Reporte

Bubble

ORDENAMIENTO DE BURBUJA (Bubble Sort) sencillo algoritmo de ordenamiento ya que funciona comparando cada elemento de la lista que será ordenada intercambiado de posición si están en el orden incorrecto. Se revisa varias veces toda la lista para que no se necesiten más intercambios, lo que significa que la lista está ordenada correctamente. El algoritmo tiene su nombre por la forma en la que suben por la lista los elementos en los intercambios, como si fueran pequeñas burbujas.

Ya que solo usa comparaciones para trabajar con los elementos, se le dice que es un algoritmo de comparación, siendo el más fácil de entender e interpretar.

Aun que bubble es uno de los algoritmos más sencillos de implementar, su orden O(n^2) lo convierte en un algoritmo muy ineficiente para usarlo en listas que tengan un número muy chico de elementos e incluso entre los algoritmos de ordenamiento de orden O(n^2) y otros algoritmos como el Ordenamiento por inserción son considerados más eficientes. Debido a lo simple que es, el ordenamiento de burbuja es utilizado para entender el concepto de algoritmo, o de algoritmo de ordenamiento para los estudiantes que van iniciando.

Def burbuja(A):

for i in range(1,len(A)-1):

for j in range(0, len(A)-i):

if(A[j+1) <A[j]:

aux=A[j]

A[j]=A[j+1]

A[j+1]=aux

Print(A)

Insertion

El ordenamiento por inserción requiere O(n²) operaciones para ordenar una lista de n elementos.

Principalmente se tiene un solo elemento, que sabemos que es un conjunto ordenado. Luego, cuando tienes h elementos ordenados de menor a mayor, se toma el elemento h+1 y se compara con todos los elementos que ya están ordenados, parando cuando encuentra un elemento más chico ya que todos los elementos mayores han sido desplazados una posición a la derecha o puede que ya no se encuentran elementos y todos los elementos fueron desplazados y es el más pequeño. Aquí se coloca el elemento h+1 teniendo que desplazar a los demás elementos.

Pseudocodigo

Def ord-ins(lista):

For índice in range(1,len(lista)):

Valor=lista[índice]

i=índice-1

while(i>0)

if(valor<lista[i]):

lista[i+1]=lista[i]

lista[i+1]=valor

i=i-1

else

break

Quicksort

El algoritmo fundamental es el siguiente:

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

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.

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.

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. Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño. En este caso, el orden de complejidad del algoritmo es O(n·log n).

En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de 0(n²). El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas. En el caso promedio, el orden es O(n·log n).

Funcion qsort(primero, ultimo:entero)

i = primero

j = ultimo

central = A[(primero,ultimo) div 2]

repetir

mientras A[i]central

j = j - 1

fin mientras

si i < = j

aux = A[i]

A[j] = A[i]

A[i] = aux

i = i + 1

j = j - 1

fin si

hasta que i > j

si primero < j

partir(primero,j)

fin si

si i < ultimo

partir (i, ultimo)

fin si

fin funcion qsort

Selection:

Este algoritmo mejora ligeramente el algoritmo de la burbuja. En el caso de tener que ordenar un vector de enteros, esta mejora no es muy sustancial, pero cuando hay que ordenar un vector de estructuras más complejas, la operación intercambiar () sería más costosa en este caso. Este algoritmo realiza muchas menos operaciones intercambiar () que el de la burbuja, por lo que lo mejora en algo. Si la línea comentada con (!) se sustituyera por intercambiar(lista[i], lista[j]) tendríamos una versión del algoritmo de la burbuja (naturalmente eliminando el orden intercambiar del final).

Otra desventaja de este algoritmo respecto a otros como el de burbuja o de inserción directa es que no mejora su rendimiento cuando los datos ya están ordenados o parcialmente ordenados. Así como, por ejemplo, en el caso de la ordenación de burbuja se requeriría una única pasada para detectar que el vector ya está ordenado y finalizar, en la ordenación por selección se realizarían el mismo número de pasadas independientemente de si los datos están ordenados o no.

