Pablo Emilio Lopez Avila Mat. 1740087 Matemáticas computacionales.

Algoritmos de Ordenamiento.

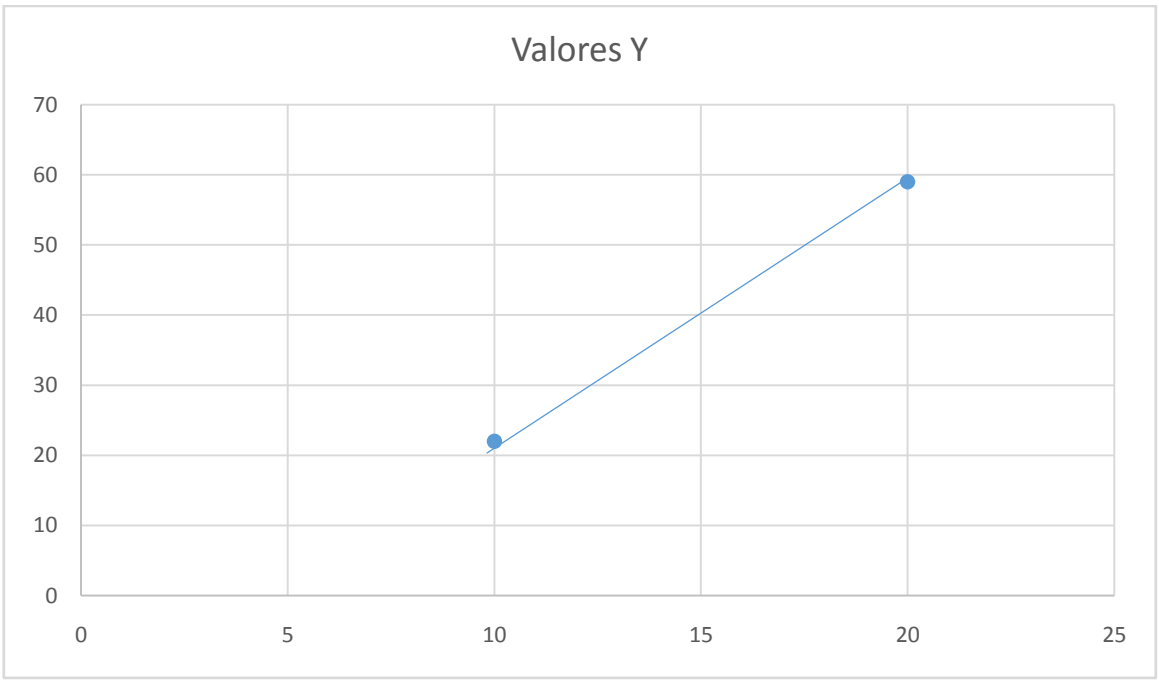
Algoritmo Quicksort:

Es un algoritmo de ordenamiento que se descubrió “por casualidad” buscando algoritmos que buscaban facilitar la búsqueda de palabras en el diccionario; Actualmente en su tipo es la mejor opción en cuanto a **complejidad** ya que en el mejor de sus casos su tiempo de ejecución es $On(ln(n))$ y en el peor On^2. Sin tanta formalidad lo que hace Quicksort es tomar un número que debería ser no tan grande en la lista ni tan pequeño, el cual servirá como punto de comparación entre 2 grupos de números, a este número se le llama “Pivote”. Los números más pequeños o iguales al pivote formaran otra lista, el elemento pivote se aparatará y la tercer lista la cual contiene números mayores o iguales al pivote. Estas nuevas listas irán formando otras pequeñas listas en las cuales se tomarán nuevos pivotes hasta que se ordenen en su totalidad los elementos.

Este es un pseudocodigo para utilizar el algoritmo desde una lista a ingresar.

```
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)
```

Su mayor ventaja es la rapidez y el hecho de no ocupar memoria adicional, pero su desventaja es que utiliza muchos recursos dentro del lenguaje y también la diferencia entre el peor y mejor de los casos.



