

# Trabajo Integrador – Programación I

---

Tema: Árboles binarios con listas

Alumnos: María Cecilia Fontana – cefontana16@gmail.com / Iván Domínguez – ivan\_dominguez2016@hotmail.com

Materia: Programación I

Fecha de entrega: 09 de junio de 2025

## Índice

1. Introducción
2. Marco Teórico
3. Caso Práctico
4. Metodología utilizada
5. Resultados obtenidos
6. Conclusiones
7. Bibliografía
8. Anexos

## 1. Introducción

Este trabajo tiene como objetivo implementar y analizar el funcionamiento de un árbol binario de búsqueda utilizando listas y funciones en Python, sin el uso de clases. Se eligieron valores numéricos enteros para facilitar la representación del árbol y observar su estructura mediante distintos recorridos. En programación entender este tema es muy importante ya que ayuda a la optimización de procesos.

## 2. Marco Teórico

Un árbol binario de búsqueda es una estructura jerárquica donde cada nodo puede tener como máximo dos hijos, izquierdo y derecho. Se le llama nodo a cada elemento que contiene el árbol, son centros de información porque contienen distintos tipos de datos. Éstos elementos están unidos entre sí por líneas denominadas ramas. Los nodos se clasifican de 2 maneras:

1)Según su relación con otros nodos:

Nodo padre: se utiliza el término para llamar a todos los nodos que tienen al menos un hijo.

Nodo hijo: los hijos son todos aquellos nodos que tienen un padre.

Nodo hermando: Son aquellos que comparten un mismo padre en común dentro de la estructura.

2)Según su posición dentro del árbol:

Nodo raíz: se refiere al primer nodo de un árbol. Sólo un nodo del árbol puede ser la raíz.

Nodo hoja: son aquellos nodos que no tienen hijos por lo cual siempre se sitúan en el extremo de la estructura.

Nodo rama: son nodos que tienen padre y al menos un hijo.

El árbol a partir del nodo raíz, se divide en dos sub-árboles. El sub-árbol izquierdo y el sub-árbol derecho.

Entonces un árbol binario de búsqueda es una estructura de datos donde: cada nodo tiene un valor, un sub-árbol izquierdo y un sub-árbol derecho. Los valores en el sub-árbol izquierdo son menores que el valor del nodo. Y los valores en el sub-árbol derecho son mayores o iguales al valor del nodo.

La representación elegida en este trabajo en python es mediante listas con la forma: [valor, subárbol izquierdo, subárbol derecho]. Se utilizan funciones recursivas para insertar y recorrer el árbol en tres formas: inorden, preorden y postorden.

Para el recorrido inorden primero se recorre el sub-árbol izquierdo, procesa la raíz y luego recorre el sub-árbol derecho.

Para el recorrido preorden procesa la raíz, recorre el sub-árbol izquierdo y luego el derecho.

Para el recorrido postorden primero recorre el sub-árbol izquierdo, luego el sub-árbol derecho y procesa la raíz.

Existen distintos tipos de árboles binarios,(además del tipo que vamos a ver en este trabajo práctico que es el de búsqueda):

- Árboles binarios distintos • Árboles binarios similares • Árboles binarios equivalentes • Árboles binarios completos • Árboles binarios llenos • Árboles binarios degenerados • Árboles binarios de búsqueda • Árboles equilibrados .

El código a implementar en python es el siguiente:

```
# Trabajo Integrador - Árboles con listas (versión con números)
```

```
# Esta función crea un nodo del árbol. Cada nodo es una lista:
```

```
# [valor, subárbol izquierdo, subárbol derecho]
```

```
def crear_nodo(valor):
```

```
    return [valor, None, None]
```

```
# Inserta un valor en el árbol comparando con el nodo actual.
```

```
# Si es menor, va a la izquierda; si no, a la derecha.
```

```

def insertar_nodo(arbol, valor):
    if arbol is None:
        return crear_nodo(valor)
    if valor < arbol[0]:
        arbol[1] = insertar_nodo(arbol[1], valor) # izquierda
    else:
        arbol[2] = insertar_nodo(arbol[2], valor) # derecha
    return arbol

# Recorre el árbol en orden: izquierda → valor → derecha
def inorden(arbol):
    if arbol is not None:
        inorden(arbol[1])
        print(arbol[0], end=" ")
        inorden(arbol[2])

# Recorre el árbol en preorden: valor → izquierda → derecha
def preorden(arbol):
    if arbol is not None:
        print(arbol[0], end=" ")
        preorden(arbol[1])
        preorden(arbol[2])

# Recorre el árbol en postorden: izquierda → derecha → valor
def postorden(arbol):
    if arbol is not None:
        postorden(arbol[1])
        postorden(arbol[2])
        print(arbol[0], end=" ")

# Programa principal

# Lista de números a insertar en el árbol
numeros = [50, 30, 70, 20, 40, 60, 80]

# Árbol vacío al comienzo
arbol = None

# Insertamos los números uno por uno
for numero in numeros:
    arbol = insertar_nodo(arbol, numero)

# Mostramos los recorridos
print("\nRecorrido Inorden:")
inorden(arbol)

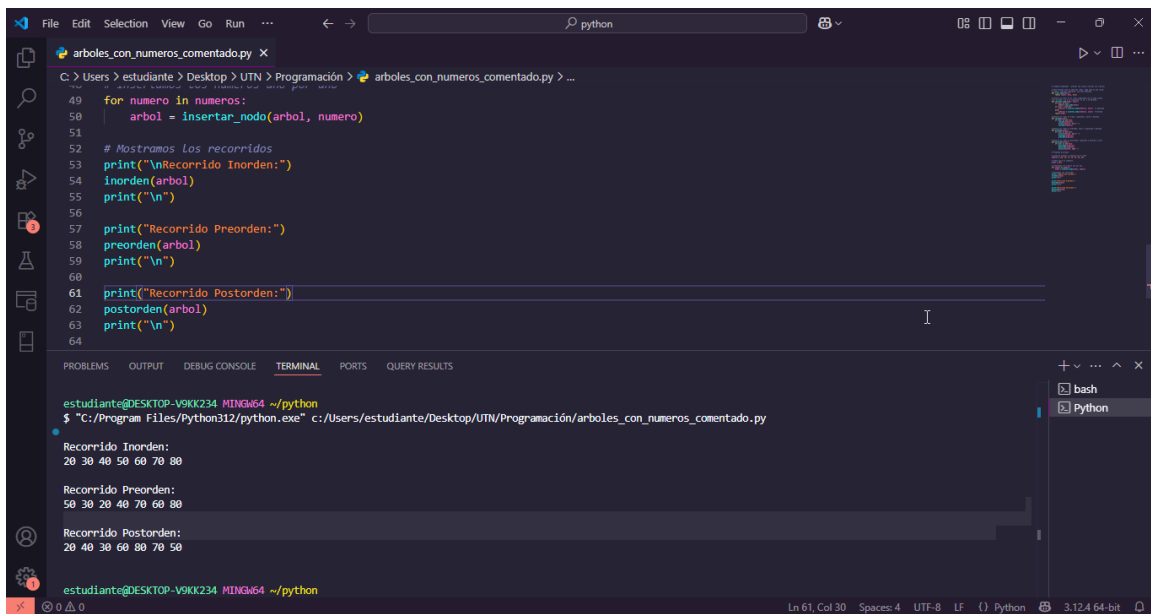
```

```
print("\n")
```

```
print("Recorrido Preorden:")  
preorden(arbol)  
print("\n")
```

```
print("Recorrido Postorden:")  
postorden(arbol)  
print("\n")
```

