cumplir esta condición para realizar lo que está dentro del programa

            cnt=cnt+1

            if (valor<array[i]): # se compara que el segundo valor sea menor al primer valor si no es así se sale de la condición

                array[i+1]=array[i] # el valor que se encuentra en la segunda posición será intercambiado por el valor de la primera posición

                array[i]=valor #el valor de la primera posición ocupara el lugar del valor de la segunda posición

                i=i-1 #se disminuye un índice y se vuelve a revisar la condición anterior si no se cumple se sale del ciclo

            else:

                break

    return cnt


#PROGRAMA INSERCION

array= {2,1,3,5}

print(array)

print(ordenamiento_por_insercion)

print(cnt)
```

Algoritmo Quicksort

Desde que existe la ciencia de la computación, uno de los mayores problemas con los que los ingenieros se encontraban en su día a día, era el de ordenar listas de elementos. Por su causa, diversos algoritmos de ordenación fueron desarrollados a lo largo de los años y siempre existió un intenso debate entre los desarrolladores sobre cuál de todos los algoritmos de ordenación era el más rápido.

El debate finalizó abruptamente en 1960 cuando Sir Charles Antony Richard Hoare, nativo de Sri Lanka y ganador del premio Turing en 1980, desarrolló el algoritmo de ordenación Quicksort casi por casualidad mientras ideaba la forma de facilitar la búsqueda de palabras en el diccionario.

El algoritmo Quicksort se basa en la técnica de "divide y vencerás" por la que, en cada recursión, el problema se divide en subproblemas de menor tamaño y se resuelven por separado (aplicando la misma técnica) para ser unidos de nuevo una vez resueltos.

En la práctica, es el algoritmo de ordenación más rápido conocido, su tiempo de ejecución promedio es O (n log (n)), siendo en el peor de los casos O(n2), caso altamente improbable. El hecho de que sea más rápido que otros algoritmos de ordenación con tiempo promedio de O (n log (n)) (como SmoothSort o HeapSort) viene dado por que Quicksort realiza menos operaciones ya que el método utilizado es el de partición

Los pasos que realiza este algoritmo son:

Selecciona un valor del arreglo como pivote es decir un numero por el cual todos los elementos van a ser comparados.

Se realizan dos búsquedas: una de izquierda a derecha, buscando un elemento mayor que el pivote, y otra de derecha a izquierda, buscando un elemento menor que el pivote.

Cuando se han encontrado los dos, se intercambian, y se sigue realizando la búsqueda hasta que las dos búsquedas se encuentran.

Luego se organizan los subarreglos que quedaron a mano derecha y izquierda.

Ejemplo:

Tenemos un arreglo que está definido con los valores {22,40,4,10,12,35} los pasos en Quicksort para arreglarlo son los siguientes:

Se toma como pivote el numero 22 recuerden puede ser cualquier número.

La búsqueda de izquierda a derecha encuentra el valor 40 que es mayor a pivote y la búsqueda de derecha a izquierda encuentra el valor 35 no lo toma porque es mayor a el numero pivote (recuerden que la búsqueda de derecha a izquierda busca los menores) entonces continua y encuentra a 12 que es menor que el pivote y se intercambian el resultado sería {21,12,4,10,40,35}

Si seguimos la búsqueda la primera encuentra el valor 40, y la segunda el valor 10, pero ya se han cruzado, así que paramos. Para terminar la división, se coloca el pivote en su lugar el numero encontrado por la segunda búsqueda, el 10 quedando: {10,12,4,22,40,35}

Ahora tenemos dividido el arreglo en dos arreglos más pequeños que son {10,12,4} y el {40,35}, y en ellos se repetirá el mismo proceso.

PSEUDOCODIGO

```
def seleccion_pivote(L): # selecciona un pivote
    return (L [0])
#
def ordenar_por_separacion(L):
    if len(L)<=1:
        return(L)
    pivote=seleccion_pivote(L)
    L1= []; L2= []; Lp= []
    for item in L:
        if ítem<pivote:
            L1.append(item)
        alif ítem>pivote:
            L2.append(ítem)
        alif ítem==pivote:
            Lp. append(ítem)
return(ordenar_por_separacion(L1) +Lp+ordenar_por_separacion(L2))
```

CODIGO

```
import random
def quicksort(arr): #se define la función quicksort con la cantidad de elementos a trabajar la cual es nombrada arr
    global cnt
    cnt+=1
    if len(arr)<=1: #si la cantidad de elementos que se encuentra en arr es menor igual a uno se realiza lo siguiente
        return arr
```

piv=arr [0] # se selecciona el pivote a trabajar (lo más recomendable es seleccionar el que se ubica en la primera posición)

izquierda= [] # los números que ocuparan el arreglo izquierdo al momento de ser comparadas con pivote

derecha= []# los números que ocuparan el arreglo derecha al momento de ser comparadas con pivote

for i in range (1, len(arr)): #se hace un ciclo para seleccionar a los números que compararemos con pivote los cuales deben ser los que se encuentran después del hasta la última posición del arreglo

#cnt+=1

if (arr[i]<piv): # si el número ubicado en tal posición cumple con esto se acomodará en el lado izquierdo del arreglo

izquierda. append(arr[i])

else:

derecha.append(arr[i]) # si el número no cumple la condición anterior se acomodara por consiguiente en el lado derecho

return quicksort(izquierda)+[piv]+quicksort(derecha) #se acomoda el arreglo

#PROGRAMA DE QUICKSORT

arr= { 8,3,6,4,2,5,7,1}

print(quicksort)

print(arr)

print(cnt)

Algoritmo Bubble

Es el algoritmo de ordenamiento más sencillo de todos, conocido también como método del intercambio directo, el funcionamiento se basa en la revisión de cada elemento de la lista que va a ser ordenada con el elemento siguiente, intercambiando sus posiciones si están en el orden equivocado, para esto se requieren varias revisiones hasta que ya no se necesiten más intercambios, lo que indica que la lista ha sido ordenada.

El origen del nombre de este algoritmo proviene de la forma con la que suben por la lista los elementos durante los intercambios, tal y como si fueran "burbujas", el algoritmo fundamental de este método es la simple comparación de elementos siendo así el más fácil de implementar.

la siguiente, es una implementación en Pseudocódigo, donde A es un arreglo de N elementos

PSEUDOCODIGO

BURBUJA (A, N)

{I, J y AUX son variables de tipo entero}

1. Repetir con I desde 2 hasta N

    1.1 Repetir con J desde N hasta I

        1.1.1 Si A[J-1] > A[J] entonces

            Hacer AUX? A[J-1]

            A[J-1]? A[J]

            A[J]? AUX

        1.1.2 {Fin del condicional del paso 1.1.1}

    1.2 {Fin del ciclo del paso 1.1}

2. {Fin del ciclo del paso 1

CODIGO

```
import random

def burbuja (lista): #(arr) es la cantidad de elementos que contiene la lista

    cnt=0

    for recorrido in range (0, len(arr)-1): #es la cantidad de ciclos a recorrer en el arreglo

      for posición in range (0, len(arr)-1): #es la posición del arreglo menos una cantidad de elementos porque el último elemento ya está ordenado

            cnt+=1

            if (arr[posición]> arr[posicion+1]): #se hace la comparación del valor que está en la posición de la lista con el otro para ver si es mayor y si lo es se cambian ya que así es el método burbuja por ejem (18>9)

                temp=arr[posición] #la variable temporal sirve para guardar el valor mayor de la posición comparada en el paso anterior por ejem (18 ahora es variable temporal)

                arr[posición]=arr[posicion+1] #el primer elemento comparado se reemplaza por el segundo

                arr[posicion+1]=temp # el segundo elemento es la variable temporal
```

```
    return cnt


#PROGRAMA BUBBLE
lista= {4,12,8,2,1}
print (lista)
burbuja(lista)
print(contador)
```

Algoritmo Selection

El método de ordenamiento por selección consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenar todo el arreglo

Explicación:
Su funcionamiento es el siguiente:

Buscar el mínimo o máximo elemento de la lista

Intercambiarlo con el primero

Buscar el mínimo o el máximo en el resto de la lista

Intercambiarlo con el segundo

Y en general:

Buscar el mínimo o máximo elemento entre una posición i y el final de la lista

Intercambiar el mínimo o máximo con el elemento de la posición i.

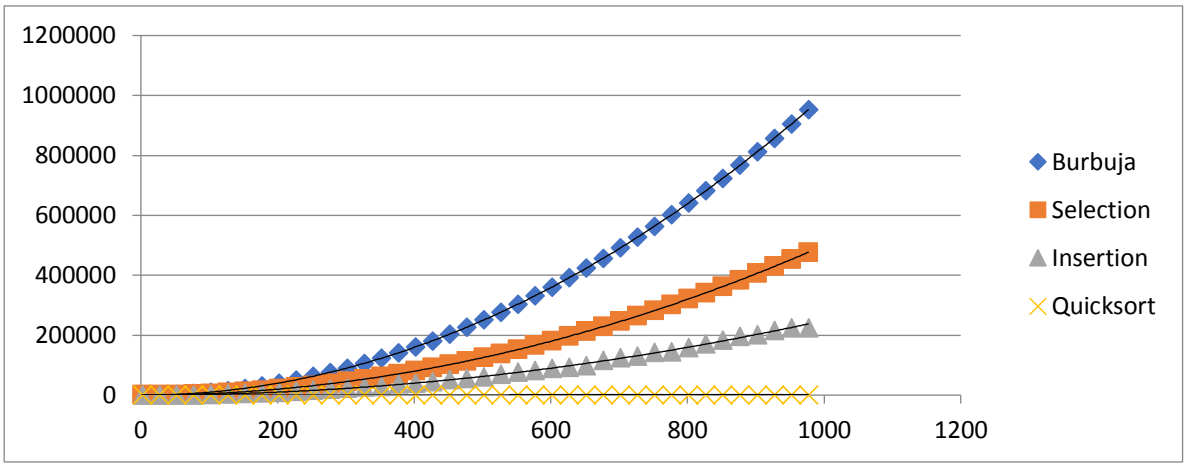
PSEUDOCODIGO

