# Actividad Guiada 1

## Javier Rodríguez Juárez

[Link a Github]

## Torres de Hanoi

```
def torres_hanoi(n, origen, destino, pivote):
  if n == 1:
    print(f"Mover bloque {n} desde {origen} a {destino}")
    return
  # El orden de los postes: origen, destino, pivote; ´
  # determina las reglas del juego (colocar piezas menores sobre piezas mayores)
  torres_hanoi(n - 1, origen, pivote, destino)
  print(f"Mover bloque {n} desde {origen} a {destino}")
  torres_hanoi(n - 1, pivote, destino, origen)
```

```
torres_hanoi(4, "P1", "P3", "P2")
```

```
Mover bloque 1 desde P1 a P2
Mover bloque 2 desde P1 a P3
Mover bloque 1 desde P2 a P3
Mover bloque 3 desde P1 a P2
Mover bloque 1 desde P3 a P1
Mover bloque 2 desde P3 a P2
Mover bloque 1 desde P1 a P2
Mover bloque 4 desde P1 a P3
Mover bloque 1 desde P2 a P3
Mover bloque 2 desde P2 a P1
Mover bloque 1 desde P3 a P1
Mover bloque 3 desde P2 a P3
Mover bloque 1 desde P1 a P2
Mover bloque 2 desde P1 a P3
Mover bloque 1 desde P2 a P3
```

## Cambio de monedas

```
def cambio_monedas(cantidad, sistema):
  print(f"El sistema es {sistema}")
  solucion = [0 for i in range(len(sistema))]   # Tantos ceros como elementos tiene el sistema
  valor_acumulado = 0

  for i in range(len(sistema)):
    monedas = int((cantidad - valor_acumulado) / sistema[i])
    solucion[i] = monedas
    valor_acumulado += monedas * sistema[i]
    if valor_acumulado == cantidad:
      break
  return solucion
```

```
monedas = [25, 10, 5, 1]

cambio_monedas(117, monedas)
```

```
El sistema es [25, 10, 5, 1]
[4, 1, 1, 2]
```

```
def cambio_monedas(cantidad, sistema):
  print(f"El sistema es {sistema}")
  cambio = []
  for m in sistema:
    cambio.append(cantidad // m)
    cantidad = cantidad % m
  return cambio
```

```
monedas = [25, 10, 5, 1]

cambio_monedas(117, monedas)
```

```
El sistema es [25, 10, 5, 1]
[4, 1, 1, 2]
```

## Distancia entre dos puntos

Por **fuerza bruta**, calculando la distancia entre todos los puntos.

La distancia se inicializa con infinito.

Comprobamos la distancia entre todos los puntos, tomando el mínimo de todos ellos

```
def distancia_1d_bf(lista_ptos):
  resultado = float('inf')
  for i in range(len(lista_ptos)):
    for j in range(i + 1, len(lista_ptos)):
      dist = abs(lista_ptos[i] - lista_ptos[j])
      resultado = dist if dist < resultado else resultado
  return resultado
```

```
import numpy as np
np.random.seed(42)
lista_1d = list(set(np.random.randint(low=0, high=2*10**6, size=250)))

lista_1d.sort()

print(f"Listado de puntos = {lista_1d}")
```

```
Listado de puntos = [12666, 13986, 21959, 23247, 24538, 25939, 41090, 43585, 48984, 64044, 65725, 68148, 75766, 84654, 103355, 1065
```

```
%%timeit
distancia_1d_bf(lista_1d)
```

```
11.8 ms ± 6.27 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
print(f"Distancia mínima = {distancia_1d_bf(lista_1d)}")
```

```
Distancia mínima = 78
```

Por cada iteración se realizan 3 operaciones:

- Cálculo de la distancia entre dos puntos
- Comparación entre la distancia calculada y la mínima hasta el momento
- En el peor de los casos, asignación a la variable de la distancia mínima calculada

La complejidad sería igual a:

$\sum_{i=1}^{n}\sum_{j=i+1}^{n} 3 = \sum_{i=1}^{n} 3 \times (n-j) = \sum_{i=1}^{n} 3 \times (n-i-1) = \frac{3}{2}n(n-3) \equiv \mathcal{O}(n^2)$

Por **recursividad**:

```
def distancia_1d_rec(lista_ptos):
  # Casos base:
  if len(lista_ptos) == 1:
    return float('inf')

  elif len(lista_ptos) == 2:
    return abs(lista_ptos[1] - lista_ptos[0])

  else:
    # Dividimos en dos mitades
    mid = len(lista_ptos) // 2
    mitad_izqda = lista_ptos[:mid]
    mitad_dcha = lista_ptos[mid:]

    # LLamadas recursivas
    dist_izqda = distancia_1d_rec(mitad_izqda)
    dist_dcha = distancia_1d_rec(mitad_dcha)
    # Distancia entre conjuntos contiguos
    # con los puntos más próximos a cada lado
    dist_frontera = abs(mitad_izqda[-1] - mitad_dcha[0])
    return min(dist_izqda, dist_dcha, dist_frontera)
```

```
import numpy as np
np.random.seed(42)
lista_1d = list(set(np.random.randint(low=0, high=2*10**6, size=250)))
```

```
lista_1d.sort()

print(f"Listado de puntos = {lista_1d}")
```

```
Listado de puntos = [12666, 13986, 21959, 23247, 24538, 25939, 41090, 43585, 48984, 64044, 65725, 68148, 75766, 84654, 103355, 1065
```

```
%%timeit
distancia_1d_rec(lista_1d)
```

```
177 µs ± 36.4 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

```
print(f"Distancia mínima = {distancia_1d_rec(lista_1d)}")
```

```
Distancia mínima = 78
```

Para el mejor de los casos, en los que haya únicamente uno o dos puntos:

$T^m(n) = 1 \equiv \mathcal{O}(1)$

Para el peor de los casos, por cada iteración se realizan 5 operaciones:

- Cálculo del tamaño de cada partición
- División de la lista en la parte izquierda
- División de la lista en la parte izquierda

Después se realiza la llamada recursiva a la función con cada mitad de la lista Por último, habiendo llegado al caso base:

- Se realiza el cáclulo de la ditancia entre los puntos a ambos lados de cada partición
- Cálculo de la mínima distancia

La complejidad, en cada iteración, sería igual a:

$T^p(n) = 5 + 2 \times T(\frac{n}{2}) = 5 + 2 \times (5 + 2 \times T(\frac{n}{4})) = 5 + 2 \times (5 + 2 \times (5 + 2 \times T(\frac{n}{8}))) = \ldots$
$T^p(n) = 5 + 10 + 15 + \ldots + 5 \times 2^{i-1} + 2^i \times T(\frac{n}{2^i}) = 5 \times (2^i - 1) + 2^i \times T(\frac{n}{2^i})$
$T^p(n) = 5 \times (2^{log_2 n} - 1) + 2^{log_2 n} \times T(1) = 5 \times n - 5 + n = 6n - 5 \equiv \mathcal{O}(n)$