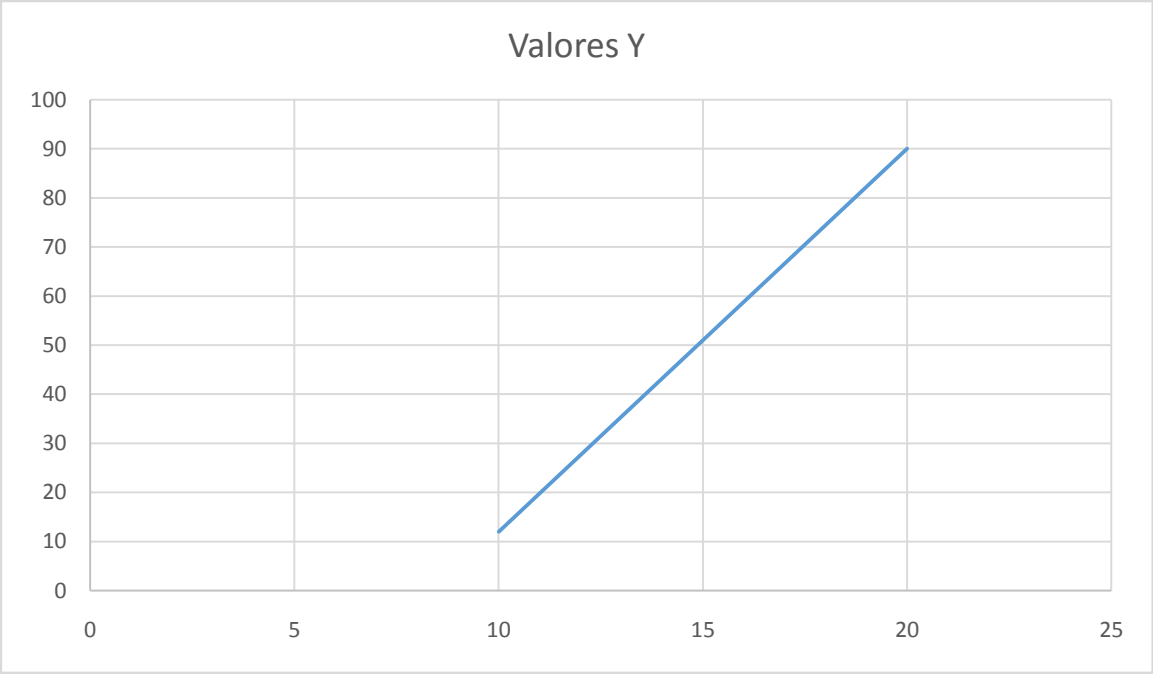
Algoritmo Selection

Es un algoritmo de ordenamiento, el cual no es muy eficiente del todo, su **complejidad** varia del número de elementos a ordenar, representado como O= $\frac{\mathbf{n^2+n}}{\mathbf{2}}$ **,** de manera simple este algoritmo se basa en leer la lista completa y tomar el valor másmínimo y ponerlo en la primer posición intercambiando el lugar con el elemento en esa posición , seguido de volver a leer ahora toda la listaa partir del segundo elemento buscando el segundo más mínimo y ponerlo en esa posición sustituyendo al elemento de la segunda posición de esa manera hasta que se ordena completamente la lista.

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

```
    return contador
```

Su Ventaja consiste en no ocupar memoria adicional, además de hacer muy pocos cambios o movimientos entre el peor o mejor casos, reflejado en que no hay mucha diferencia entre el peor o mejo caso. Pero suele ser lento y hace múltiples comparaciones.



Algoritmo Bubble

Es un algoritmo basado en un ordenamiento muy simple en el cual va revisando cada elemento del arreglo o lista, en la cual toma el primer elemento y lo va comparando de izquierda a derecha viendo si ese primer elemento es mayor a los que se va encontrando, si encuentra un elemento mayor a él mismo, deja ahí el primer término y continua la comparación con ese elemento mayor, así sigue hasta acabar con los elementos en la lista o arreglo, lo que hace necesario que se repita varias veces la misma acción. Su Complejidad va dependiendo de sus elementos representado como O= $\frac{\mathbf{n^2 - n}}{\mathbf{2}}$.Su nombre proviene de una representación de una burbuja de gas la cual crece conforme avanza, así es con el elemento que vamos comparando en el arreglo o lista.

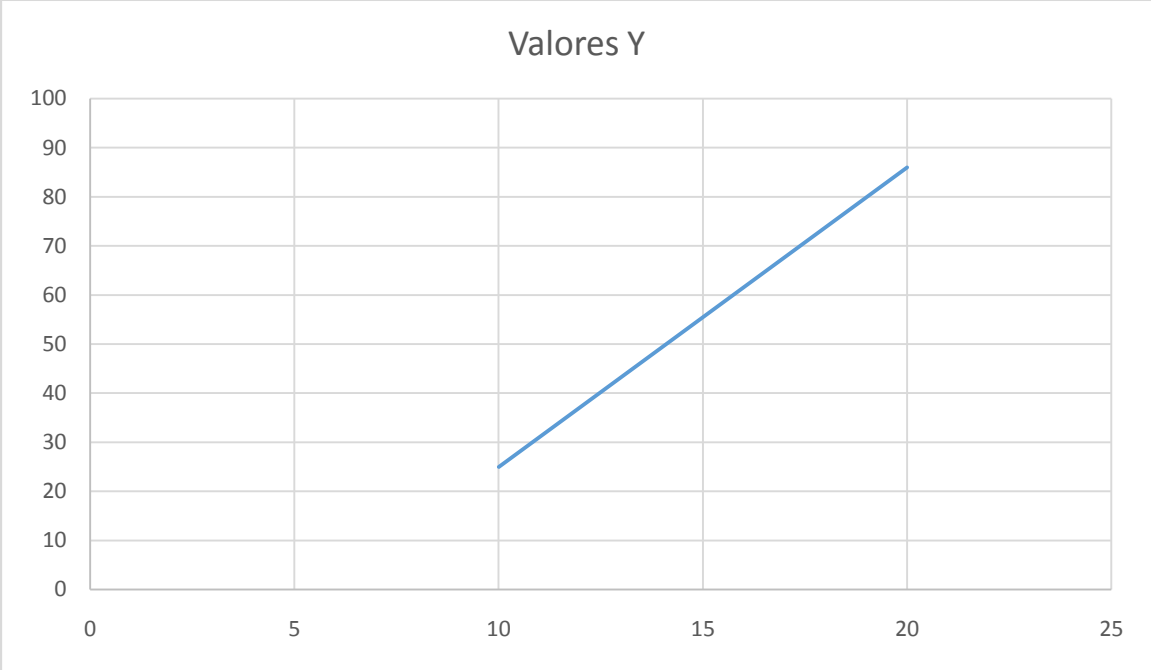
```
defburbuja(A):
    for i in range (1,len(A)):
        for j in range (0,len(A)-1):
            if (A[j+1]<A[j]):
                aux=A[j]
```

