

REPORTE DE ALGORITMOS DE ORDENAMIENTO

MATEMÁTICAS COMPUTACIONALES

BENNY OZIEL ZAPATA VALLES

1727838

Ordenamiento Burbuja

Este algoritmo toma los dos primeros elementos del arreglo y los compara entre sí, en caso del elemento que esté más a la izquierda, es decir, el primer elemento del arreglo, sea mayor al siguiente entonces cambian de posición, en caso de que no el elemento más cercano a la izquierda sea menor, entonces pasa a comparar los siguientes otros dos elementos del arreglo, de tal manera que encuentra el número mayor y lo lleva al final.

Este tipo de algoritmo es muy ineficiente ya que siempre realiza la misma cantidad de comparaciones (dependiendo de n elementos del arreglo), su rendimiento es de $(n^2 - n)/2$

Cómo pseudocódigo de este algoritmo tenemos como ejemplo:

```
#Declaración de Funciones

def burbuja(A):
    for i in range(1, len(A)):
        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;

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

Ordenamiento por inserción

En este ordenamiento tomaremos el segundo elemento de el arreglo y lo comparamos con el primero, dependiendo de cuál sea menor, se orden en primer lugar del arreglo, después pasa a tomar el tercer elemento y lo compara con los anteriores, para así colocarlo en la posición en la que debería estar. Así hasta terminar con el arreglo entero.

El rendimiento de este ordenamiento por inserción de una lista de tamaño n puede insumir (en el peor caso) tiempo del orden de n^2 .

Un ejemplo de la utilización de método sería:

```
cnt=0

def orden_por_insercion(array):
    global cnt
    for indice in range (1,len(array)):
        valor=array[indice]
        i=indice-1
        while i>0:
            cnt+=indice-1
            if valor<array[i]:
                array[i+1]=array[i]
                array[i]=valor
                i-=indice-1
            else:
                break
        return array
array=[26,2,45,30,1,450,1]
print orden_por_insercion(array)
cnt
```

Ordenamiento por selección

La forma por la que trabaja este método es simple, busca el elemento más pequeño del arreglo y lo pone en el primer lugar, después busca el segundo y lo pone en su lugar, y así sucesivamente hasta terminar el arreglo completo.

En este caso, el número de comparaciones no depende de la posición de los términos sino, la cantidad de ellos, dando así un rendimiento de n^2 .

Como ejemplo de este método de ordenamiento, tenemos:

```
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 != i:
            aux=arr[i]
            arr[i]=arr[val]
            arr[val]=aux
        return arr
    print(arr)
import random
def ran_n(n,lim_i=0,lim_s=100):
    arr=[]
    for i in range(n):
        arr.append(random.randint(lim_i,lim_s))
    return arr

```

Quicksort

Este método elige un elemento pivote, después coloca 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. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda. 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.

Por último, repite 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.

El rendimiento de este código esta dado por n^2 en el mejor de los casos y $n \cdot \log(n)$ para el peor.

Un pseudocódigo sería:

```

contador=0
def quicksort(arr):
    global contador
    if arr==[]:
        return arr
    izq,der,m=[],[],arr[0]
    for k in arr[1:]:
        contador+=1
        if k<m:

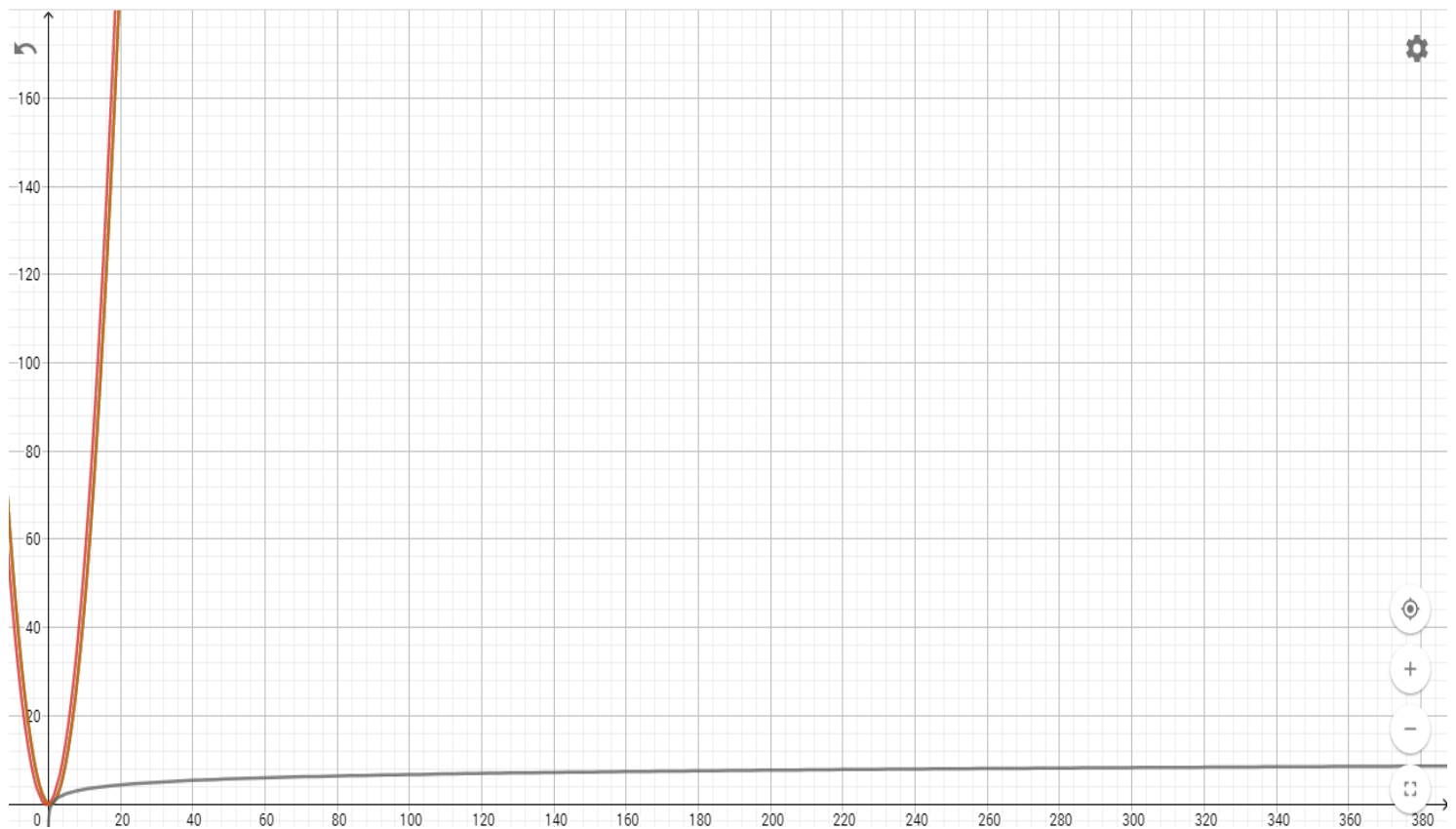
```










```

        izq.append(k)
    else:
        der.append(k)
    return quicksort(izq)+[m]+quicksort(der)
import random
arr= random.sample(range(0,1000) ,10)
print(arr)
arrs=quicksort(arr)
print(contador)
print(arrs)

```

RENDIMIENTO DE ALGORITMOS



	$b(x) = \frac{x^2 - x}{2}$	
	$s(x) = \frac{x^2 + x}{2}$	
	$l(x) = \frac{x^2 - x}{2}$	
	$Q(X) = \log_2(X)$	
	Entrada...	

Con lo ya explicado de cada algoritmo y con este gráfico podemos observar que, el ordenamiento por selección es el menos favorable para la optimización del tiempo, mientras que inserción y burbuja tardan lo mismo, aún así, para ordenar un arreglo, reduciendo al menor tiempo posible el proceso, el algoritmo de quicksort es el mejor, ya que como observamos en el gráfico, todas las ecuaciones representan el caso mas desfavorable para cada algoritmo, dando una gráfica con respecto al número de operaciones.

Es posible que si corremos estos programas en computador, no notemos la diferencia de tiempo u operaciones por la velocidad con la que son ejecutados, pero si se toman valores muy grandes o el número de elementos en el arreglo es muy amplio se notara una mayor diferencia.