Universidad Tecnológica Equinoccial

Tecnología Superior en Desarrollo de Software

Programación

Nombre: Jossue Guerrero

Paralelo: "B"

ALGORITMOS DE ORDENAMIENTO

.-Estos algoritmos son fundamentales en la organización de datos porque permiten estructurarlos de manera mas eficiente que ayuden a su búsqueda y manipulación

Características:

.-Fácil de entender e implementar.

.-Ineficiente para listas grandes debido a su complejidad

. Hay varias técnicas básicas de ordenamiento cada una con su ventaja como por ejemplo;

1. Algoritmos de Ordenamiento Básicos

a) Burbuja (Bubble Sort)

Ayuda a comparar elementos adyacentes como intercambiar las posiciones si el orden esta incorrecto o mal ejecutado.

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:  # Intercambia si están en orden incorrecto
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

# Ejemplo de uso
arreglo = [34, 7, 23, 32, 5, 62]
print("Arreglo ordenado:", bubble_sort(arreglo))
```

```
PS C:\Users\alumnos\Downloads> & C:/Users/alumnos/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumnos/Downloads/EJEMPLOS
Arreglo ordenado: [5, 7, 23, 32, 34, 62]
PS C:\Users\alumnos\Downloads>
```

2. Ordenamiento por Selección (Selection Sort)

Esto ayuda a buscar el elemento de la fila menor y lo intercambia con el primer comando no ordenado.

```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]  # Desplaza elementos
            j -= 1
        arr[j + 1] = key  # Inserta en su lugar
    return arr

# Ejemplo de uso
arreglo = [34, 7, 23, 32, 5, 62]
print("Arreglo ordenado:", insertion_sort(arreglo))
```

```
PS C:\Users\alumnos\Downloads> & C:/Users/alumnos/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumnos/Downloads/EJEMPLOS
Arreglo ordenado: [5, 7, 23, 32, 34, 62]
```

3. Ordenamiento por Inserción (Insertion Sort)

Toma cada elemento y lo inserta en la posición en la parte ordenada.

```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:  # Encuentra el menor
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]  # Intercambio
    return arr

# Ejemplo de uso
arreglo = [34, 7, 23, 32, 5, 62]
print("Arreglo ordenado:", selection_sort(arreglo))
```

```
PS C:\Users\alumnos\Downloads> & C:/Users/alumnos/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumnos/Downloads/EJEMPLOS
Arreglo ordenado: [5, 7, 23, 32, 34, 62]
PS C:\Users\alumnos\Downloads>
```

Conclusión:

Los algoritmos son esenciales para la organización de datos, aunque los métodos básicos como los anteriores mostrados son fáciles de entender su eficiencia es limitada cuando se trabaja con altos volúmenes de datos. Aunque normalmente para programas mas complejos se recomienda **MERGE SORT y QUICK SORT**