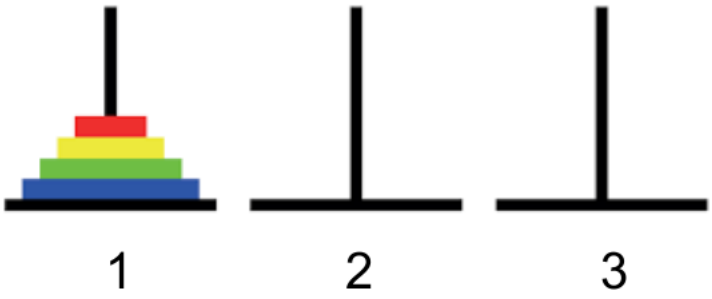
Actividad Guiada 2 de Algoritmos de Optimizacion

Nombre: Briam Sebastian Ramos Guevara

https://colab.research.google.com/drive/16u_8Qg-hIIFx_J9NNp7k2goZEequ3MV2

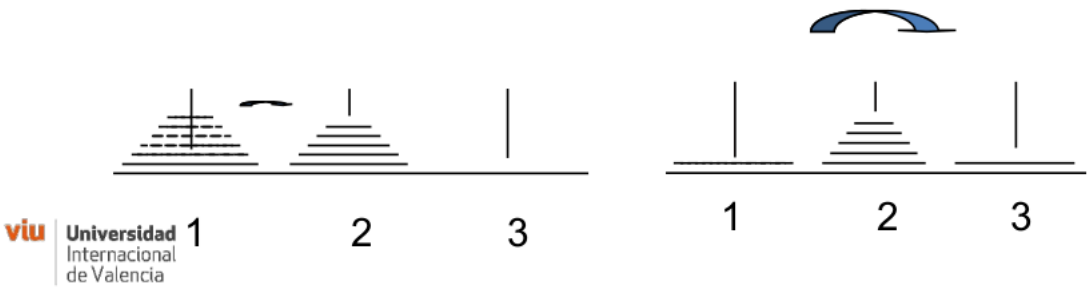
https://github.com/bramosguevara/AlgoritmosdeOptimizacion

Torres de Hanoi - Divide y venceras



Resolver(Total_fichas=4, Desde=1 , Hasta=3) es valido con:

- Resolver(Total_fichas=3, Desde=1, Hasta=2)
- Mover(Desde=1, Hasta=3)
- Resolver(Total_fichas=3, Desde=2, Hasta=3)



viu Universidad Internacional de Valencia

```
#Torres de Hanoi - Divide y venceras
##############################################################################
```

```
#############################################################
def Torres_Hanoi(N, desde, hasta):
  #N - Nº de fichas
  #desde - torre inicial
  #hasta - torre final
  if N==1 :
    print("Lleva la ficha desde " + str(desde) + " hasta " + str(hasta))

  else:
    Torres_Hanoi(N-1, desde, 6-desde-hasta)
    print("Lleva la ficha desde " + str(desde) + " hasta " + str(hasta))
```

```
    Torres_Hanoi(N-1, 6-desde-hasta,  hasta)

Torres_Hanoi(4, 1, 3)
#################################################################
```

```
Lleva la ficha desde 1 hasta 2
Lleva la ficha desde 1 hasta 3
Lleva la ficha desde 2 hasta 3
Lleva la ficha desde 1 hasta 2
Lleva la ficha desde 3 hasta 1
Lleva la ficha desde 3 hasta 2
Lleva la ficha desde 1 hasta 2
Lleva la ficha desde 1 hasta 3
Lleva la ficha desde 2 hasta 3
Lleva la ficha desde 2 hasta 1
Lleva la ficha desde 3 hasta 1
Lleva la ficha desde 2 hasta 3
Lleva la ficha desde 1 hasta 2
Lleva la ficha desde 1 hasta 3
Lleva la ficha desde 2 hasta 3
```

Cambio de monedas - Técnica voraz

```
#Cambio de monedas - Técnica voraz
#################################################################################
SISTEMA = [11, 5 ,1  ]
#################################################################
def cambio_monedas(CANTIDAD,SISTEMA):
#....
  SOLUCION = [0]*len(SISTEMA)
  ValorAcumulado = 0

  for i,valor in enumerate(SISTEMA):
    monedas = (CANTIDAD-ValorAcumulado)//valor
    SOLUCION[i] = monedas
    ValorAcumulado = ValorAcumulado + monedas*valor

    if CANTIDAD == ValorAcumulado:
      return SOLUCION

  print("No es posible encontrar solucion")
cambio_monedas(15,SISTEMA)


#################################################################
```

```
[1, 0, 4]
```

N Reinas - Vuelta Atrás(Backtracking)

```
#N Reinas - Vuelta Atrás()
#################################################################################

#Verifica que en la solución parcial no hay amenzas entre reinas
#################################################################
def es_prometedora(SOLUCION,etapa):
```

```
#################################################################
  #print(SOLUCION)
  #Si la solución tiene dos valores iguales no es valida => Dos reinas en la misma fila
  for i in range(etapa+1):
    #print("El valor " + str(SOLUCION[i]) + " está " +  str(SOLUCION.count(SOLUCION[i])) + " veces")
    if SOLUCION.count(SOLUCION[i]) > 1:
      return False

    #Verifica las diagonales
    for j in range(i+1, etapa +1 ):
      #print("Comprobando diagonal de " + str(i) + " y " + str(j))
      if abs(i-j) == abs(SOLUCION[i]-SOLUCION[j]) : return False
  return True

#Traduce la solución al tablero
#################################################################
def escribe_solucion(S):
#################################################################
  n = len(S)
  for x in range(n):
    print("")
    for i in range(n):
      if S[i] == x+1:
        print(" X " , end="")
      else:
        print(" - ", end="")



#Proceso principal de N-Reinas
#################################################################
def reinas(N, solucion=[],etapa=0):
#################################################################
### ....
  if len(solucion) == 0:         # [0,0,0...]
    solucion = [0 for i in range(N) ]

  for i in range(1, N+1):
    solucion[etapa] = i
    if es_prometedora(solucion, etapa):
      if etapa == N-1:
        print(solucion)
      else:
        reinas(N, solucion, etapa+1)
    else:
      None

  solucion[etapa] = 0

reinas(5,solucion=[],etapa=0)
```

```
[1, 3, 5, 2, 4]
[1, 4, 2, 5, 3]
[2, 4, 1, 3, 5]
[2, 5, 3, 1, 4]
[3, 1, 4, 2, 5]
[3, 5, 2, 4, 1]
[4, 1, 3, 5, 2]
[4, 2, 5, 3, 1]
[5, 2, 4, 1, 3]
[5, 3, 1, 4, 2]
```

```
escribe_solucion([1, 5, 8, 6, 3, 7, 2, 4])
```

```
 X  -  -  -  -  -  -  -
 -  -  -  -  -  -  X  -
 -  -  -  -  X  -  -  -
 -  -  -  -  -  -  -  X
 -  X  -  -  -  -  -  -
 -  -  -  X  -  -  -  -
 -  -  -  -  -  X  -  -
 -  -  X  -  -  -  -  -
```

Practica Individual

Problema: Encontrar los dos puntos más cercanos.

• Dado un conjunto de puntos se trata de encontrar los dos puntos más cercanos

• Guía para aprendizaje:

- Suponer en 1D, o sea, una lista de números: [3403, 4537, 9089, 9746, 7259, ….
- Primer intento: Fuerza bruta
- Calcular la complejidad. ¿Se puede mejorar?
- Segundo intento. Aplicar Divide y Vencerás
- Calcular la complejidad. ¿Se puede mejorar?
- Extender el algoritmo a 2D: [(1122, 6175), (135, 4076), (7296, 2741)…
- Extender el algoritmo a 3D.

Implementando el algoritmo por resolución de Fuerza Bruta:

```
import random

def lista_1D(longitud, rango_min, rango_max):

    return [random.randint(rango_min, rango_max) for _ in range(longitud)]

def encontrar_puntos_mas_cercanos(puntos):
    # Inicializamos los valores mínimos con un valor alto.
    min_distancia = float('inf')
    punto1 = None
    punto2 = None

    # Recorremos cada par posible de puntos para calcular sus distancias.
    for i in range(len(puntos)):
        for j in range(i + 1, len(puntos)):
            distancia = abs(puntos[i] - puntos[j])
            if distancia < min_distancia:
                min_distancia = distancia
                punto1 = puntos[i]
                punto2 = puntos[j]

    return punto1, punto2, min_distancia

# Generamos una lista de puntos aleatorios.
puntos = lista_1D(longitud=10, rango_min=1, rango_max=10000)
print("Lista de puntos generada:", puntos)

# Llamamos a la función y mostramos el resultado.
p1, p2, distancia = encontrar_puntos_mas_cercanos(puntos)
print(f"Los dos puntos más cercanos son {p1} y {p2} con una distancia de {distancia}.")
```

```
Lista de puntos generada: [3737, 4120, 8399, 6781, 7272, 5297, 3405, 5081, 1414, 1558]
Los dos puntos más cercanos son 1414 y 1558 con una distancia de 144.
```

La complejidad del algoritmo anterior es $O(n^2)$

En este caso, el algoritmo anterior se puede mejorar dando un enfoque basado en el ordenamiento, donde se puede pasar la complejidad del algoritmo a $O(nlog(n))$ en el cual la idea es recorrer la lista ordenada y comparar cada punto con su vecino, por lo que los puntos más cercanos estarán adyacentes después de ordenar. Esto optimiza el algoritmo de modo que se baja la complejidad y permite reconocer los puntos más cercanos de forma más eficiente.

Implementando la técnica de Divide y Venceras para la comparación de eficiencia con la resolución por Fuerza Bruta:

```
import random

def lista_1D(longitud, rango_min, rango_max):

    return [random.randint(rango_min, rango_max) for _ in range(longitud)]

def divide_y_venceras(puntos):
    def distancia_minima_recursiva(puntos_ordenados):
        # Dividimos la lista en dos mitades
        mitad = len(puntos_ordenados) // 2
        izquierda = puntos_ordenados[:mitad]
        derecha = puntos_ordenados[mitad:]
```

```
        # Resolver recursivamente en cada mitad
        p1_izq, p2_izq, dist_izq = distancia_minima_recursiva(izquierda) if len(izquierda) > 1 else (izquierda[0], izquierda[0], float('inf'
        p1_der, p2_der, dist_der = distancia_minima_recursiva(derecha) if len(derecha) > 1 else (derecha[0], derecha[0], float('inf'))

        # Obtener la distancia mínima entre las dos mitades
        if dist_izq < dist_der:
            min_p1, min_p2, min_distancia = p1_izq, p2_izq, dist_izq
        else:
            min_p1, min_p2, min_distancia = p1_der, p2_der, dist_der

        # Revisar puntos cercanos al límite entre las dos mitades
        centro = puntos_ordenados[mitad - 1]  # Último de la izquierda
        en_banda = [p for p in puntos_ordenados if abs(p - centro) < min_distancia]

        # Comparar los puntos dentro de la banda
        for i in range(len(en_banda)):
            for j in range(i + 1, len(en_banda)):
                distancia = abs(en_banda[i] - en_banda[j])
                if distancia < min_distancia:
                    min_distancia = distancia
                    min_p1, min_p2 = en_banda[i], en_banda[j]

        return min_p1, min_p2, min_distancia

    # Ordenamos los puntos inicialmente para el algoritmo
    puntos_ordenados = sorted(puntos)
    return distancia_minima_recursiva(puntos_ordenados)

# Generamos una lista de puntos aleatorios.
puntos = lista_1D(longitud=10, rango_min=1, rango_max=10000)
print("Lista de puntos generada:", puntos)

# Llamamos a la función de Divide y Vencerás y mostramos el resultado.
p1, p2, distancia = divide_y_venceras(puntos)
print(f"Los dos puntos más cercanos son {p1} y {p2} con una distancia de {distancia}.")
```

```
Lista de puntos generada: [4891, 3481, 2360, 5949, 2327, 6116, 3570, 7908, 7226, 1242]
Los dos puntos más cercanos son 2327 y 2360 con una distancia de 33.
```

Para el algoritmo anterior se tiene que la complejidad es $O(nlog(n))$ por lo que se concluye que ya es óptimo en términos de complejidad.
En este ejercicio directamente la técnica Divide y Vencerás es más óptima para encontrar los puntos más cercanos en una lista 1D comparada con la resolución por Fuerza Bruta.

