	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Jorge Solano
Asignatura:	Estructura de Datos y Algoritmos II
Grupo:	2
No de Práctica(s):	8
Integrante(s):	Martínez Ostoa Néstor Iván
Semestre:	2019-1
Fecha de entrega:	12/Octubre/2018
Observaciones:	

CALIFICACIÓN: _____

Introducción:

Las estructuras de datos de tipo árbol son muy relevantes en las Ciencias de la Computación, principalmente por la eficiencia que manejan a la hora de buscar y al mismo tiempo que otras operaciones básicas, como insertar por ejemplo. Los árboles son estructuras de datos no lineales, sin embargo, cuentan con solo un antecesor y muchos predecesores. En el caso particular de los árboles binarios, estos cuentan con solo un antecesor y solo dos sucesores.

En esta práctica analizaremos tres recorridos posibles dentro de un árbol binario y algunas de sus reglas. Estos son los siguientes:

- Recorrido Infijo:
  - Recorrer el subárbol izquierdo
  - Visitar la raíz
  - Recorrer el subárbol derecho

- Recorrido Prefijo:
  - Visitar la raíz
  - Recorrer el subárbol izquierdo
  - Recorrer el subárbol derecho

- Recorrido Sufijo:
  - Recorrer el subárbol izquierdo
  - Recorrer el subárbol derecho
  - Visitar la raíz

Objetivo:

Implementar una estructura de árbol binario junto con funciones prefijas, posfijas e infijas para recorrer un árbol binario. Aunado a esto, obtener la complejidad de recorrido de un árbol.

Desarrollo:

1. Implementación en Python

1.1 Recorrido infijo

```
def orderedTreeWalk(x, tree):
    global orderCounter
    if x < len(tree):
        orderCounter += 1
        orderedTreeWalk(left(x), tree)
        #print(tree[x])
        orderedTreeWalk(right(x),tree)
```

1.2 Recorrido prefijo

```
def preOrderedTreeWalk(x, tree):
    global preOrderCounter
    if x < len(tree):
        preOrderCounter += 1
        #print(tree[x])
        preOrderedTreeWalk(left(x),tree)
        preOrderedTreeWalk(right(x),tree)
```

1.3 Recorrido sufijo

```
def postOrderedTreeWalk(x, tree):
    global posOrderCounter
    if x < len(tree):
        posOrderCounter += 1
        postOrderedTreeWalk(left(x),tree)
        postOrderedTreeWalk(right(x),tree)
        #print(tree[x])
```

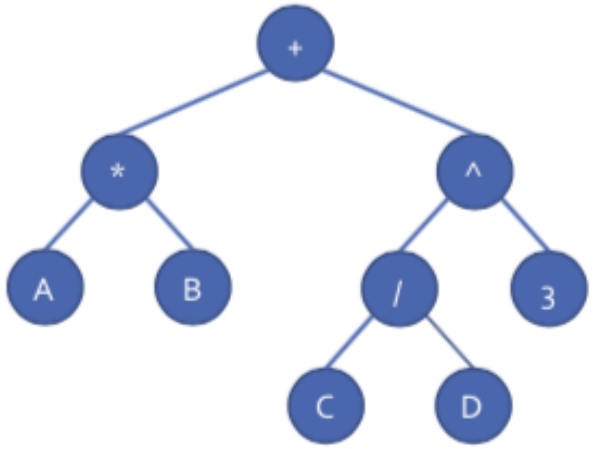
1.4 Funciones adicionales:

```
def left(x):
    return 2*x

def right(x):
    return (2*x) + 1
```

2. Pruebas de funcionamiento

Árbol implementado



Representación y recorrido en Python:

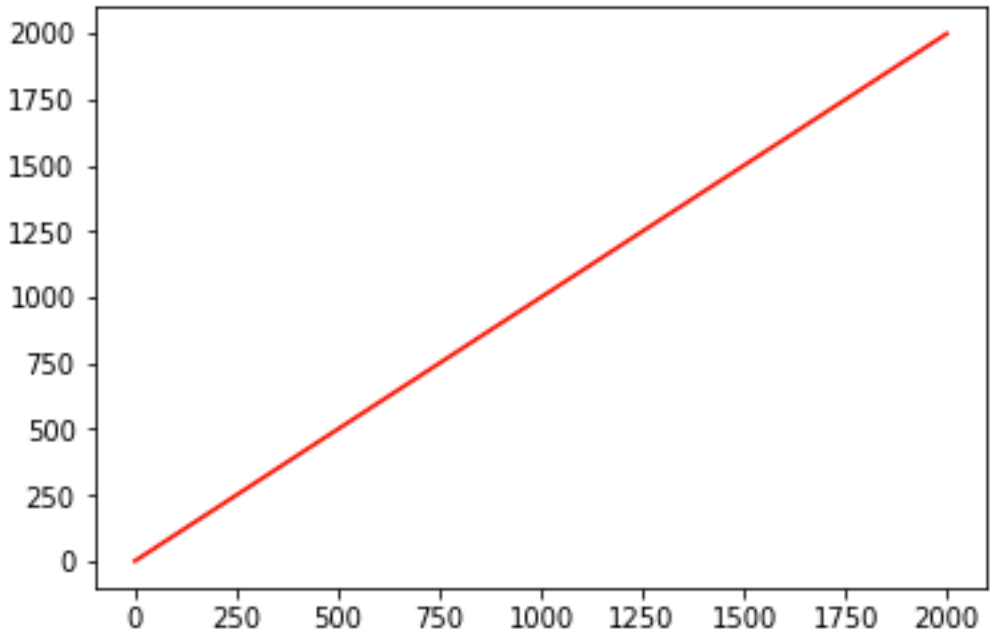
```
Inorder Tree Walk
A
*
B
+
C
/
D
^
3
```

```
Preorder Tree Walk
+
*
A
B
^
/
C
D
3
```

