Algoritmos de Ordenamiento

Yadira Yaneth Marroquín de la Peña

Matricula: 1701504

yaz_marroquin@outlook.com

https://github.com/yadira02/1701504MatComp

1 de septiembre de 2017

En el siguiente documento observaremos los beneficios y las características de cada uno de los algoritmos de ordenación.

1. Selection Sort

El método de ordenación 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.

```
1   cnt=0
2   def selection(arr):
3           global cnt
4           for i in range(0,len(arr)-1):
5                   val=i
6                   for j in range(i+1,len(arr)):
7                           cnt=cnt+1
8                           if arr[j]<arr[val]:
9                                   val=j
10                  if val!=i:
11                          aux=arr[i]
12                          arr[i]=arr[val]
13                          arr[val]=aux
14          return arr
15  #programa principaL
16  print("ARREGLO DESORDENADO:")
17  A=[3,7,9,4,0,5,1]
```

Al principio se tiene un elemento, que se considera un conjunto ordenado. Después, al haber k elementos ordenados de menor a mayor se coge el elemento k+1 y se va comparando con los elementos ya ordenados deteniéndose hasta que se encuentra un elemento mayor.

Como se puede ver en el código de la función, para buscar el máximo elemento en un segmento de lista se debe recorrer todo ese segmento, por lo que en nuestro caso debemos recorrer en el primer paso N elementos, en el segundo paso N —1 elementos, en el tercer paso N — 2 elementos, etc. Cada visita a un elemento implica una cantidad constante y pequeña de comparaciones (que no depende de N). Por lo tanto, tenemos que

$T(N) = c * (2 + 3 + . . . + N) = c * N * (N + 1)/2 = N^2$

Ventajas:

- Fácil implementación.
- No requiere memoria adicional.
- Realiza pocos intercambios.
- Rendimiento constante: poca diferencia entre el peor y el mejor caso.
- No hay almacenamiento temporal

Desventajas:

- Lento.
- Realiza numerosas comparaciones.
- Este método requiere n al cuadrado de número de pasos para ordenar n elementos.
- Además, su rendimiento es fácilmente influenciado por el orden inicial de los
- elementos antes del proceso de ordenamiento.
- El ordenamiento por selección sólo es apto para una lista de pocos elementos
- que estén en orden aleatorio.

2. Bubble Sort

El método de la burbuja es uno de lo más simples, es tan fácil comparar todos los elementos de una lista contra todos, si se cumple que uno es mayor o menor a otro, entonces los intercambia de posición. Se le llama burbuja debido a que los valores más pequeños “burbujean” gradualmente (suben) hacia la cima o parte superior del array de modo similar a como suben las burbujas en el agua, los valores mayores se hunden en la parte inferior del array. El ordenamiento es uno de los procesos más comunes y útiles en el procesamiento de datos, es la clasificación u ordenación de los mismos.

Características: funciona comprando elementos de dos en dos en un ciclo, intercambiándolo según sea el caso ascendente o descendente. Es necesario revisar varias veces toda lista hasta que no se necesiten más intercambios.

```
cnt=0
def burbuja(A):
        global cnt
        for i in range(1,len(A)):
                        for j in range(0, len(A)-1):
                                cnt+=1
                                if(A[j+1]<A[j]):
                                        aux=A[j]
                                        A[j]=A[j+1]
                                        A[j+1]=aux
                                        #print(A)
        return A

#programa principaL
print("ARREGLO DESORDENADO:")
A=[6,5,3,1,8,7,2,4]
print(A)
print("ARREGLO ORDENADO: \n", burbuja(A))
```

Ventajas:

- Es bastante sencillo
- Es un código reducido.
- Realiza el ordenamiento.
- Eficaz

Desventajas:

- Consume bastante tiempo de computadora.
- Requiere muchas lecturas, escrituras en memoria.
- Su desventaja principal, es uno de los menos eficientes y por ello, normalmente,
- se aprende su técnica, pero no se utiliza.

3. Insertion Sort

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 o 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.