Implementando la técnica de Divide y Venceras pero incluyendo una lista aleatoria 2D:

```
import random

def lista_2D(longitud, rango_min, rango_max):

    return [(random.randint(rango_min, rango_max), random.randint(rango_min, rango_max)) for _ in range(longitud)]

def divide_y_venceras(puntos):
    def distancia(p1, p2):
        return ((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)**0.5

    def distancia_minima_recursiva(puntos_ordenados):
        # Dividimos la lista en dos mitades
        mitad = len(puntos_ordenados) // 2
        izquierda = puntos_ordenados[:mitad]
        derecha = puntos_ordenados[mitad:]

        # Resolver recursivamente en cada mitad
        p1_izq, p2_izq, dist_izq = distancia_minima_recursiva(izquierda) if len(izquierda) > 1 else (izquierda[0], izquierda[0], float('inf'
        p1_der, p2_der, dist_der = distancia_minima_recursiva(derecha) if len(derecha) > 1 else (derecha[0], derecha[0], float('inf'))

        # Obtener la distancia mínima entre las dos mitades
        if dist_izq < dist_der:
            min_p1, min_p2, min_distancia = p1_izq, p2_izq, dist_izq
        else:
            min_p1, min_p2, min_distancia = p1_der, p2_der, dist_der

        # Revisar puntos cercanos al límite entre las dos mitades
```

```
        linea_division_x = puntos_ordenados[mitad][0]
        en_banda = [p for p in puntos_ordenados if abs(p[0] - linea_division_x) < min_distancia]
        en_banda.sort(key=lambda punto: punto[1])

        # Comparar los puntos dentro de la banda
        for i in range(len(en_banda)):
            for j in range(i + 1, min(i + 7, len(en_banda))):
                distancia_actual = distancia(en_banda[i], en_banda[j])
                if distancia_actual < min_distancia:
                    min_distancia = distancia_actual
                    min_p1, min_p2 = en_banda[i], en_banda[j]

        return min_p1, min_p2, min_distancia

    # Ordenamos los puntos inicialmente por coordenada X para el algoritmo
    puntos_ordenados = sorted(puntos, key=lambda punto: punto[0])
    return distancia_minima_recursiva(puntos_ordenados)

# Generamos una lista de puntos aleatorios en 2D.
puntos = lista_2D(longitud=10, rango_min=1, rango_max=10000)
print("Lista de puntos generada:", puntos)

# Llamamos a la función de Divide y Vencerás y mostramos el resultado.
p1, p2, distancia = divide_y_venceras(puntos)
print(f"Los dos puntos más cercanos son {p1} y {p2} con una distancia de {distancia}.")
```

```
Lista de puntos generada: [(7285, 1691), (2251, 4752), (2915, 5812), (5064, 1192), (5984, 2813), (6464, 2597), (3767, 2657), (883, 2800)
Los dos puntos más cercanos son (6464, 2597) y (5984, 2813) con una distancia de 526.3610927870714.
```

Implementando la técnica de Divide y Venceras pero incluyendo una lista aleatoria 3D:

```
import random

def lista_3D(longitud, rango_min, rango_max):

    return [(random.randint(rango_min, rango_max), random.randint(rango_min, rango_max), random.randint(rango_min, rango_max)) for _ in rang

def divide_y_venceras(puntos):
    def distancia(p1, p2):
        return ((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2 + (p1[2] - p2[2])**2)**0.5

    def distancia_minima_recursiva(puntos_ordenados):
        # Dividimos la lista en dos mitades
        mitad = len(puntos_ordenados) // 2
        izquierda = puntos_ordenados[:mitad]
        derecha = puntos_ordenados[mitad:]

        # Resolver recursivamente en cada mitad
        p1_izq, p2_izq, dist_izq = distancia_minima_recursiva(izquierda) if len(izquierda) > 1 else (izquierda[0], izquierda[0], float('inf'
        p1_der, p2_der, dist_der = distancia_minima_recursiva(derecha) if len(derecha) > 1 else (derecha[0], derecha[0], float('inf'))

        # Obtener la distancia mínima entre las dos mitades
        if dist_izq < dist_der:
            min_p1, min_p2, min_distancia = p1_izq, p2_izq, dist_izq
        else:
            min_p1, min_p2, min_distancia = p1_der, p2_der, dist_der

        # Revisar puntos cercanos al límite entre las dos mitades
        linea_division_x = puntos_ordenados[mitad][0]
        en_banda = [p for p in puntos_ordenados if abs(p[0] - linea_division_x) < min_distancia]
        en_banda.sort(key=lambda punto: punto[1])  # Ordenamos por coordenada Y

        # Comparar los puntos dentro de la banda
        for i in range(len(en_banda)):
            for j in range(i + 1, min(i + 7, len(en_banda))):
                distancia_actual = distancia(en_banda[i], en_banda[j])
                if distancia_actual < min_distancia:
                    min_distancia = distancia_actual
                    min_p1, min_p2 = en_banda[i], en_banda[j]

        return min_p1, min_p2, min_distancia

    # Ordenamos los puntos inicialmente por coordenada X para el algoritmo
    puntos_ordenados = sorted(puntos, key=lambda punto: punto[0])
    return distancia_minima_recursiva(puntos_ordenados)
```

```
# Generamos una lista de puntos aleatorios en 3D.
puntos = lista_3D(longitud=10, rango_min=1, rango_max=10000)
print("Lista de puntos generada:", puntos)

# Llamamos a la función de Divide y Vencerás y mostramos el resultado.
p1, p2, distancia = divide_y_venceras(puntos)
print(f"Los dos puntos más cercanos son {p1} y {p2} con una distancia de {distancia}.")
```

```
Lista de puntos generada: [(762, 1032, 9395), (8993, 5383, 3357), (4648, 5569, 5985), (5354, 8615, 1115), (1711, 1083, 7099), (1654, 731
Los dos puntos más cercanos son (2593, 287, 5997) y (1711, 1083, 7099) con una distancia de 1620.4764731399218.
```