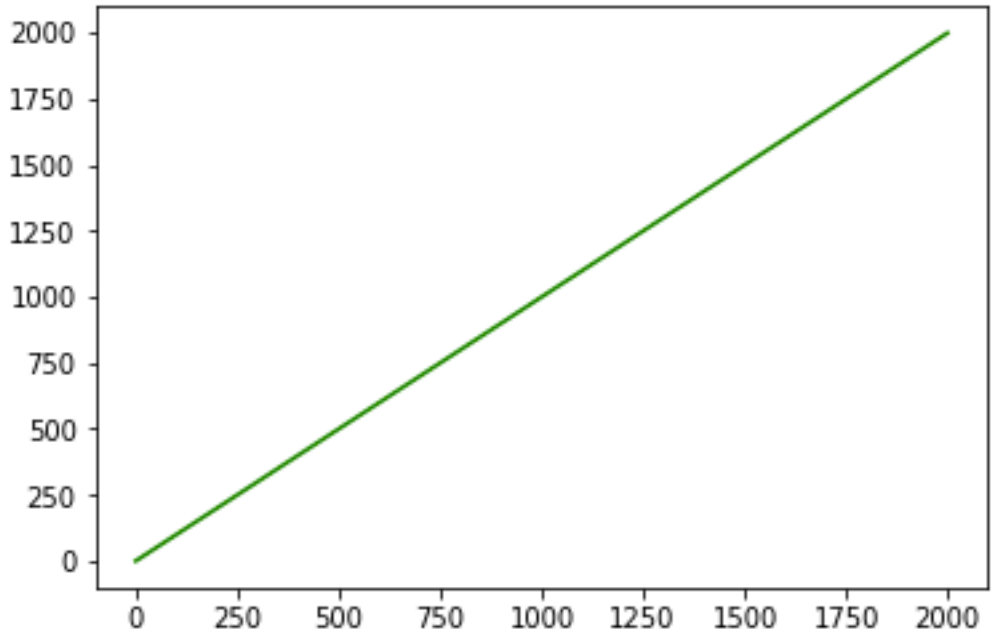
```
Postorder Tree Walk
A
B
*
C
D
/
3
^
+
```

3. Gráficas para 2000 datos

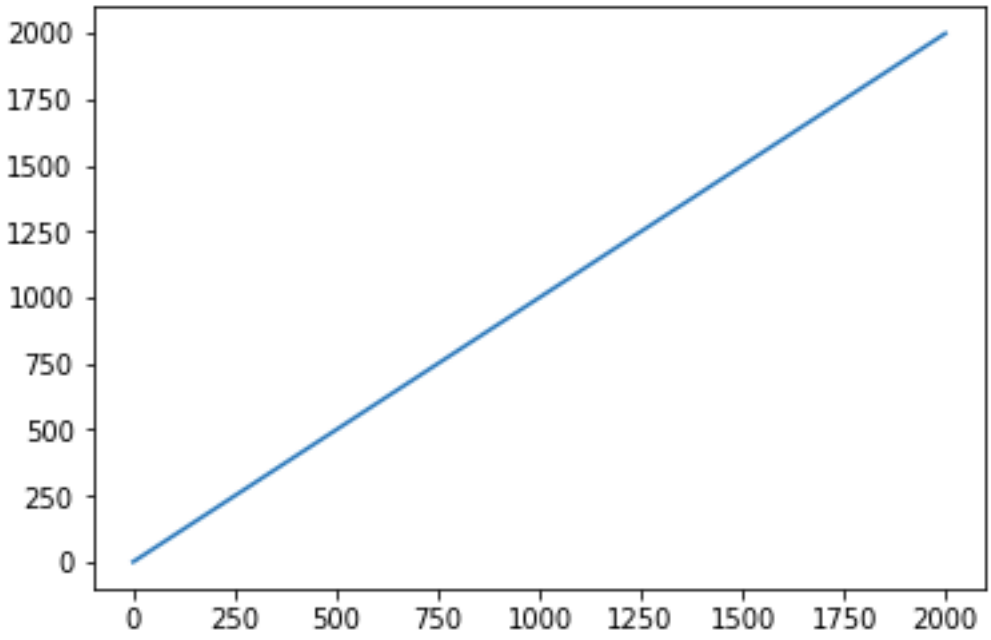
3.1 Recorrido infijo



3.2 Recorrido prefijo



3.3 Recorrido sufijo



Conclusiones

En los tres casos vemos un comportamiento lineal, puesto que al hablar de un árbol, y un recorrido, el algoritmo tiene que recorrer todo el árbol binario hasta los nodos hojas, o empezando desde ahí.

La complejidad depende completamente del número de elementos que tenga un árbol binario.