
**Universidad Autónoma**

**de Nuevo León**


Facultad de Ciencias Físico Matemáticas

# Reporte de investigación:

# Algoritmos de ordenamiento

Daniela Carrizales

Profesor: José Anastacio

Introducción.

Este reporte de investigación consiste en los algoritmos de ordenación hecho en el programa Python, tiene como objetivo conocer lo básico de los cuatros algoritmos de ordenación que a continuación se presentará:

1. Método de Burbuja

2. Método de Selección

3. Método de Insercción

4. Método de Ordenación Rápida

Al terminar de exponer cada uno de manera breve, se pretende que se tenga un concepto, identificar con cuál se gustaría trabajar y con el pseudocódigo pensar en el código adecuado para el programa que se desea realizar.

*Algoritmos de ordenamiento.*

*Burbuja (Bubble Sort):* es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas burbujas. También es conocido como el método del intercambio directo. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo el más sencillo de implementar.

El procedimiento de la burbuja es el siguiente:

Ir comparando desde el elemento 0, de número a número hasta hallar el mayor, si este efectivamente es el mayor del arreglo se va hasta el último elemento, en caso contrario, sólo se cambiará por uno mayor.

El procedimiento anterior se repite hasta ordenar todos los elementos del arreglo.

Desventaja:

La desventaja de este algoritmo de ordenación, es que, cuando ya tienes una parte ordenada del arreglo, vuelve a partir desde a principio comparando de nuevo cada elemento.

```
def ordenamientoBurbuja(lista):
    for numPasada in range(len(lista-1)):
        for i in range(num):
            if lista[i]>lista[i+1]:
                temp = lista[i]
                lista[i] = lista[i+1]
                lista[i+1] = temp

lista = [54,26,93,17,77,31,44,55,20]
ordenamientoBurbuja(lista)
print(lista)
```

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.

Este algoritmo de ordenación consiste en ir insertando un elemento de la 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.

```
#Declaracion de Funciones
def insercion(A):
    for i in range(len(A)):
        for j in range(i,0,-1):
            if(A[j-1] > A[j]):
                aux=A[j];
                A[j]=A[j-1];
                A[j-1]=aux;
    print A;

#Programa Principal
A=[6,5,3,1,8,7,2,4];
print A;
insercion(A);
```

Selección (Selection sort).

Este algoritmo de ordenamiento se basa en hallar el menor de todos los elementos del arreglo y cambiarlo con el primer elemento, en seguida el segundo mas chico, hasta acomodar a cada uno.

```
def selectionsort(lista,tam):
    for i in range(0,tam-1):
        min=i
        for j in range(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_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()
selectionsort(A,len(A))
imprimeLista(A,len(A))
```

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.

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.

```
def quicksort(L, first, last):
    # definimos los índices y calculamos el pivote
    i = first
    j = last
    pivote = (L[i] + L[j]) / 2

    # iteramos hasta que i no sea menor que j
    while i < j:
        # iteramos mientras que el valor de L[i] sea menor que
```

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 $O(n \cdot \log n)$.

En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de $0(n^2)$. El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas.

```
    # iteramos mientras que el valor de L[j] sea mayor que pivote
    while L[j] > pivote:
        # decrementamos el índice
        j-=1
    # si i es menor o igual que j significa que los índices se han cruzado
    if i < = j:
        # creamos una variable temporal para guardar el valor de L[j]
        x = L[j]
        # intercambiamos los valores de L[j] y L[i]
        L[j] = L[i]
        L[i] = x
        # incrementamos y decrementamos i y j respectivamente
        i+=1
        j-=1

# si first es menor que j mantenemos la recursividad
if first < j:
    L = quicksort(L, first, j)
# si last es mayor que i mantenemos la recursividad
if last > i:
    L = quicksort(L, i, last)

# devolvemos la lista ordenada
return L
```

En el caso promedio, el orden es O(n·log n).

Conclusión.

Con las diferentes sesiones que tuvimos en clase de matemáticas computacional,se tuvo esta información de parte de los compañeros, los cuales varios concordamos que aunque el método de ordenación rápida es el más efectivo y rápido se nos complican un poco al querer hacer el código, sin embargo, nos gustaría practicar los cuatros algoritmos de ordenación para cuando ocupemos hacer el código podamos probar con cuál se nos facilitaría realizarlo.

Sin duda los algoritmos es una manera más fácil de aprender para programar, ya que, ya hay bases estructuradas sin olvidar que fundamentar más los conocimientos obtenidos en estas clases.

Bibliografía.

https://www.ecured.cu/Ordenamiento_de_burbuja

https://cdklhph.wordpress.com/2015/08/10/ordenamiento-insercion/

https://beastieux.com/2011/01/24/codigo-python-ordenamiento-por-seleccion/

http://mis-algoritmos.com/ordenamiento-rapido-quicksort