Por ejemplo, otra forma de pensar acerca del ordenamiento. Imagina que estás jugando un juego de cartas. Tienes las cartas en tu mano y las cartas están ordenadas. Tomas exactamente una nueva carta del mazo. La tienes que colocar en el sitio correcto de manera que las cartas en tu mano sigan estando ordenadas.

```
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
#programa principaL
print("ARREGLO DESORDENADO:")
A=[3,6,8,2,0,7,9,4]
print(A)
print("ARREGLO ORDENADO: \n", insercion(A))
input("presione enter para continuar")
```

Ventajas:

- Fácil de implementar
- Requiere mínimo de memoria
- Rendimiento constante: poca diferencia entre el peor y el mejor caso.

Desventajas:

- Lento
- Realiza numerosas comparaciones

Características:

Requerimientos de Memoria: Al igual que el ordenamiento burbuja, este algoritmo sólo necesita una variable adicional para realizar los intercambios. Tiempo de Ejecución: El ciclo externo se ejecuta n veces para una lista de n elementos. Cada búsqueda requiere comparar todos los elementos no clasificados.

4. Quick Sort

El ordenamiento por partición (Quick Sort) se puede definir en una forma más conveniente como un procedimiento recursivo. Tiene aparentemente la propiedad de trabajar mejor para elementos de entrada desordenados completamente. Esta situación es precisamente la opuesta al ordenamiento de burbuja.

Este tipo de algoritmos se basa en la técnica "divide y vencerás", o sea es más rápido y fácil ordenar dos arreglos o listas de datos pequeños, que un arreglo o lista grande. El orden de complejidad del algoritmo es entonces de $0(n^2)$. 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).

```python
import random
cnt = 0

def quicksort(arr):
        global cnt
        if len(arr) < 2:
                return arr
        p = arr.pop(0)
        menores, mayores = [],[]
        for e in arr:
                cnt +=1
                if e <= p:
                        menores.append(e)
                else:
                        mayores.append(e)
        return quicksort(menores) + [p] + quicksort(mayores)

def rndar(long):
        arr = []
        for i in range(long):
                arr.append(random.randint(0,long))
        return arr


l = 10



while l <= 10:
        for replica in range(10):
                ori = rndar(l)
                arr = quicksort(ori)
                print( l, cnt, arr, ori)
                cnt = 0
        l*=2
```

Características:

- 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 sablistas, una formada por los
- elementos a la izquierda del pivote, y otra por los elementos a su
- derecha.

5. Grafica

A continuación, observaremos en cada algoritmo como es su rendimiento y cuanto tiempo lo hace, la grafica se mostrara con ayuda de Microsoft Excel.

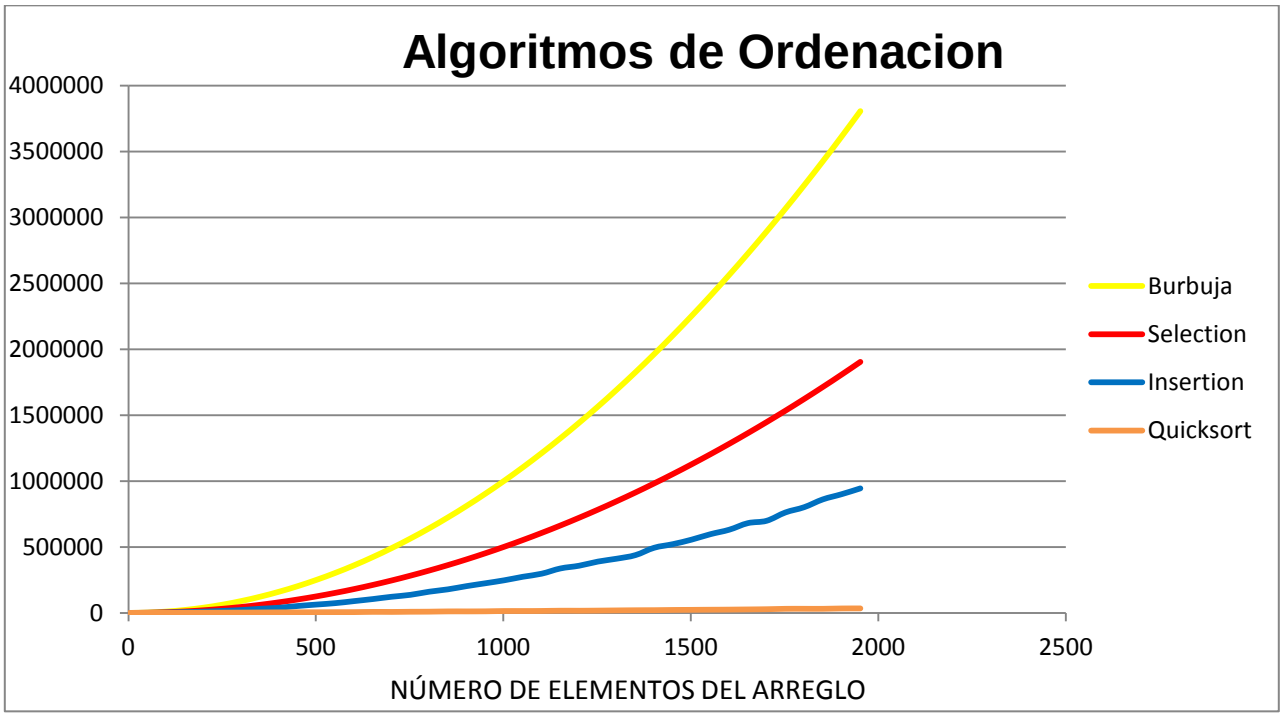
Como podemos ver la tabla muestra las 4 pruebas que realizamos para ver el funcionamiento de cada algoritmo es decir los ordena más rápido y funciona mucho más mejor que los demás.