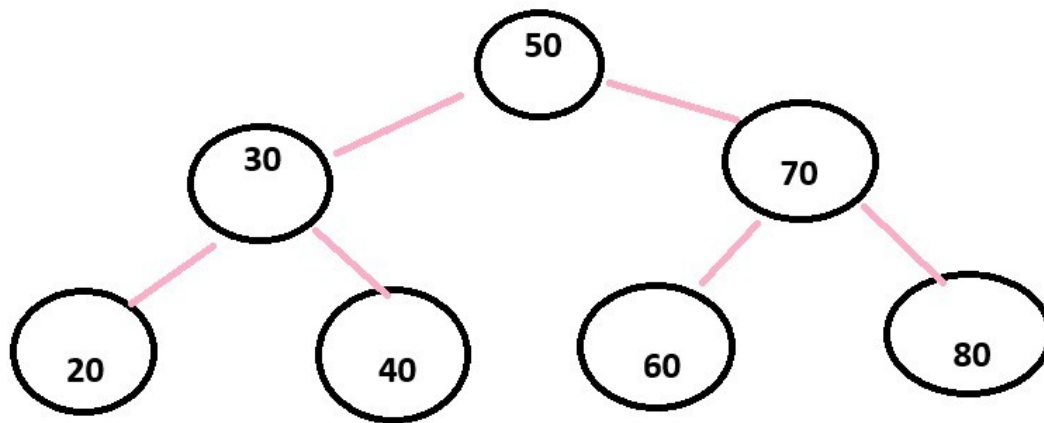
Obtenemos los siguientes resultados:



The screenshot shows a Visual Studio Code editor window with a Python file named `arboles_con_numeros_comentado.py`. The code defines a function `insertar_nodo` and performs three traversals of a binary tree: Inorden, Preorden, and Postorden. The terminal output shows the results of these traversals for a tree with nodes 20, 30, 40, 50, 60, 70, and 80.

```
arboles_con_numeros_comentado.py X  
C:\Users\estudiante\Desktop\UTN\Programación> python arboles_con_numeros_comentado.py ...  
49 for numero in numeros:  
50     arbol = insertar_nodo(arbol, numero)  
51  
52 # Mostramos los recorridos  
53 print("\nRecorrido Inorden:")  
54 inorden(arbol)  
55 print("\n")  
56  
57 print("Recorrido Preorden:")  
58 preorden(arbol)  
59 print("\n")  
60  
61 print("Recorrido Postorden:")  
62 postorden(arbol)  
63 print("\n")  
64  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS  
estudiante@DESKTOP-V9KK234 MINGW64 ~/python  
$ "C:/Program Files/Python312/python.exe" c:/Users/estudiante/Desktop/UTN/Programación/arboles_con_numeros_comentado.py  
Recorrido Inorden:  
20 30 40 50 60 70 80  
  
Recorrido Preorden:  
50 30 20 40 70 60 80  
  
Recorrido Postorden:  
20 40 30 60 80 70 50  
  
estudiante@DESKTOP-V9KK234 MINGW64 ~/python  
Ln 61, Col 30 Spaces: 4 UTF-8 LF Python 3.12.4 64-bit
```

Estructura del árbol utilizada en el caso práctico (Punto 3):



**números = 50, 30, 70, 20, 40, 60,80**

### 3. Caso Práctico

La lista de valores utilizada es: 50, 30, 70, 20, 40, 60, 80. A medida que se insertan en el árbol, se respeta el orden definido por la estructura binaria de búsqueda. El árbol resultante fue recorrido utilizando funciones específicas que imprimen los valores según cada tipo de recorrido. Este procedimiento permite verificar visualmente que la estructura se formó correctamente.

