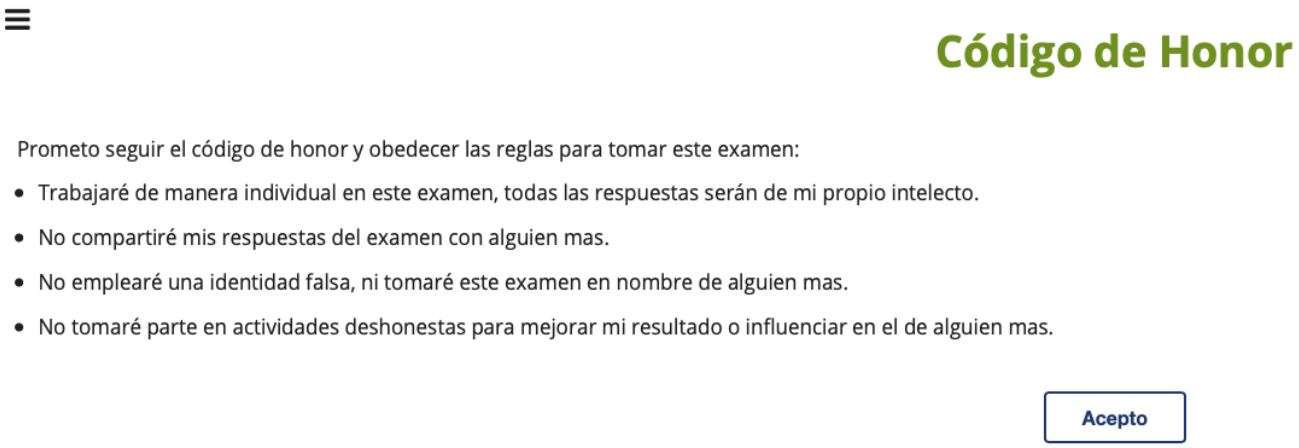
CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ACADEMIA DE INDUSTRIA DEL SOFTWARE

Nombre del Estudiante:	José Luis Sandoval Pérez	**Fecha:**	10/04/2024
Materia:	**TEORIA DE LA COMPLEJIDAD COMPUTACIONAL**	**Carrera:**	ICI
Profesores:	Dr. Miguel Angel Meza de Luna	**Semestre:**	6º
Periodo:	(X) Enero – Junio ( ) Agosto - Diciembre	**Aciertos:**	
Tipo de Examen:	Parcial: 1° ( ) 2° (X) 3° ( ) Otro:	**Calificación:**	

Instrucciones: Lee el código de honor.	



II. Instrucciones: Realiza un programa que de solución a lo indicado.	**Valor: 4 puntos**

Del problema de la moneda investiga y pon a punto la solución con las técnicas de programación: 1)Divide y Vencerás, 2)Algoritmo Voraz, 3)Programación Dinámica.

1. Divide y Venceraz

```
Número mínimo de monedas: 3
Monedas utilizadas: [1, 5, 25]
```

Nuestro primer set de entrada es:
- Cantidad a dar es 31
- Monedas disponibles [1,5,10,25]

```
Número mínimo de monedas: 4
Monedas utilizadas: [1, 10, 12, 25]
```

Segundo set de entrada:
- Cantidad a dar es 48
- Monedas disponibles [1,5,10,12,20,25]

2. Algoritmo Voraz

```
Número mínimo de monedas: 3
Monedas utilizadas: [25, 5, 1]
```

Nuestro primer set de entrada es:
- Cantidad a dar es 31
- Monedas disponibles [1,5,10,25]

```
Número mínimo de monedas: 5
Monedas utilizadas: [25, 20, 1, 1, 1]
```

Segundo set de entrada:
- Cantidad a dar es 48
- Monedas disponibles [1,5,10,12,20,25]

3. Programacion dinamica

```
Número mínimo de monedas: 3
Monedas utilizadas: [25, 5, 1]
```

Nuestro primer set de entrada es:
- Cantidad a dar es 31
- Monedas disponibles [1,5,10,25]

```
Número mínimo de monedas: 4
Monedas utilizadas: [25, 12, 10, 1]
```

Segundo set de entrada:
- Cantidad a dar es 48
- Monedas disponibles [1,5,10,12,20,25]

Programa a punto con 3 tecnicas distintas.

III. Instrucciones: De los programas anteriores responde correctamente lo indicado.	**Valor: 6 puntos**

1. **Dependiendo de las características específicas del problema y de los requisitos de rendimiento, se puede elegir una de estas técnicas de programación para resolver el problema de la moneda. ¿Cuál es el mejor?, justifica tu respuesta.**
   Sí, dependiendo de la complejidad del problema. Por ejemplo, si tenemos un problema que se puede dividir en sub-problemas independientes tenemos 2 maneras de hacerlo, divide y venceras y greedy. Con divide y venceraz nos aseguramos de encontrar la solución optima aunque el uso de memoria puede tener un coste mayor, sin embargo con un algoritmo greedy el uso de la memoria puede reducir en coste, ya que en cada sub-problema se elige la solución mas optima local para así después llegar a la solución optima global. Este tipo de algoritmo greedy tiene un problema ya que en algunos casos la solución no siempre es optima.
   Ahora la programación dinámica de igual manera nos ayuda a reducir el coste en memoria, debido a que utiliza la memoizacion para guardar resultados anteriormente calculados, es decir, si es necesario realizar un calculo que ya haya sido previamente calculado no será necesario hacer de nuevo ese cálculo, sino únicamente obtener el resultado donde haya sido guardado, optimizando memoria y espacio.
   La selección de cual es el mejor es un poco ambigua, depende del problema, tamaño del set de entrada, si es divisible en problemas más pequeño, pero basado en mi criterio y en lo aprendido en la materia diría que el más completo seria la programación dinámica.

