

# Informe de Laboratorio 04

## Tema: Sort y Listas Enlasadas

Tabla 1: Mi tabla extendida

### INTRODUCCIÓN

aquí ira la intro

### MARCO CONCEPTUAL

aquí ira la MARCO CONCEPTUAL

### SOLUCIONES Y PRUEBAS

Ejercicio 1

//taylor

Ejercicio 2

Para lograr ejecutar el algoritmo de ordenamiento por insercion primero fue necesario crear la clase ListaDoble el cual tenia 2 nodos, uno inicial y otro final. Con esto se lograba recorrer la lista de el fin al inicio y en viceversa.

El metodo addFinal permitia insertar un nodo al final y el otro metodo addInicio, al inicio, dependeiendo si es que estaba vacia o no la lista.

```
public void addInicio(E x){
    if(!isEmpty()){
        inicio=new Node<>(x,inicio,null);
        inicio.getNextNode().setPreviousNode(inicio);
    }else {
        inicio=fin= new Node<>(x);
    }
}
```

```
public void addFinal(E x){
    if(!isEmpty()){
        fin=new Node<>(x,null,fin);
        fin.getPreviousNode().setNextNode(fin);
    }else {
        inicio=fin= new Node<>(x);
    }
}
```

Asimismo se agrego el metodo isEmpty para saber si estaba vacia y se sobrescribio el metodo toString para mostrar retornar la data del 1er al ultimo nodo. Los getters y setters se omitieron en la imagen.

```
public boolean isEmpty() {  
    return inicio == null;  
}  
//getters y setters de inicio y fin  
@Override  
public String toString() {  
    String text="";  
    if (!isEmpty()){  
        Node<E> aux = inicio;  
        while (aux!=null){  
            text += aux.getData()+" ";  
            aux = aux.getNextNode();  
        }  
    }  
    return text;  
}
```

Se creo la clase Test para implementar el insertionSort y generarPeorCaso. El metodo insertionSort se modifico haciendo que se verificara si es que la lista no estuviera vacia y en caso fuera asi recorria con un for en ves de un while hasta que el nodo siguiente fuera nulo. Dentro de este while habia otro que intercambiaba la data, en este caso Integer, de los nodos haciendo que los menores quedaran al inicio y los mayores al final. Retornaba el tiempo que se demoraba haciendo la operacion.

```
if (!lista.isEmpty()) {  
    keyNode = lista.getInicio().getNextNode();  
    while (keyNode != null) {  
        Integer key = keyNode.getData();  
        currentNode = keyNode.getPreviousNode();  
        while (currentNode != null && currentNode.getData() > key) {  
            currentNode.getNextNode().setData(currentNode.getData());  
            currentNode = currentNode.getPreviousNode();  
        }  
        if (currentNode == null) lista.getInicio().setData(key);  
        else currentNode.getNextNode().setData(key);  
        keyNode = keyNode.getNextNode();  
    }  
}
```

Listing 1: Retorno de insertionSort

```
return nano_endTime - nano_startTime;
```

El metodo generarPeorCaso se modifico para que retornara una ListaDoble donde su primer elemento fuera el tamaño-1 y el ultimo 1, donde tamaño era el valor que se recibia por argumento, quedando asi el peor caso para hacer insertionSort.

```
ListaDoble<Integer> listaDoble = new ListaDoble<>();  
for (int i = 0; i < tamaño; i++)  
    listaDoble.addFinal(tamaño - i);  
return listaDoble;
```

## LECCIONES APRENDIDAS Y CONCLUSIONES

aquí irá la LECCIONES APRENDIDAS Y CONCLUSIONES

## REFERENCIAS Y BIBLIOGRAFÍA

aquí irá la REFERENCIAS Y BIBLIOGRAFÍA

### 0.1. USAR COMO GUIA PARA EL INFORME

- Se tuvieron que implementar las funciones negative, join y under. En el código se detalla lo que hacen.

Listing 2: Picture version 1

```
1 import colors  
2 from colors import *  
3 class Picture:  
4     def __init__(self, img):  
5         self.img = img;  
6     def __eq__(self, other):  
7         return self.img == other.img  
8     def _invColor(self, color):  
9         if color not in inverter:  
10            return color  
11        return inverter[color]  
12    def negative(self):  
13        """ Devuelve un negativo de la imagen """  
14        neg = []  
15        for fila in self.img:  
16            cadena = ""  
17            for color in fila:  
18                cadena += self._invColor(color)  
19            neg.append(cadena)  
20        return Picture(neg)  
21    def join(self, p):  
22        """ Devuelve una nueva figura poniendo la figura del argumento  
23            al lado derecho de la figura actual """  
24        juntos = []  
25        for i in range(len(self.img)):   
26            juntos.append(self.img[i] + p.img[i])  
27        return Picture(juntos)  
28    def under(self, p):  
29        """ Devuelve una nueva figura poniendo la figura p sobre la  
30            figura actual """  
31        return Picture(self.img[::] + p.img[::])
```

Para implementar el negativo se cambia el color de cada carácter, a excepción del espacio ya que este no tiene inverso con doble for(1 anidado). Juntando estos caracteres por medio en una cadena y después juntando esta cadena al nuevo arreglo con el método append.

## Listing 3: Ejercicio 2 a

```
1 from interpreter import draw
2 from chessPictures import *
3
4 draw(knight.join(knight.negative()).under(knight.negative().join(knight)))
```

Para poder hacer que se dibuje como se mostrara el la figura de abajo se tuvo de juntar un caballo blanco con uno negro en la primera fila; para ello se utilizo join y negative. Para la segunda fila se necesito saltar a la siguiente fila por lo que se utilizo el metodo under.



Figura 1: Ejecución exitosa ejercicio 2 a

## 0.2. Pregunta: Explique: ¿Para qué sirve el directorio pycache?

- Sirve para guardar los compilados de python, asi como en java al ejecutarlos se crean los .class en python se crean los .pyc. Esto se hace automaticamente ya que se importan modulos de otras clases como se da en este caso.