```
def ordenar_por_seleccion(L):
        for i in range(len(L)-1):
                k=i
                for j in range (i+1, len(L)):
                        if L[j]<L[K]:
                                k=j
                temp=L[i]
                L[i]=L[k]
                L[k]=temp
```

return[L]

CODIGO

```
import random

def selection(arr):

    cnt=0

    for i in range (0, len(arr)-1): #se recorre hasta len(arr)-1 porque el último elemento no es necesario ordenarlo

        mínimo =i #el primer elemento por default es el mínimo por lo pronto

        for j in range (i+1, len(arr)): #aquí se recorre con los demás elementos para preguntarles si tú eres el mínimo

            cnt+=1

        if arr[j]<arr[mínimo]: #aquí se hace la comprobación si en verdad encontramos el mínimo comparándolo con el primer elemento mínimo default

            mínimo=j #en caso de que se cumpla la desigualdad este será el nuevo mínimo

        temp=arr[i]# primero se guarda uno de los valores a una variable temporal

        arr[i]=arr[mínimo]# luego nos llevamos uno de los valores a las posiciones que hemos guardado antes

        arr[min]=temp# finalmente se hace el cambio en el otro sentido

    return cnt

#ALGORITMO DE SELECCION

arr= {6,4,2,1,3,5}

print(arr)

print(selection)

print(cnt)
```



CONCLUSIONES

De los cuatro algoritmos vistos en clase con la explicación de cada equipo y de mi profesor al principio no entendía muy bien acerca de que hacía cada uno hasta cuando investigue por mi cuenta y en la práctica ya sabía que la principal función que todos tienen en común es ordenar un conjunto de datos en un arreglo en el orden en que se ha solicitado la función de cada programa, al momento de realizarlo en la parte práctica batalle mucho para que hicieran sus respectivas funciones cada algoritmo de ordenamiento pero después ya no se me complico tanto y me di cuenta de que todos los algoritmos funcionan perfectamente y como se deben aunque hubo algunos inconvenientes en algunos como por ejemplo el burbuja para mi aunque es muy práctico siento yo pensando cuando está haciendo las operaciones que es un poco tardado a comparación de quicksort, el algoritmo de inserción es un poco más rápido que el burbuja aunque sigue la misma mecánica de comparaciones pero este lo hace en forma viceversa es decir inserción pregunta si el primer elemento es menor al siguiente que a comparación de burbuja es al revés la condición que sería si el primer elemento es mayor al que le sigue, el algoritmo de selección pienso que además que es eficiente como los otros tres este es más tardado al momento de hacer los intercambios de posiciones ya que tienen que primero asumir que el primer elemento es el mínimo de todos y después buscar de entre todos los demás elementos quienes es el mínimo y hacer la comparación con el primer elemento si en verdad es el mínimo que nosotros buscamos en el resto de los otros números que no consideramos como mínimos y si es así se hace el intercambio y se sigue el mismo proceso y es un poco tedioso y tardado aunque no lo parezca al momento de que aparezca impreso en pantalla y por último se encuentra el algoritmo quicksort que es para mí el más tardado de todos los cuatro algoritmos y el más peligroso de todos ya que al momento de escoger tu pivote y realizar comparaciones entre los números si es menor al pivote o si es mayor al pivote y separarlos en distintos grupos y luego de ahí en subgrupos hasta que te queden dos elementos puede que el arreglo al final que esta todo ordenado el pivote termina al final del arreglo y tengas que hacer de nuevo el código por eso para mí es más preferente escoger como pivote el elemento que se encuentre en la primera posición del arreglo ya que son las pocas probabilidades en que ocurra este inconveniente. Pero al final cuando vi las gráficas me di cuenta que la menos tardada es quicksort y la más tardada es burbuja