Estructura de Datos y Algoritmos 2

Semestre 2023-1

Profesor: Jorge Alberto Solano Galvez

Práctica 01. Algoritmos de ordenamiento. Parte 1.

Grupo 06

Integrantes:
Aguilar Martinez Erick Yair
Casillas Herrera Leonardo Didier

Práctica 1

Objetivo:
Identificar la estructura de los algoritmos de ordenamiento bubble sort y merge sort.
Actividades:

• Implementar el algoritmo bubble sort en lenguaje Python para ordenar una secuencia de datos.

• Implementar el algoritmo merge sort en lenguaje Python para ordenar una secuencia de datos.

Instrucciones:

• Implementar en lenguaje Python los algoritmos de ordenamiento bubble sort y de
merge sort que permitan ordenar números

• A partir de los algoritmos implementados en Python, obtener los polinomios del
mejor, el peor y el caso promedio de complejidad de bubble sort y de merge sort

• A partir del algoritmo implementado en Python, generar la gráfica del tiempo de
ejecución para los algoritmos de bubble sort y de merge sort para diferentes instancias de tiempo (listas de 1 a 1000 elementos) para los siguientes casos:

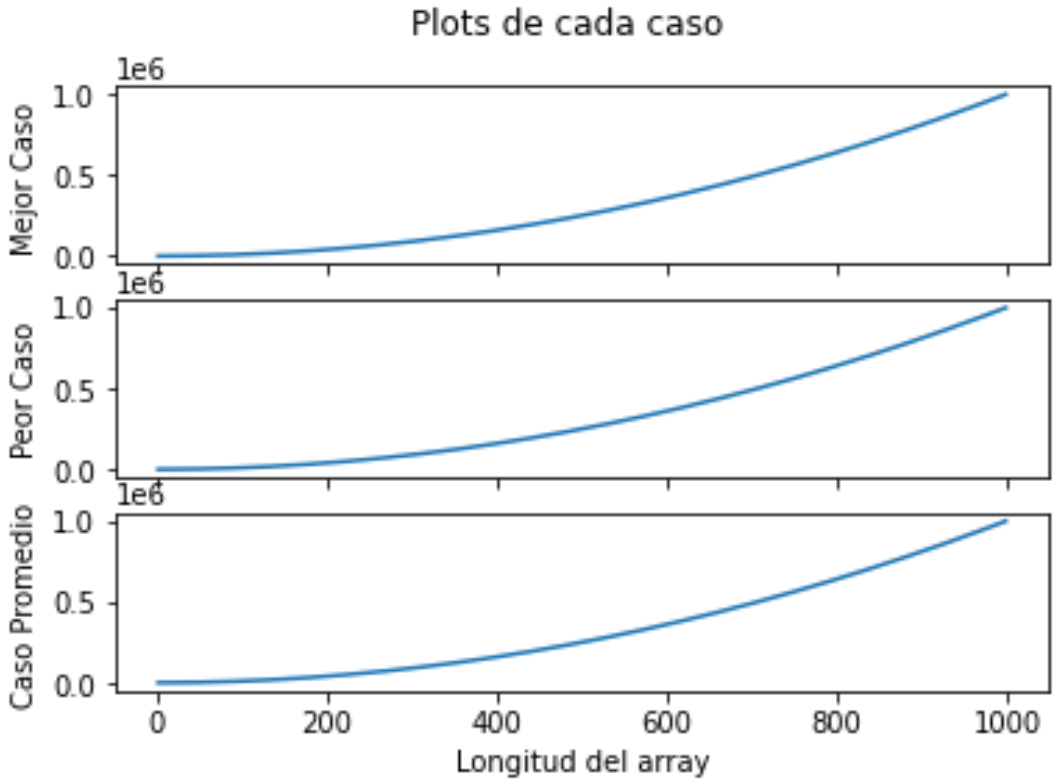
– El mejor caso: con una lista ordenada
– El peor caso: lista ordenada en forma inversa
– El caso promedio: una lista aleatoria

La práctica se puede realizar de manera individual, en parejas o en equipo de 3 integrantes.
Cada persona debe subir su práctica a Google Classroom porque las conclusiones son individuales. Durante la revisión de la práctica se le puede cuestionar a cualquier integrante
del equipo, la calificación es única para todo el equipo.

Resultados Obtenidos.
Bubble Sort.

```
def burbuja(array):
    for i in range(len(array)):
        for j in range(len(array) - 1):
            if array[j] > array[j+1]:
                tmp = array[j+1]
                array[j+1] = array[j]
                array[j] = tmp
```

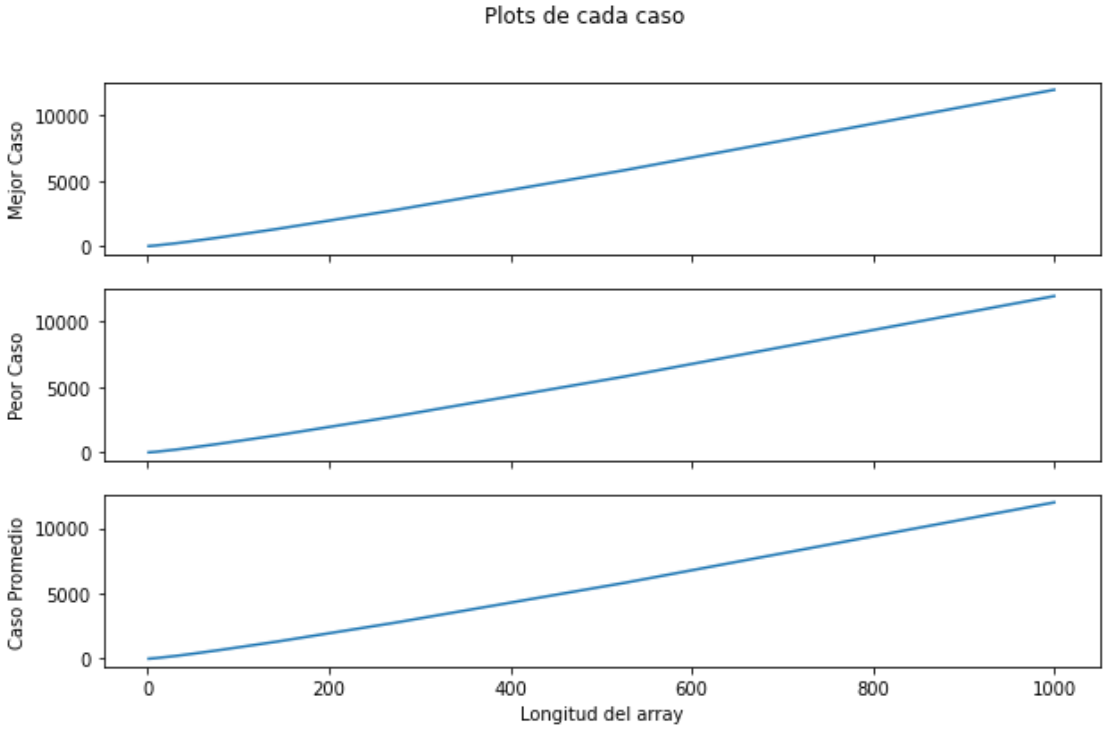
Gráficas



Merge Sort.

```python
def CrearSubArreglo(A,indIzq,indDer):
  return A[indIzq:indDer+1]
def Merge(A,p,q,r):
  Izq = CrearSubArreglo(A,p,q)
  Der = CrearSubArreglo(A,q+1,r)
  i = 0
  j = 0
  for k in range(p,r+1):
    if(j >= len(Der)) or (i < len(Izq) and Izq[i] < Der[j]):
      A[k] = Izq[i]
      i = i + 1
    else:
      A[k] = Der[j]
      j = j + 1
def MergeSort(A,p,r):
  if r - p > 0:
    q = int((r+p) / 2)
    MergeSort(A,p,q)
    MergeSort(A,q+1,r)
    Merge(A,p,q,r)
```

Gráficas.



Conclusiones.

- Aguilar Martinez Erick Yair:

Ordenar un conjunto de datos, en este caso una lista, es una de las operaciones más costosas pero que con un poco de análisis se pueden conseguir menores costos.

Ante el problema de ordenar un conjunto de datos podemos utilizar diferentes estrategias para la creación de algoritmos y en este caso se han utilizado en particular dos, fuerza bruta y divide y vencerás.

- Casillas Herrera Leonardo Didier:

Los algoritmos de ordenamiento tienen distintas complejidades y por lo tanto, distintos tiempos de ejecución, en base a eso debemos evaluar cuál es el algoritmo que más nos conviene usar en el ordenamiento de un arreglo además de revisar cuántos recursos de la computadora se utilizan.