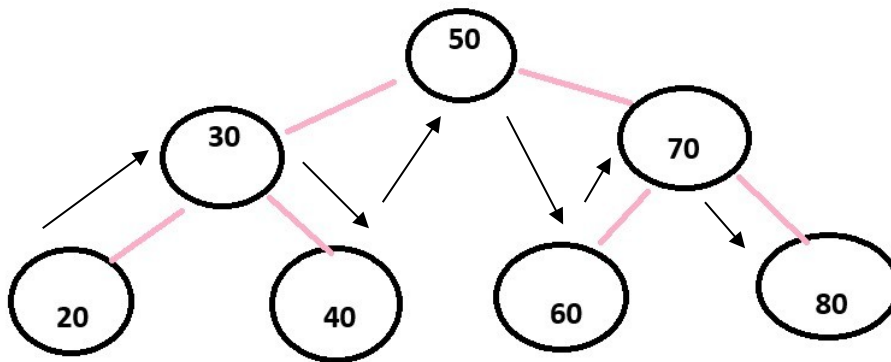
### 4. Metodología utilizada

1. Definir la estructura del nodo como lista con valor y referencias a izquierda y derecha.
2. Crear funciones para insertar nodos en el árbol respetando el orden binario.
3. Desarrollar funciones recursivas para recorrer el árbol en inorden, preorden y postorden.
4. Ejecutar el programa con una lista de valores predefinidos.
5. Verificar los resultados de los recorridos para asegurar que el árbol fue construido correctamente.

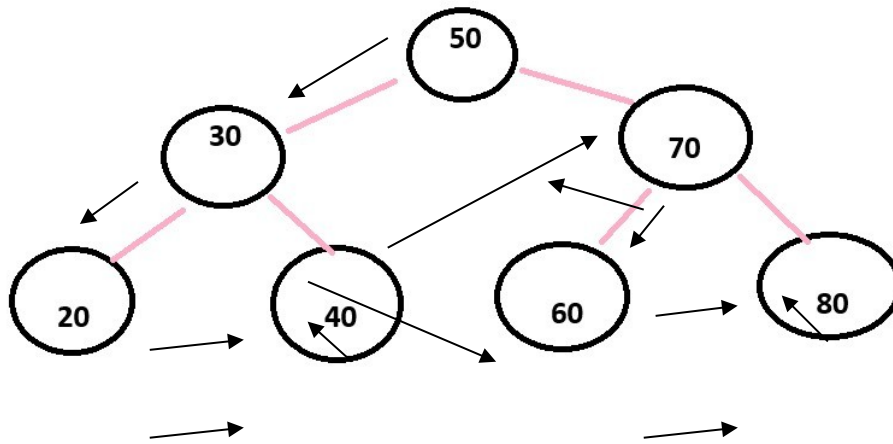
### 5. Resultados obtenidos

El recorrido inorden mostró los valores en orden creciente, lo cual confirma que la inserción fue exitosa. El preorden y postorden permitieron observar el recorrido desde la raíz y desde las hojas respectivamente. El uso de listas en lugar de objetos no afectó la lógica del árbol, lo que demuestra la versatilidad del lenguaje Python para representar estructuras complejas con herramientas simples.

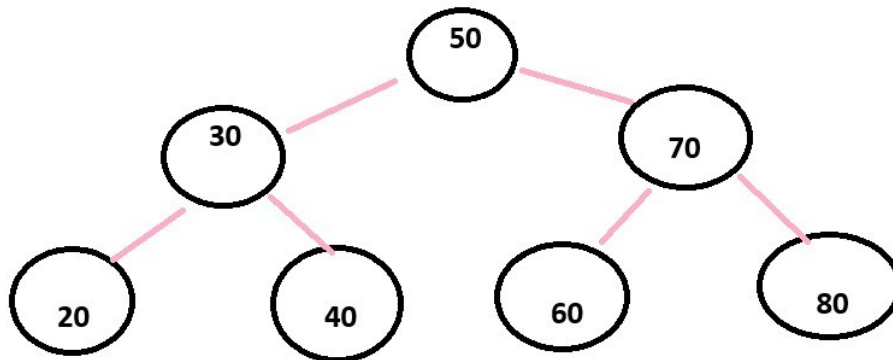
**Recorrido Inorden:**



Recorrido Preorden



Recorrido Posorden



## 6. Conclusiones

La experiencia de trabajar con árboles binarios sin clases permitió fortalecer el concepto de recursividad y estructuras dinámicas. El código se mantuvo sencillo y accesible, ideal para un primer acercamiento a esta estructura. Además, se logró cumplir con el objetivo del trabajo integrador usando solamente funciones, listas y condicionales.

## 7. Bibliografía

- Apuntes de clase de Programación I
- Documentación oficial de Python

- Videos de la cátedra sobre árboles binarios
- Clase8-Arbolespdf-Universidad Veracruzana