Open in Colab

Actividad Guiada 1 de Algoritmos de Optimización

- **Nombre y apellidos**: Gerardo Mauricio Gutierrez Quintana
- **URL Colab:** https://colab.research.google.com/drive/12W5nzlHjdMdAhTMMuzf6w7YwT5Eo2sm2?usp=sharing
- **URL Github:** https://github.com/MauGutierrez/03MAIR---Algoritmos-de-Optimizacion---2021/tree/main/AG1

Torres de Hanoi - Divide y Venceras

In [1]:
```
# Resolvemos recursivamente el problema
def torres_hanoi(n, torre_inicio, torre_aux, torre_final):
    # caso base cuando solo tenemos que mover una pieza
    if n == 1:
        print(f'Mover {torre_inicio} a {torre_final}')
    else:
        # Queremos mover a la torre auxiliar
        torres_hanoi(n-1, torre_inicio, torre_final, torre_aux)
        print(f'Mover {torre_inicio} a {torre_final}')
        # Queremos mover desde la torre auxiliar
        torres_hanoi(n-1, torre_aux, torre_inicio, torre_final)

torres_hanoi(3, 1, 2, 3)
```

```
Mover 1 a 3
Mover 1 a 2
Mover 3 a 2
Mover 1 a 3
Mover 2 a 1
Mover 2 a 3
Mover 1 a 3
```

Cambio de monedas - Técnica Voraz

In [2]:
```
def cambio_monedas(cantidad, sistema_monetario):
    n = len(sistema_monetario)
```

```
    n = len(sistema_monetario)
    solucion = [0]*n
    valor_acumulado = 0
    # Calculamos la cantidad de monedas necesarias por moneda del sistema monetario
    for i in range(n):
        monedas = int((cantidad-valor_acumulado)/sistema_monetario[i])
        solucion[i] = monedas
        valor_acumulado += monedas * sistema_monetario[i]
        # si la cantidad acumulada es igual a la cantidad terminamos el problema
        if valor_acumulado == cantidad:
            return solucion

    return solucion

print(cambio_monedas(48, [25, 10, 5, 1]))
```

```
[1, 2, 0, 3]
```

N Reinas - Vuelta Atrás

In [3]:

```
# funcion helper para detectar si hay un elemento
# en la misma fila o amenaza entre diagonales
def solucion_prometedora(solucion, etapa):
    n = etapa
    for i in range(n+1):
        # Checar amenaza entre renglones
        if solucion.count(solucion[i]) > 1:
            return False
        # Checar amenaza entre diagonales
        for j in range(i+1, n+1):
            if abs(i-j) == abs(solucion[i]- solucion[j]):
                return False
    # Si terminamos de iterar el array solucion y quiere decir que no hay amenaza
    return True

def reinas(n, solucion=[], etapa=0):
    # Inicializamos el array solucion
    if len(solucion) == 0:
        solucion = [0 for i in range(n)]
```

```
    # Iteramos a través del tablero
    for i in range(1, n+1):
        # Colocamos una reina en el primer lugar disponible de la fila
        solucion[etapa] = i
        # Si no hay amenaza
        if solucion_prometedora(solucion, etapa):
            # si la etapa es igual a la dimension del tablero, hemos terminado
            if etapa == n-1:
                print(solucion)
            # si no, todavía tenemos que seguir iterando con las siguientes reinas
            else:
                reinas(n, solucion, etapa+1)
        else:
            None
    # No hay solucion para cierta reina
    solucion[etapa] = 0

reinas(4)
```

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

Practica Individual

Encontrar los dos puntos más cercanos por Fuerza Bruta

In [4]:
```
import random
import time


def fuerza_bruta(lista):
  minimia_distancia = 99999
  puntos = []

  for i in range(len(lista)):
    distancia_parcial = 0
    for j in range(len(lista)):
```

```
      if (i != j) and (lista[i] != lista[j]):
        distancia_parcial = abs(lista[i] - lista[j])
        if distancia_parcial < minimia_distancia:
          minimia_distancia = distancia_parcial
          puntos = [lista[i], lista[j]]

  return puntos, minimia_distancia

lista_1d = [random.randrange(1, 10000) for x in range(2000)]
start = time.time()
dos_puntos, distancia = fuerza_bruta(lista_1d)
stop = time.time()

print(f'La distancia minima entre los puntos {dos_puntos} es : {distancia}')
print(f'Tiempo de ejecución algoritmo por fuerza bruta: {stop-start}')
```

```
La distancia minima entre los puntos [7222, 7223] es : 1
Tiempo de ejecución algoritmo por fuerza bruta: 1.3008148670196533
```

Cálculo de la complejidad Algoritmo Fuerza Bruta

Despreciando la complejidad del ciclo donde inicializamos la lista, podemos decir que el algoritmo por fuerza bruta tiene una complejidad de $O(n^2)$ donde n será la longitud de nuestra lista.

Decimos que tiene una complejidad de orden cuadrático ya que para cada elemento del array iteramos n veces para encontrar las distancias menores. Por lo tanto, para los n elementos de la lista, iteramos n veces.

Intuitivamente podríamos mejorar el algoritmo cuando se encuentren dos puntos con una distancia igual a 1, ya que es la menor distnacia posible que puede haber entre un par de puntos diferentes.

Con la propuesta antes mencionada, en el mejor de los casos podríamos tener una complejidad de $O(1)$ suponiendo que para el primer par de puntos que comparemos la distancia sea igual a 1, por lo tanto, solo habríamos recorrido la lista una vez.

En el peor de los casos la complejidad seguira siendo $O(n^2)$ suponiendo que los puntos con menor distancia sean el penultimo y ultimo de la lista. De ser así habría que iterar todos los elementos dos veces hasta encontrar la menor distancia.

Encontrar los dos puntos más cercanos por Divide y Vencerás

Lista 1D

In [10]:
```
import random
import math
import time

def merge_sort(array):
  if len(array) > 1:
    mid = len(array) // 2
    izquierda = array[:mid]
    derecha = array[mid:]

    merge_sort(izquierda)
    merge_sort(derecha)

    i = j = k = 0

    while i < len(izquierda) and j < len(derecha):
      if izquierda[i] < derecha[j]:
        array[k] = izquierda[i]
        i += 1
      else:
        array[k] = derecha[j]
        j += 1
      k += 1

    while i < len(izquierda):
      array[k] = izquierda[i]
      i += 1
      k += 1

    while j < len(derecha):
      array[k] = derecha[j]
      j += 1
      k += 1

def minima_distancia(array):
  minima_distancia = 9999
  puntos = []
  for i in range(0, len(array)-1):
```

```
    if array[i] != array[i+1]:
      distancia_parcial = abs(array[i] - array[i+1])
      if distancia_parcial < minima_distancia:
        puntos = [array[i], array[i+1]]
        minima_distancia = distancia_parcial

  return puntos, minima_distancia

lista_1d = [random.randrange(1, 10000) for x in range(2000)]

start = time.time()
merge_sort(lista_1d)
dos_puntos, distancia = minima_distancia(lista_1d)
stop = time.time()


print(f'La distancia minima entre los puntos {dos_puntos} es : {distancia}')
print(f'Tiempo de ejecución algoritmo por fuerza bruta: {stop-start}')
```

```
La distancia minima entre los puntos [22, 23] es : 1
Tiempo de ejecución algoritmo por fuerza bruta: 0.011330842971801758
```

Cálculo de la complejidad Algoritmo Divide y Vencerás

Después de ejecutar los algortimos de fuerza bruta y Divide y Vencerás en una lista con datos aleatorios del mismo tamaño, podemos observar que el algoritmo de Divide y Vencerás ofrece un mejor tiempo de ejecución.

Comparando ambos resultados, podemos observar que Divide y Vencerás tuvo un tiempo de ejecución aproximadamente 100 veces más rápido.

Podemos decir que el Algoritmo de Divide y Vencerás tiene una complejidad en todos los casos (peor, promedio y mejor) de por lo siguiente:

- El algoritmo siempre divide en mitades la cada lista que es recibida, por lo tanto hiremos reduciendo el problema original con una complejidad de $O(Log \cdot n)$ donde n es el tamaño de la lista inicial.
- Una vez dividida la lista, hacemos n operaciones para realizar el merge y de ambas listas en la lista original. Por lo tanto, para el Merge tenemos una complejidad de $O(n)$
- Al combinar ambas complejidades, obtenemos una complejidad total de

- Al combinar ambas complejidades, obtenemos una complejidad total de

Aunque el algoritmo de Merge sort es muy eficiente en tiempo de ejecución, tiene la desventaja de que requiere de espacio adicional para realizar el ordenamiento. Si en nuestro caso quisieramos optimizar la complejidad en el espacio utilizado, podríamos utilizar otra alternativa llamada Quick Sort, sin embargo, a diferencia del merge sort donde en los 3 casos tiene la misma complejidad, en quick sort en el peor de los casos tenemos una complejidad de $O(n^2)$

Encontrar los dos puntos más cercanos por Divide y Vencerás Lista 2D

In [15]:
```
import random
import math
import time

def distancia_euclidiana(punto_1, punto_2):
  temp = 0
  for i in range(len(punto_1)):
    temp += (punto_1[i] - punto_2[i])**2

  return math.sqrt(temp)


def minima_distancia(array):
  minima_distancia = 9999
  puntos = []
  for i in range(0, len(array)-1):
    distancia_parcial = distancia_euclidiana(array[i], array[i+1])
    if distancia_parcial < minima_distancia:
      puntos = [array[i], array[i+1]]
      minima_distancia = distancia_parcial

  return puntos, minima_distancia

lista_2d = [[random.randrange(1, 10000), random.randrange(1, 10000)] for x in range(2000)]

start = time.time()
merge_sort(lista_2d)
dos_puntos, distancia = minima_distancia(lista_2d)
```

```
stop = time.time()


print(f'La distancia minima entre los puntos {dos_puntos} es : {distancia}')
print(f'Tiempo de ejecución algoritmo por fuerza bruta: {stop-start}')
```

```
La distancia minima entre los puntos [[8406, 5918], [8408, 5927]] es : 9.219544457292887
Tiempo de ejecución algoritmo por fuerza bruta: 0.013436555862426758
```

Encontrar los dos puntos más cercanos por Divide y Vencerás Lista 3D

In [17]:

```
import random
import math
import time

def minima_distancia(array):
  minima_distancia = 9999
  puntos = []
  for i in range(0, len(array)-1):
    distancia_parcial = distancia_euclidiana(array[i], array[i+1])
    if distancia_parcial < minima_distancia:
      puntos = [array[i], array[i+1]]
      minima_distancia = distancia_parcial

  return puntos, minima_distancia

lista_2d = [[random.randrange(1, 10000), random.randrange(1, 10000), random.randrange(1, 10000)]
for x in range(2000)]

start = time.time()
merge_sort(lista_2d)
dos_puntos, distancia = minima_distancia(lista_2d)
stop = time.time()


print(f'La distancia minima entre los puntos {dos_puntos} es : {distancia}')
print(f'Tiempo de ejecución algoritmo por fuerza bruta: {stop-start}')
```

```
La distancia minima entre los puntos [[6390, 5975, 9576], [6390, 6051, 9713]] es : 156.66843970627
906
Tiempo de ejecución algoritmo por fuerza bruta: 0.014549970626831055
```