2. **Realiza varias ejecuciones para ver el comportamiento de trabajo. Adjunta screenshot y tu análisis correspondiente.**

1. **Divide y Venceraz**

```
Número mínimo de monedas: 3
Monedas utilizadas: [1, 5, 25]
```

Nuestro primer set de entrada es:
- Cantidad a dar es 31
- Monedas disponibles [1,5,10,25]

```
Número mínimo de monedas: 4
Monedas utilizadas: [1, 10, 12, 25]
```

Segundo set de entrada:
- Cantidad a dar es 48
- Monedas disponibles [1,5,10,12,20,25]

El algoritmo divide y vencerás para el problema de la moneda resulta un poco ambiguo, ya que el problema no puede ser divisible en varios subproblemas mas pequeños, la manera en la que se puede realizar es la siguiente, por cada moneda va restando el valor de la moneda a la cantidad restante así hasta que el moneda sea mayor o igual a la cantidad, esto lo hace mediante recursión, esto lo hace para calcular el numero mínimo de monedas a utilizar por cada moneda, al final la función recursiva regresa la cantidad de monedas utilizadas para la cantidad y las va comparando con las demás monedas para encontrar el optimo. Este enfoque para este problema no es del todo claro y resulta muy complicado de entender, además que el uso de recursos y memoria es muy excesivo por tantas llamadas recursivas.

2. **Algoritmo voraz**

```
Número mínimo de monedas: 3
Monedas utilizadas: [25, 5, 1]
```

Nuestro primer set de entrada es:
- Cantidad a dar es 31
- Monedas disponibles [1,5,10,25]

```
Número mínimo de monedas: 5
Monedas utilizadas: [25, 20, 1, 1, 1]
```

Segundo set de entrada:
- Cantidad a dar es 48
- Monedas disponibles [1,5,10,12,20,25]

La forma de trabaja con este enfoque si cumple con la definicion de la tecnia, que es ir seleccionado la mejor solucion local en cada paso, para el primer set de entrada funciona perfecto, sin embargo, notamos que en el segundo set de entrada utiliza una moneda mas que en el enfoque anterior ¿Por qué sucedió esto? Sabemos que los algoritmos greedy toman la mejor solucion local y ya no pueden regresarse, analicemos la respuesta con el 2 set de entrada. Notemos que dentro del algoritmo greedy las iteraciones tambien son de moneda en moneda. Primero organizamos las monedas de mayor a menor para asi elegir siempre la mejor opcion, nuestro arreglo quedaria asi [25,20,12,10,5,1], ahora iteramos sobre la primera moneda que es 25 y preguntamos 25<=48 , sí , ahora restamos 25 a 48, teniendo como cantidad restante 23 como 25 ya no es <= 23 pasa a la siguiente moneda, ahora sigue la siguiente moneda que es la de 20, preguntamos 20<=23, sí, restamos y ahora sigue la siguiente moneda que es 12, al ser mayor que la cantidad restante (3) omitimos la moneda y pasamos a la siguiente, pasa lo mismo hasta llegar a la moneda de 1, ahora sí 1<=3 se resta esta cantidad a 3, quedandonos como resultado 2, se sigue cumpliendo la condicion hasta que ya no sea menor.Ahora como resultado nos da que se utilizaron 5 monedas, por lo que no se llego a la solucion optima que es utulizar 4 monedas, el enofque greedy presenta este problema, a pesar de ser mas eficiente en tiempo y espacio no siempre es el mejor.

3. **Programacion dinamica**

```
Número mínimo de monedas: 3
Monedas utilizadas: [25, 5, 1]
```

Nuestro primer set de entrada es:

- Cantidad a dar es 31
- Monedas disponibles [1,5,10,25]

```
Número mínimo de monedas: 4
Monedas utilizadas: [25, 12, 10, 1]
```

Segundo set de entrada:

- Cantidad a dar es 48
- Monedas disponibles [1,5,10,12,20,25]

El enfoque de programación dinámica es similar al enfoque divide y venceras para este problema, con la única diferencia de que para cada moneda se va guardando la cantidad de monedas utilizadas para para cada cantidad, para así establecer un nuevo valor mínimo de cantidad de monedas utilizadas para cada valor de las monedas, es decir se ocupan 31 de 1 para la primera moneda, 6 para la moneda de 5, 3 para la moneda de 10 y 1 para la moneda de 25, después al tener seleccionado el mínimo se vuelve a hacer el mismo proceso de memorizar cuantas monedas se necesitan para completar la cantidad sobrante con cada una de las nomenclaturas.

3. **Saca el Big O de cada técnica de programación, puedes usar la función de Python.**
    1. Divide y venceras: O(n*cantidad a regresar)
    2. Greedy: O(n log n)
    3. Programacion dinamica: O(cantidad a regresar * n)

    Notamos que es mejor el algoritmo greedy pero como se menciono no siempre se llegara a la solución optima

4. **Indica de cada técnica de programación el consumo de memoria aproximado.**

    Divide y venceras: alto
    Greedy: bajo
    Programacion dinámica alto

    Esto se ve reflejado en la complejidad del algoritmo