Mis códigos

def bubble(R):

global cnt1

for u in range(1,len(R)-1):

for v in range (0,len(R)-u):

cnt1+=1

if (R[v+1]<R[v]):

repuesto=R[v]

R[v]=R[v+1]

R[v+1]=repuesto

def selection(arr):

cnt3=0

for i in range(0,len(arr)-1):

val=i

for j in range(i+1,len(arr)):

cnt3+=1

if (arr[j]<arr[val]):

val=j

if(val!=i):

aux=arr[i]

arr[i]=arr[val]

arr[val]=aux

return arr

inserccion

def ord\_inser(arr):

global cnt2

for ind in range(1,len(arr)):

valor=arr[ind]

i=ind-1

while (i>=0):

cnt2+=1

if (valor < arr[i]):

arr[i+1]=arr[i]

arr[i]=valor

i=i-1

else:

break

Quicksort:

def quicksort(arr):

global cnt4

if len(arr)<=1:

return arr

p= arr.pop(0)

menores,mayores=[],[]

for e in arr:

cnt4+=1

if e <=p:

menores.append(e)

else:

mayores.append(e)

return quicksort(menores) + [p] + quicksort(mayores)

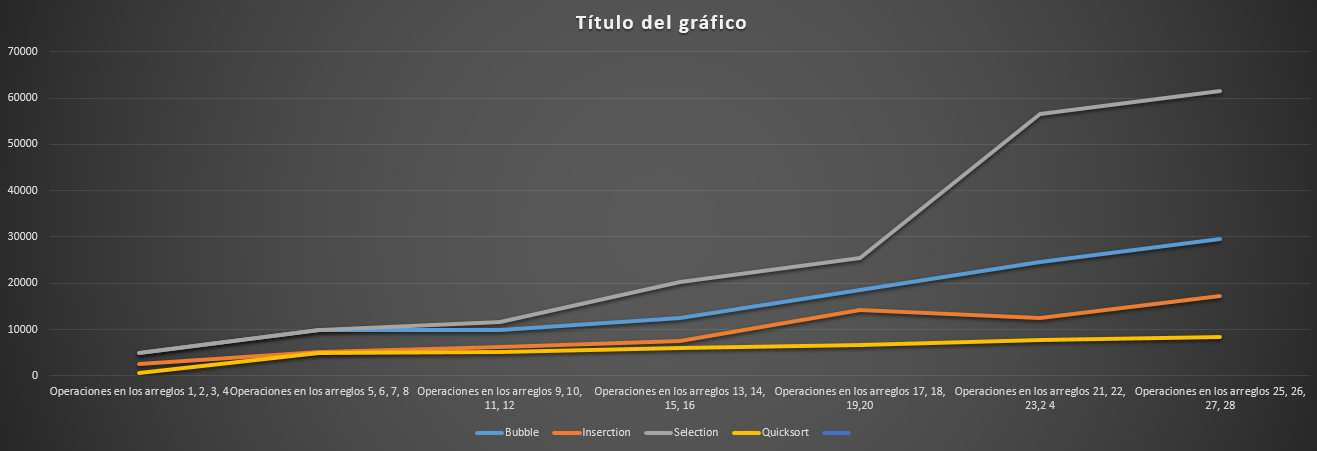
Mis conclusiones:

En lo que he visto quicksort es la mejor opción en ordenamiento ya que logra hacer menos iteraciones que los demás a diferencia de bubble que es más ineficiente si de operaciones hablamos ya que hace una mayor cantidad de operaciones que otros.

El inconveniente es cuando los arreglos solo están ordenados de forma invertida (mayor a menos)

O los arreglos de tamaños muy grandes o por el contrario muy chicos es cuando el rendimiento entre los algoritmos puede cambiar

En esta tabla esta la comparacion de los algoritmos en cuanto operaciones en 28 diferentes arreglos con diferente cantidad de elementos,salteados entre ellos y desordenados por medio de random.shuffle(arr)



En este caso se logra apreciar que selección fue el perdedor y quicksort nuestro ganador al mantenerse debajo de todos los demás, a lo mucho inserccion se mantuvo por un periodo casi a la par pero este finalmente fue creciendo en sus iteraciones.