En este reporte se explicará sobre algunos de los distintos algoritmos de ordenamiento que existen.

Algoritmos de ordenamiento:

1. Bubbles:
En este algoritmo es de los más sencillos de programar, pero es demasiado largo. Su nombre se debe a que los elementos al ser intercambiados suben como “burbujas” para ordenarse. En el peor de l os casos se tendrán que hacer n^2 intercambios, siendo n el número de elementos.
El desempeño de bubbles es bueno en el ordenamiento de pequeñas bases de datos, en cambio mientras crece el tamaño de datos a ordenar más tardara el algoritmo en ordenarlos.

```
valor=0

def bubble(arr):
        aux=arr[:]
        global valor
        for i in range(len(arr)):
                for j in range(0,len(arr)-i-1):
                        if(aux[j]>aux[j+1]):
                                aux[j],aux[j+1]=aux[j+1],aux[j]
                                valor+=1
        return aux

import random
p= random.sample(range(0,200),100)
print(p)
psorted=bubble(p)
print(valor)
print(psorted)
```

2. Inserction:
Este algoritmo, en el peor de los casos realiza n^2 procesos. Lo que hace es tomar el primer elemento del conjunto y compararlo con el siguiente. Esto hace que se muevan las posiciones y se “inserten” en otras posiciones ya con el orden que se establece.
El desempeño de inserction es bueno en el ordenamiento de pequeñas bases de datos, en cambio mientras crece el tamaño de datos a ordenar más tardara el algoritmo en ordenarlos.

```
cnt=0
def orden_por_inserccion(array):
        global cnt
        for indice in range (1, len(array)):
                valor=array[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:
                        cnt+=1
                        if valor<array[i]:     #comparamos valor con el elemento anterior
                                array[i+1]=array[i]
                                array[i]=valor
                                i-=1
                        else:
                                break
                return array
```

3. Selection:
En este algoritmo se busca el mínimo elemento entre una posición hasta el final de la lista e intercambia el mínimo con esa posición. Se mejora ligeramente el algoritmo “Bubbles”. En el peor de los casos se realizan n

comparaciones.
El desempeño de selection es bueno en el ordenamiento de pequeñas bases de datos, en cambio mientras crece el tamaño de datos a ordenar más tardara el algoritmo en ordenarlos.

```
def selection (arr):
        for i in range (0,len(arr)-1):
                val=i
                for j in range(i+1, len(arr)):
                        if arr[j]<arr[val]:
                                val=j
                if val !=1:
                        aux=arr[i]
                        arr[i]=arr[val]
                        arr[val]=aux
        return arr
```

4. QuickShort :
Se elige el primer elemento, al que llamaremos pivote, se crea un nuevo conjunto donde el pivote está en medio, entonces comienza a ordenar a la izquierda y derecha del pivote dependiendo del lugar que se le asigne. Se repite el proceso hasta terminar. Este algoritmo tiene una complejidad de nLog2(n).
El desempeño de este algoritmo es eficiente cuando la cantidad de datos a ordenar es muy grande, en cambio cuando la cantidad de datos es muy pequeña su eficiencia disminuye cosiderablemente.

```
import random
cnt=0

def quicksort(arr):
        global cnt
        if len(arr) <= 1:
                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)

```

Conclusiones:

Los algoritmos de ordenamiento sirven para facilitar el orden de una enorme cantidad de datos en el menor de los tiempos, pero obviamente cada uno tienes sus pros y contras que hacen que unos sean eficientes en algunos casos y deficientes en otros.
Es importante identificar la cantidad de datos que queremos ordenar para saber cuál de todos es mejor para nuestra tarea.

Función de arreglos aleatorios

Graficas