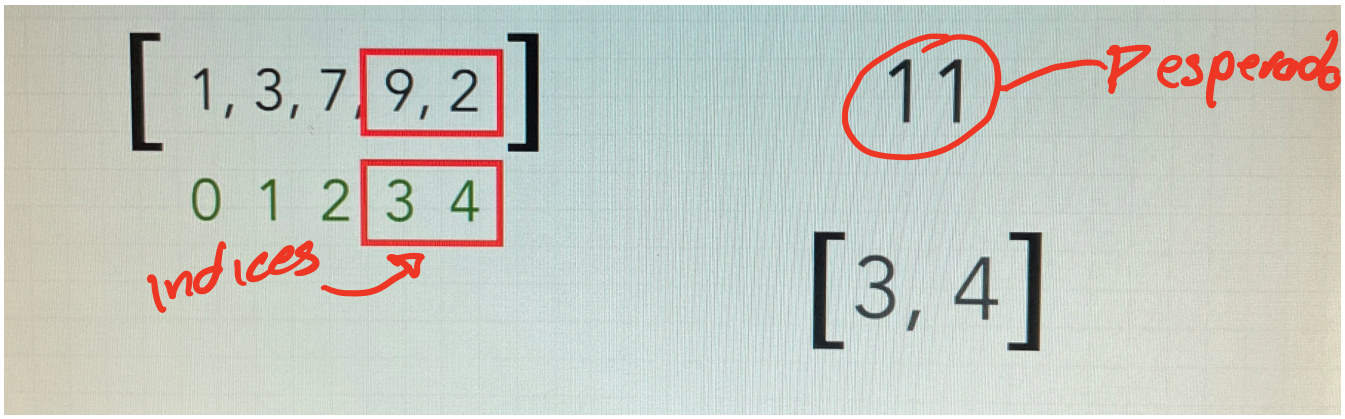
Arreglos

Encontrar los indices de dos números,

Dado un arreglo de enteros regresar los dos indices de los números que sumados de un elemento esperado.



nums = [4, 3, 7, 2] target = 6

Pseudocódigo

Sin optimizar

```
func get_indexes(nums:[int], target:int)-> [int]:
 for p1 in range(len(nums)):
    num_searched = target - nums[p1]
    for p2 in range(p1+1, len(nums)):
        if num_searched == nums[p2]:
            return [p1, p2]

 return Null
```

Optimizado

```
func get_indexes(nums:[int],target:int)->[int]:
    nums_map = {}
    for i,n1 in enumerate(nums):
        if n2 in nums_map:
            return [nums_map[n1],i]
        nom_searched = target-n1
        nums_map[num_searched]=i
    return Null
```

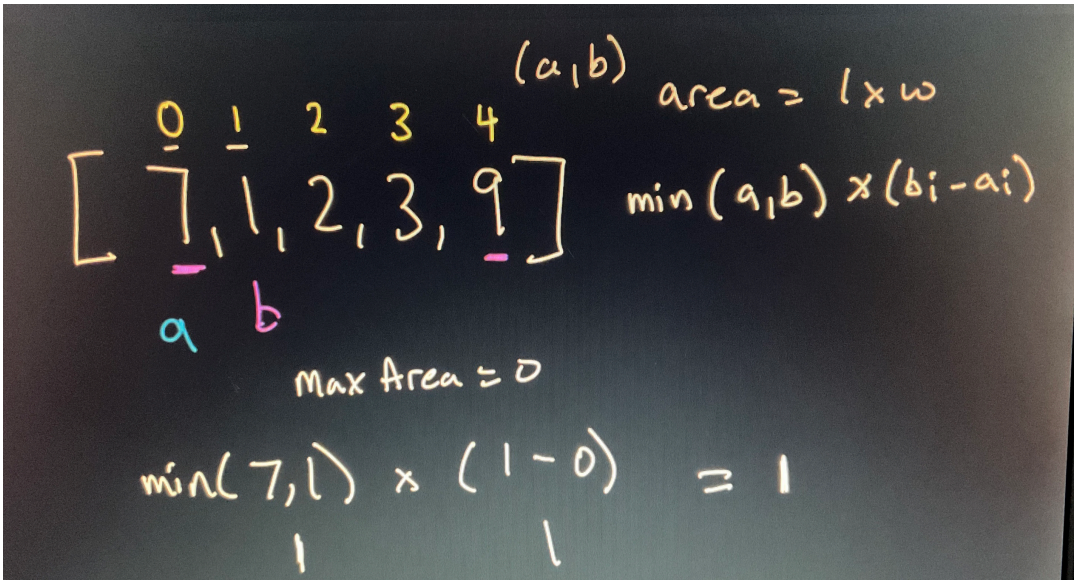
nums = [2, 8, 9, 1] target = 11

Encontrar la mayor area que se puede obtener

Dado un arreglo con enteros positivos encontra el area mayor que se puede generar entre los elementos

Consideraciones:

* Los bordes izquierdos y derechos tienen que ser mayores a las del centro para que se pueda usar