Tabla de los 4 algoritmos de Ordenación

Longitud	Burbuja	Selection	Insertion	Quicksort
2	1	1	1	1
52	2601	1326	702	314
102	10201	5151	2511	612
152	22801	11476	6016	1106
202	40401	20301	9544	1615
252	63001	31626	15432	2104
302	90601	45451	21954	2581
352	123201	61776	31479	3281
402	160801	80601	39367	4067
452	203401	101926	52435	4489
502	251001	125751	63374	4935
552	303601	152076	74024	5726
602	361201	180901	89485	5912
652	423801	212226	105126	7728
702	491401	246051	122452	7436
752	564001	282376	137063	8662
802	641601	321201	160746	9357
852	724201	362526	178677	11433
902	811801	406351	203124	11414
952	904401	452676	224883	12059
1002	1002001	501501	247217	14188
1052	1104601	552826	274081	14176
1102	1212201	606651	297484	14787
1152	1324801	662976	337123	16722

Ahora podemos ver ya con estos valores mostraremos la grafica en cómo se comporta cada algoritmo de ordenación.

Grafica de Ordenación de los 4 algoritmos



En la grafica muestra que burbuja es algoritmo de eficaz porque es el que ordena de mayor facilidad los valores obtenidos depende de cada arreglo que le podamos dar y así programa puede tomar diferentes valores y el Quicksort es el más eficiente porque se tarda mucho en ordenar además de ser muy difícil el algoritmo para programarlo.

6. Conclusión

Observamos que cada uno de los algoritmos de ordenación es más útil, sencillo y rápido. El algoritmo que es el más fácil de usar es el de burbuja además de ser muy rápido. Porque su código es muy pequeño y puedes visualizar su procedimiento. Este algoritmo de ordenación nos sirve mucho porque puedes implementar ejercicios en búsqueda de ordenar una gran cantidad de números y los acomoda de manera individual. Una desventaja que tiene mucha capacidad para tener grandes arreglos.

En el caso de **selection** funciona tomando una lista cualquiera de los elementos para irlos ordenando de manera correcta, para así conseguir que cada uno de los algoritmos lo forme más rápida que haciéndolo a mano.

En el algoritmo de **burbuja** podemos ver como ordena los algoritmos demasiado rápidos, la ventaja que cuenta no necesita de almacenamiento temporal y tiene arreglos enormes.

Quicksort funciona un poco diferente que los otro 3 algoritmos porque primero divide las lista y tomo un elemento que es llamado pivote donde compara y acomoda. Además de ser un algoritmo un poco complicado de realizar y comprender por su forma de acomodar cada elemento. No requiere de almacenamiento adicional.

Por ultimo **insertion** analiza rápidamente la lista de los elementos cada que se inserta el elemento de manera desordenada los acomoda con una mayor facilidad. Su desventaja que no funciona muy bien como los demás y ofrece espacio mínimo.