LABORATORIO DE COMPUTACION II: ORDEBAMIENTO
Prof.: Hugo Araya Carrasco

Para este laboratorio considere los códigos siguientes que implementan 4 métodos de ordenamiento escritos en lenguaje Python, usted debe analizarlos y probarlos.

Bubble Sort

```
def ordenamientoBurbuja(unaLista):
    for numPasada in range(len(unaLista)-1,0,-1):
        for i in range(numPasada):
            if unaLista[i]>unaLista[i+1]:
                temp = unaLista[i]
                unaLista[i] = unaLista[i+1]
                unaLista[i+1] = temp
```

Insertion Sort

```
def ordenamientoPorInsercion(unaLista):
   for indice in range(1,len(unaLista)):
     valorActual = unaLista[indice]
     posicion = indice
     while posicion>0 and unaLista[posicion-1]>valorActual:
         unaLista[posicion]=unaLista[posicion-1]
         posicion = posicion-1
     unaLista[posicion]=valorActual
```

Shell Sort

```
def ordenamientoDeShell(unaLista):
    contadorSublistas = len(unaLista)//2
    while contadorSublistas > 0:

      for posicionInicio in range(contadorSublistas):
        brechaOrdenamientoPorInsercion(unaLista,posicionInicio,contadorSublistas)
      contadorSublistas = contadorSublistas // 2
```

Quick Sort

```
def ordenamientoRapido(unaLista):
   ordenamientoRapidoAuxiliar(unaLista,0,len(unaLista)-1)

def ordenamientoRapidoAuxiliar(unaLista,primero,ultimo):
   if primero<ultimo:
       puntoDivision = particion(unaLista,primero,ultimo)
       ordenamientoRapidoAuxiliar(unaLista,primero,puntoDivision-1)
       ordenamientoRapidoAuxiliar(unaLista,puntoDivision+1,ultimo)

def particion(unaLista,primero,ultimo):
   valorPivote = unaLista[primero]
   marcaIzq = primero+1
   marcaDer = ultimo
   hecho = False
   while not hecho:

       while marcaIzq <= marcaDer and unaLista[marcaIzq] <= valorPivote:
           marcaIzq = marcaIzq + 1
       while unaLista[marcaDer] >= valorPivote and marcaDer >= marcaIzq:
           marcaDer = marcaDer -1
       if marcaDer < marcaIzq:
           hecho = True
       else:
           temp = unaLista[marcaIzq]
           unaLista[marcaIzq] = unaLista[marcaDer]
           unaLista[marcaDer] = temp
   temp = unaLista[primero]
   unaLista[primero] = unaLista[marcaDer]
   unaLista[marcaDer] = temp
   return marcaDer
```

Para probar los algoritmos anteriores usted debe generar 3 archivos de datos con 1.000.000 de elementos (conteniendo sólo números de a lo más 6 dígitos), el primer archivo debe estar ordenado en forma ascendente, el segundo debe estar ordenado en forma descendente y el tercer archivo debe contener los datos en forma aleatoria. En cada uno de los casos de prueba usted debe utilizar los mismo elementos y escribir el tiempo empleado para ordenar dicho caso.

Los resultados deben ser presentados en forma de gráfico de acuerdo a las tablas siguientes.

S.O. W / L	Python		
Algoritmo	Bubble Sort		
n	Ordenado	Inverso	Aleatorio
1000			
10000			
100000			
200000			
300000			
400000			
500000			
600000			
700000			
800000			
900000			
1000000			

S.O. W / L	Python		
Algoritmo	Insertion Sort		
n	Ordenado	Inverso	Aleatorio
1000			
10000			
100000			
200000			
300000			
400000			
500000			
600000			
700000			
800000			
900000			
1000000			

S.O. W / L	Python		
Algoritmo	Shell Sort		
n	Ordenado	Inverso	Aleatorio
1000			
10000			
100000			
200000			
300000			
400000			
500000			
600000			
700000			
800000			
900000			
1000000			

S.O. W / L	Python		
Algoritmo	Quick Sort		
n	Ordenado	Inverso	Aleatorio
1000			
10000			
100000			
200000			
300000			
400000			
500000			
600000			
700000			
800000			
900000			
1000000			

Para medir el tiempo de ejecución del programa (en segundos) en lenguaje Python puede seguir el esquema siguiente:

```
from time import time
if __name__ == "__main__":
      unaLista = [54,26,93,17,77,31,44,55,20]
      tiempo_inicial = time()
      ordenamientoDeShell(unaLista)
      print(unaLista)
      tiempo_final = time()
      tiempo_ejecucion = tiempo_final - tiempo_inicial
      print tiempo_ejecucion
```

Para generar un conjunto de 1.000.000 de datos aleatorios cree el siguiente programa llamado genera_aleatorio.py, después de compilado debe ejecutarlo desde la línea de comando de la forma siguiente:

python genera_aleatorio > datos_aleatorios.

```
from random import randint
def genera(n):
      for i in range(n):
            print randint(0, n-1)

if __name__ == "__main__":
      genera(1000000)
```

Para generar un conjunto de 1.000.000 de datos ordenados de menor a mayor, cree el siguiente programa llamado genera_ordenado.py, después de compilado debe ejecutarlo desde la línea de comando de la forma siguiente:

python genera_ordenado > datos_ordenados.

```
def genera(n):
      for i in range(n):
            print i

if __name__ == "__main__":
      genera(1000000)    printf("%d\n",numero);
```

Para generar un conjunto de 1.000.000 de datos ordenados de mayor a menor, cree el siguiente programa llamado genera_invertido.py, después de compilado debe ejecutarlo desde la línea de comando de la forma siguiente:

python genera_invertido > datos_invertidos.

```
def genera(n):
      i = n
      while (i > 0):
            print i
            i = i - 1

if __name__ == "__main__":
      genera(1000000)
```