| | |
|---|---|
| **Docente:** | CÉSAR AUGUSTO PENILLA SEPÚLVEDA |
| **Teléfono (opcional):** | 3122067624 |
| **Correo electrónico:** | cesar.penilla@correounivalle.edu.co |



Complejidad temporal: mide el número de unidades de tiempo que necesita un algoritmo (o una simple sentencia) para resolver un problema con una entrada de tamaño N, y se denota por T(N). Se utiliza como unidad mínima de coste temporal (1 como la unidad mínima de coste), el coste que corresponde a hacer la operación más sencilla, por ejemplo, una asignación. De esta manera, se minimiza la influencia de la arquitectura del ordenador, el sistema operativo, el compilador, etc., en el cálculo de la complejidad. Consecuentemente, el coste estimado depende más del número de operaciones realizadas que realmente de las operaciones en concreto, y sólo dependerá de N cuando el número de operaciones también dependa de aquélla, como repetir un bloque de instrucciones dentro de un bucle que depende de N.

***Ejercicios Complejidad Computacional***

Realice el cálculo de Complejidad Espacial S(n) y la complejidad Temporal T(n) para los siguientes Algoritmos, recuerde que los algoritmos iterativos pueden tener un número definido de entradas o no (calcular ambos casos).

1.

```python
def main():

    # Pedir información al usuario
    nombre = input("Digite el nombre del empleado: ")
    cedula = input("Digite la cedula del empleado: ")
    edad = int(input("Digite la edad del empleado: "))
    salario = int(input("Digite el salario del empleado: "))

    # Crear un objeto de la clase Empleado
    mi_empleado = Empleado(nombre, cedula, edad, salario)

    # Calcular las contribuciones
    salud = mi_empleado.salario * 0.125
    pension = mi_empleado.salario * 0.16
    riesgo = mi_empleado.salario * 0.035

    # Mostrar los resultados
    print(f"Deberá pagar por concepto de Salud: {salud}")
    print(f"Deberá pagar por concepto de pensión: {pension}")
    print(f"Deberá pagar por concepto de Riesgos Profesionales ARL: {riesgo}")
```

2.

```python
def main():
    vector1 = [None] * 4
    vector2 = [None] * 4
    vector3 = [None] * 4

    # Leer valores del usuario
    for i in range(len(vector1)):
        valor = int(input("Ingrese el valor {} para el vector 1: ".format(i + 1)))
        vector1[i] = valor

    for i in range(len(vector2)):
        valor = int(input("Ingrese el valor {} para el vector 2: ".format(i + 1)))
        vector2[i] = valor

    # Sumar elementos de los vectores
    for i in range(len(vector3)):
        vector3[i] = suma(vector1[i], vector2[i])

    # Imprimir resultado
    print("El resultado de la suma es: ")
    for valor in vector3:
        print(valor)


def suma(num1, num2):
    return num1 + num2
```

3.

Matriz.py

```python
def inicializar(n):
  matriz = []
  valor = 1
  for i in range(n):
    matriz.append([])
    for j in range(n):
      matriz[i].append(valor)
      valor += 1
  return matriz

def obtener_diagonal_principal(matriz):
  n = len(matriz)
  diago_principal = []
  for i in range(n):
    for j in range(n):
      if i == j:
        diago_principal.append(matriz[i][j])
  return diago_principal

def obtener_diagonal_secundaria(matriz):
  n = len(matriz)
  diago_secundaria = []
  for i in range(n):
    for j in range(n):
      if j == (n-1-i):
        diago_secundaria.append(matriz[i][j])
  return diago_secundaria

def visualizar_array(array):
  for valor in array:
    print(valor)

#Flujo Principal
matriz = inicializar(5)
print("\nDiagonal Principal")
diago_principal = obtener_diagonal_principal(matriz)
visualizar_array(diago_principal)
diago_secundaria = obtener_diagonal_secundaria(matriz)
print("\nDiagonal Secundaria")
visualizar_array(diago_secundaria)
```

## Ejercicios Listas

1. Desarrollar un programa que permita recorrer por puntero una lista simple de 6 elementos, y permita Agregar un nodo al inicio de dicha lista.
2. Desarrollar un programa que permita recorrer por puntero una lista simple de 16 elementos, y permita Eliminar un nodo al inicio de dicha lista.
3. Desarrollar un programa que permita recorrer por puntero una lista simple de 10 elementos, los cuales son números múltiplos de 4, y permita Eliminar un nodo después del primer elemento con un valor mayor a 21.
4. Desarrollar un programa que permita recorrer por puntero una lista simple de 20 elementos, y permita Eliminar un nodo entre los nodos 9 y 11.
5. Desarrollar un programa que permita recorrer por puntero una lista simple de 10 elementos, y permita agregar un nodo al final de dicha lista.
6. Desarrollar un programa que permita recorrer por puntero una lista simple de 10 elementos, los cuales son números múltiplos de 5, y permita Eliminar un nodo antes del primer elemento con un valor mayor a 37.