Actividad 0:

Exploración de Algoritmos y Problemas Computacionales

Diego Andrés Figueroa Peart
A01660987

Curso:

Análisis y diseño de algoritmos avanzados
(TC2038.652)

Profesor:

Cristhian Alejandro Ávila Sánchez

Fecha de entrega:

30 de agosto de 2023

Actividad 0: Exploración de Algoritmos y Problemas Computacionales

Algoritmo 1: Insertion Sort

Insertion sort es un algoritmo de ordenamiento, es decir, un algoritmo computacional que ordena los elementos de un arreglo de acuerdo a un criterio especificado, usualmente de menor a mayor.

Pseudocódigo:

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while
```

Funcionamiento:

Toma el segundo elemento de la lista y lo compara con el primero; si este es mayor, los intercambia; si no, los deja como están. Después toma el tercer elemento y lo compara con los anteriores, intercambiándolos si el anterior es mayor hasta que se encuentre uno que sea menor o igual. Procede así sucesivamente hasta llegar al último elemento de la lista, después de colocar este en su lugar, la lista estará ordenada de menor a mayor.

Complejidad:

Mejor caso: $O(n)$

Caso promedio: $O(n^2)$

Peor caso: $O(n^2)$

Implementación en Python:

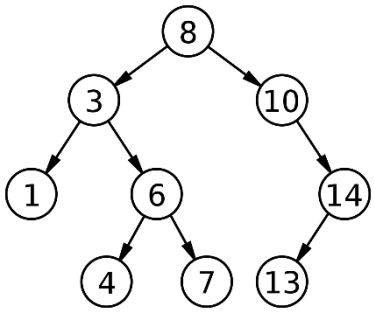
```python
def insertionSort(array):
    for i in range(1, len(array)):
        j = i
        while j > 0 and array[j-1] > array[j]:
            array[j-1], array[j] = array[j], array[j-1]
            j -= 1
    return array
```

Número de pasos para 10, 100, 1,000 y 10,000 elementos:

```
c:\Users\Diego Figueroa\Desktop>C:/Programming/Python/python.exe "c:/Users/Diego Figueroa/Desktop/insertionSort.py"
Tamaño: 10 Pasos: 25
Tamaño: 100 Pasos: 2307
Tamaño: 1000 Pasos: 250114
Tamaño: 10000 Pasos: 24773187
```

Algoritmo 2: Insertar en árbol binario de búsqueda

Este algoritmo busca insertar un nuevo dato dentro de un árbol binario de búsqueda; es decir, una estructura de datos que cumple con las características de un BST, que el valor de cualquier nodo es mayor que el valor de cualquier subnodo a su izquierda y menor al de cualquier subnodo a su derecha. Ejemplo:



Pseudocódigo

```
if insertion point is found
  create new vertex
if value to be inserted < this key
  go left
else if value to be inserted > this key
  go right
else increment frequency
```

Funcionamiento:

Se intenta insertar el dato en la raíz del árbol, si esta ya está ocupada, y si el dato es menor que el valor que la ocupa, se intenta insertar en el subnodo izquierdo; si este es mayor, se intenta insertar en el subnodo derecho. Este proceso se repite hasta que se encuentre un espacio vacío adecuado para el valor. Si el dato en algún momento es igual al valor de un nodo, este se descarta ya que un BST no permite datos duplicados.

Complejidad:

Mejor caso: O(h) h = altura del árbol binario

Caso promedio: O(log n)

Peor caso: O(n)

Implementación en Python:

```
import random

class Nodo:
    def __init__(self, dato):
        self.izquierda = None
        self.derecha = None
        self.dato = dato

class Arbol:
    def __init__(self):
        self.raiz = None

    def insertar(self, dato):
        if not self.raiz:
            self.raiz = Nodo(dato)
            return 1
        else:
            return self._insertar(dato, self.raiz, 1)

    def _insertar(self, dato, nodo, n):
        if dato < nodo.dato:
            if not nodo.izquierda:
                nodo.izquierda = Nodo(dato)
                return n + 1
            else:
                return self._insertar(dato, nodo.izquierda, n + 1)
```

```
        elif dato > nodo.dato:
            if not nodo.derecha:
                nodo.derecha = Nodo(dato)
                return n + 1
            else:
                return self._insertar(dato, nodo.derecha, n + 1)
        else:
            return n + 1
```

Número de pasos para 10, 100, 1,000 y 10,000 elementos: