



ALGORITMOS Y ESTRUCTURAS DE DATOS (TSDS)

ASIGNATURA:

ALGORITMOS Y ESTRUCTURAS DE DATOS

PROFESOR:

Ing. Lorena Chulde MSc.

PERÍODO ACADÉMICO:

2023-B

TAREA 5

TÍTULO:

**DISEÑO DE ALGORITMOS
ESTRUCTURAS ITERATIVAS O**

Nombre del estudiante:

Adrian Cadena, Santiago Cumbal, Elkin Diaz



2023-B

PROPÓSITO DE LA TAREA

Aplicar sentencias de algoritmos mediante las estructuras de cíclicas WHILE, DO-WHILE, FOR para la resolución de ejercicios sencillos.

Realizar la investigación sobre:

DISEÑO DE ALGORITMOS: Recursividad Análisis y complejidad de un algoritmo

Las indicaciones del formato adjunto el siguiente link:

La recursividad

La recursividad se refiere al proceso en el cual un método o función se invoca a sí mismo. Este enfoque se emplea en algoritmos de búsqueda, ordenación y puede reemplazar la necesidad de bucles. Es esencial considerar que, al aplicar esta técnica, es crucial garantizar la existencia de un punto final o caso base para evitar ejecuciones infinitas.

Reglas

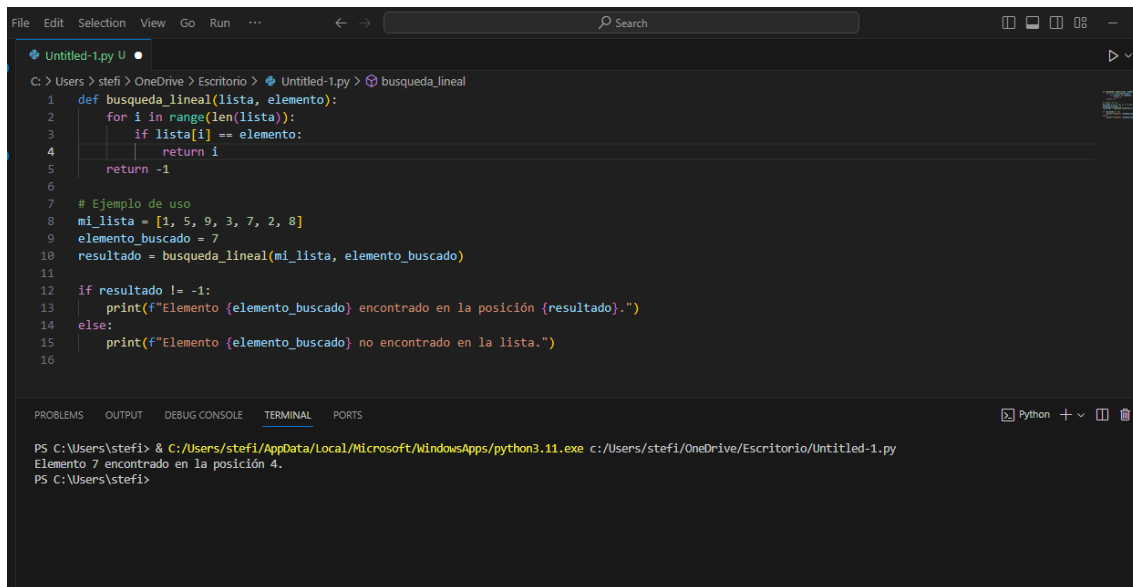
- 1.-Establecer un caso base de manera clara.
- 2.-Reducir el problema a instancias más pequeñas.
- 3.-Garantizar la producción de resultados en cada instancia.
- 4.-Elaborar un plan detallado.
- 5.-Gestionar adecuadamente las entradas válidas.

Análisis

El análisis de algoritmos es una parte fundamental de la ciencia de la computación que se ocupa de evaluar el rendimiento y la eficiencia de los algoritmos. Se centra en entender cómo se comporta un algoritmo en términos de tiempo y espacio a medida que el tamaño de la entrada aumenta.

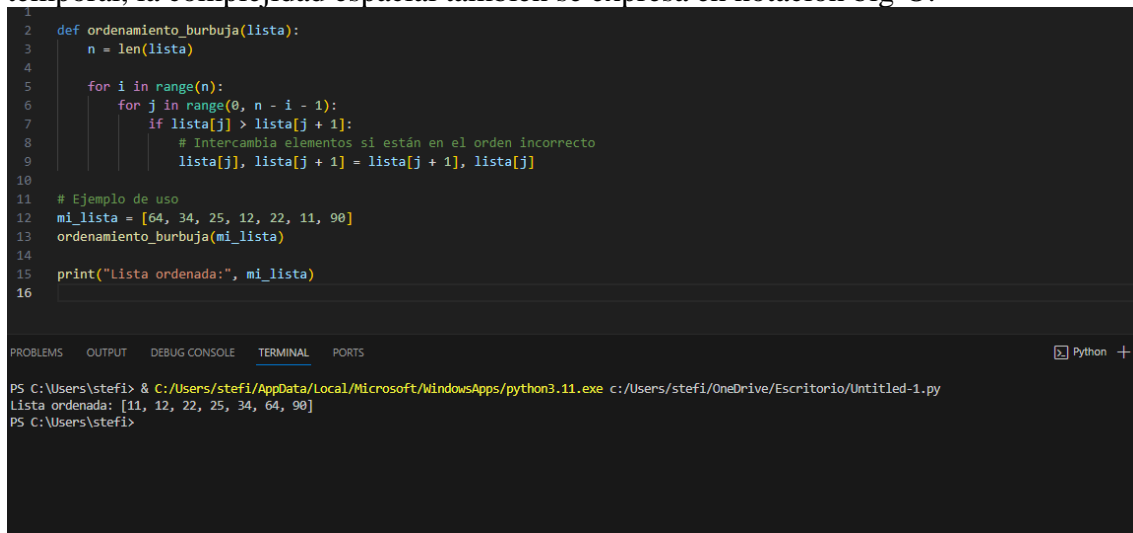
Hay dos tipos principales de análisis de algoritmos:

Análisis de tiempo (complejidad temporal): Este tipo de análisis se enfoca en determinar cuánto tiempo toma un algoritmo para completar su ejecución en función del tamaño de la entrada. La complejidad temporal se expresa comúnmente en notación big-O, que describe la tasa de crecimiento del tiempo de ejecución en el peor de los casos.



```
File Edit Selection View Go Run ... Search
Untitled-1.py U
C:\Users\stefi> OneDrive > Escritorio > Untitled-1.py > busqueda_lineal
1 def busqueda_lineal(lista, elemento):
2     for i in range(len(lista)):
3         if lista[i] == elemento:
4             return i
5     return -1
6
7 # Ejemplo de uso
8 mi_lista = [1, 5, 9, 3, 7, 2, 8]
9 elemento_buscado = 7
10 resultado = busqueda_lineal(mi_lista, elemento_buscado)
11
12 if resultado != -1:
13     print(f"Elemento {elemento_buscado} encontrado en la posición {resultado}.")
14 else:
15     print(f"Elemento {elemento_buscado} no encontrado en la lista.")
16
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + -
PS C:\Users\stefi> & C:/Users/stefi/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/stefi/OneDrive/Escritorio/Untitled-1.py
Elemento 7 encontrado en la posición 4.
PS C:\Users\stefi>
```

Análisis de espacio (complejidad espacial): Este tipo de análisis se centra en cuánta memoria (espacio) utiliza un algoritmo en función del tamaño de la entrada. Al igual que la complejidad temporal, la complejidad espacial también se expresa en notación big-O.



```
1
2 def ordenamiento_burbuja(lista):
3     n = len(lista)
4
5     for i in range(n):
6         for j in range(0, n - i - 1):
7             if lista[j] > lista[j + 1]:
8                 # Intercambia elementos si están en el orden incorrecto
9                 lista[j], lista[j + 1] = lista[j + 1], lista[j]
10
11 # Ejemplo de uso
12 mi_lista = [64, 34, 25, 12, 22, 11, 90]
13 ordenamiento_burbuja(mi_lista)
14
15 print("Lista ordenada:", mi_lista)
16
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + -
PS C:\Users\stefi> & C:/Users/stefi/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/stefi/OneDrive/Escritorio/Untitled-1.py
Lista ordenada: [11, 12, 22, 25, 34, 64, 90]
PS C:\Users\stefi>
```

Complejidad

La complejidad de los algoritmos se refiere a la medida del rendimiento de un algoritmo en términos de recursos computacionales que utiliza. Los dos tipos principales de complejidad son la complejidad temporal y la complejidad espacial.

Complejidad Temporal (Tiempo): La complejidad temporal de un algoritmo se refiere a la cantidad de tiempo que tarda en ejecutarse en función del tamaño de la entrada. Se expresa comúnmente en notación big-O, que describe el comportamiento asintótico del algoritmo en el peor de los casos.

Es importante analizar la complejidad de los algoritmos porque permite tomar decisiones informadas sobre qué algoritmo utilizar en una situación dada. Algoritmos con menor complejidad temporal y espacial son en general preferibles, especialmente cuando se trata de grandes conjuntos de datos o recursos limitados. Sin embargo, la elección del algoritmo también depende del contexto y de otros factores como la legibilidad del código y la simplicidad del diseño.

CONCLUSIONES

La recursividad es una herramienta poderosa en la programación de algoritmos, que puede simplificar la solución de problemas al dividirlos en partes más pequeñas y manejables. Sin embargo, se debe usar con precaución, considerando tanto la elegancia del diseño como la eficiencia en términos de complejidad temporal y espacial. En última instancia, la elección de utilizar la recursividad dependerá de la naturaleza del problema y de encontrar un equilibrio entre la claridad del código y la eficiencia del algoritmo.

[1]

[2]

1 Bibliografía

- [1 Khan Academy, «Khan Academy,» [En línea]. Available:
] <https://es.khanacademy.org/computing/computer-science/algorithms/recursive-algorithms/a/recursion>.
- [2 R. Academy. [En línea]. Available:
] <https://runestone.academy/ns/books/published/pythoned/AlgorithmAnalysis/QueEsAnalisisDeAlgoritmos.html>.