A[j]=A[j+1]

A[j+1]=aux

print(A)

La ventaja principal es que no ocupa memoria principal, pero tiene masdeventajas que ventajas, como lo es su lentitud la realización de muchas comparaciones y sobre todo sus multples intercambios de posición.



Algoritmo Insertion

Es un algoritmo muy común debido a que se parece mucho a la manera en que todos nosotros como humanos ordenamos las cosas, toma el primer elemento el cual se compara con el segundo, se hace el cambio si es necesario, ahora que tenemos dos elementos ordenados los tomamos de referencia a ambos, el tercer elemento lo comparamos con el mayor de los elementos que teníamos para saber si es mayor o menor a este si es mayor se deja dónde está, si es menor al segundo elemento pero mayor al primero se pone entre los dos, en resumidas palabras toma a esos dos números y los compara uno por uno con el elemento que le sigue al mismo, para así ir ordenando la lista p arreglo. La complejidad de este algoritmo siempre es $O=n^2$

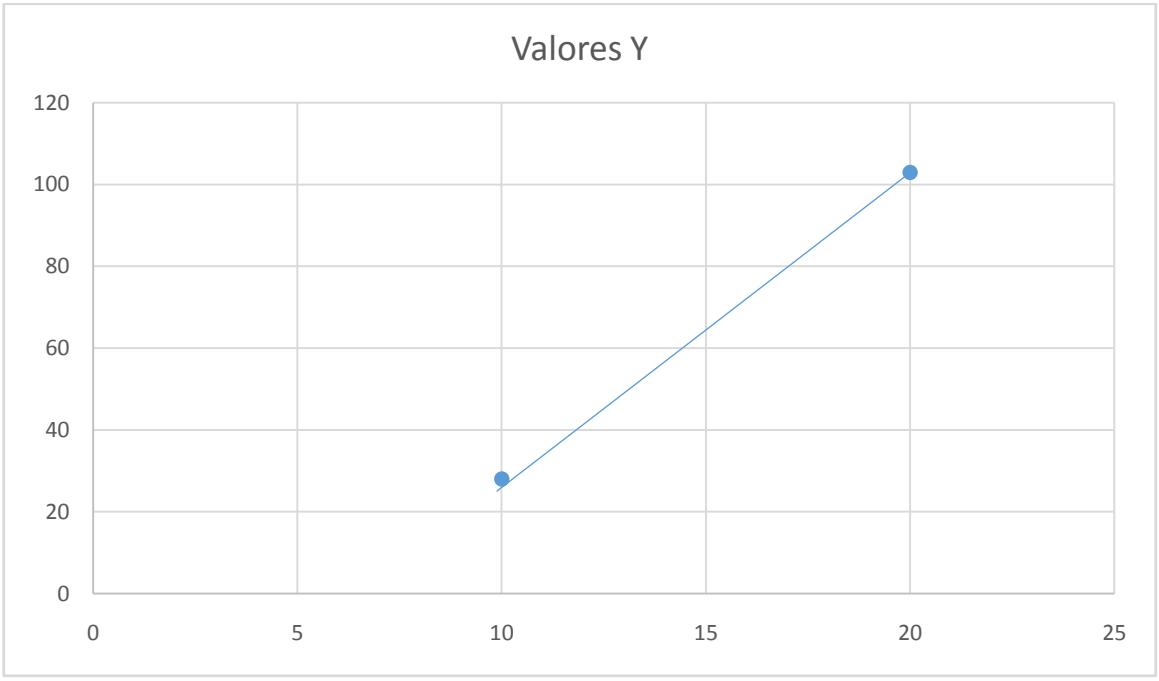
defOrdenPorInsercion(array):

global cnt

cnt=0

```
    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 comparamos
while i>=0:
cnt+=1
if valor<array[i]: #comparamos valor con el valor anterior
array[i+1]=array[i] #intercambiamos los valores
array[i]=valor
                                i-=1
else:
                                break

returnarray
```

su ventaja es su fácil implementación y sus bajos requerimentos de memoria, pero sus desventajas son una considerable lentitud y sus múltiples comparaciones dentro del algoritmo



COMPLEJIDAD

Quicksort n2 10,22- 20, 20,59

Insertion n2 10,28-, 20,103

Selection n2 10, 12-

Bubble n2 10,25 – 20-86