Pseudocódigo

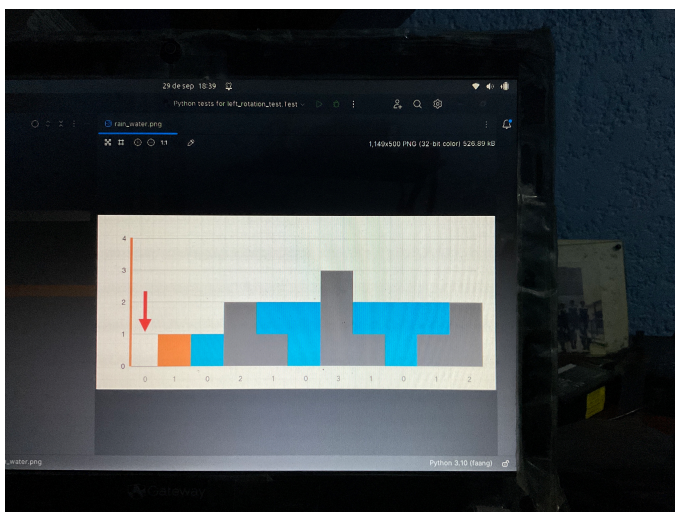
```
Func get_max_area(container:[int]) -> int:
    max_area = 0
    for n1 in range(len(container)):
        for n2 in range(n1+1, len(container)):
            height = min(container[n1], container[n2])
            width = p2 - p1
            area = height * width
            max_area = max(max_area, area)

    return max_area
```

0 1 2 3 4 5

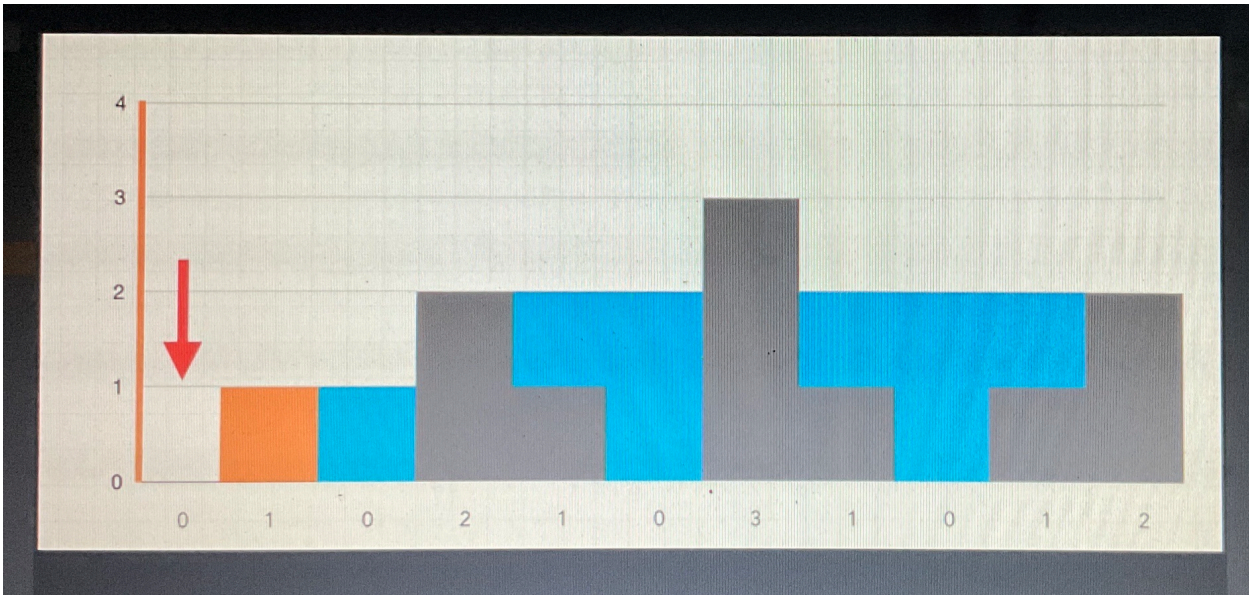
Optima [4, 8, 1, 2, 3, 9]

```
func get_max_area(container:[int]) -> int:
    max_area = 0
    a = 0
    b = len(container)-1
    while a<b:
        height = min(container[a], container[b])
        width = b - a
        area = height * width
        max_area = max(max_area, area)
        if container[a] <= container[b]:
            a += 1
        else:
            b -= 1
    return max_area
```



Contenedor de agua

Se necesito encontrar el area total que se puede almacenar en un arreglo.



Pseudocódigo

```
func get_trapped_rain_water(heights:[int])->int:
  total_water=0
  for i in range(len(heights):
    left_p=i
    right_p=i
    max_left=0
    max_right=0
    while left_p >=0:
      max_left=max(max_left,heights[left_p])
      left_p-=1
    while right_p<len(heights):
      max_right =max(max_right,heights[right_p])
      right_p+=1
    current_water=min(max_left,max_right)-heights[pi]
```

```
    if current_water >= 0:
        total_water += current_water
return total_water.
```

Optima

```
func get_trapped_water(heights: [int]) -> int:
    total_water = 0
    left = 0
    left_max = 0
    right_max = 0
    right = len(heights) - 1
    while left < right:
        if heights[left] <= heights[right]:
            if heights[left] >= left_max:
                left_max = heights[left]
            else:
                total_water += left_max - heights[left]
            left += 1
        else:
            if heights[right] >= right_max:
                right_max = heights[right]
            else:
                total_water += right_max - heights[right]
            right -= 1
    return total_water
```