Considere la siguiente información que contextualiza las preguntas 1 y 2:

I. Métodos de ordenamiento estudiados en el curso:

Función:	`default_sort_criteria(element_1, element_2)`
Propósito:	Función de comparación por defecto. Se pasa como parámetro `sort_crit` a los métodos de ordenamiento.
Entradas:	`element_1: any`
	`element_2: any`
Retorno:	En la implementación por defecto: `True` si `element_1` va antes que `element_2` (menor que). `False` en caso contrario.

Para efectos de este examen, todos los métodos de ordenamiento vistos en el curso:

- `selection_sort(my_list, sort_crit)`: Implementa Selection Sort iterativo.
- `insertion_sort(my_list, sort_crit)`: Implementa Insertion Sort iterativo.
- `shell_sort(my_list, sort_crit)`: Implementa Shell Sort iterativo.
- `merge_sort(my_list, sort_crit)`: Implementa Merge Sort recursivo.
- `quick_sort(my_list, sort_crit)`: Implementa Quick Sort recursivo.

Reciben las mismas entradas:

- `my_list`: Arreglo a ordenar del tipo: `array_list`.
- `sort_crit`: Criterio de comparación (función: `default_sort_criteria()`).

Y producen la misma salida:

- `array_list`: Arreglo ordenado ascendentemente del tipo: `array_list`.

II. Sobre `array_list`:

Un `array_list` vacío tiene la siguiente estructura y es producido por la función [illegible]

```
{ "size": 0, "elements": [] }
```

Algunas funciones útiles de `array_list`:

```
def size(my_list):
    """
    Retorna el tamaño de la lista.
    """
    return my_list["size"]

def get_element(my_list, pos):
    """
    Retorna el elemento en la posicion dada.
    """
    return my_list["elements"][pos]
```

Problema: Par con suma más cercana al objetivo.

Se tiene una lista de números enteros. Los números pueden ser positivos o negativos y no hay repetidos. Además, se recibe un número entero que representa el objetivo. El propósito es encontrar un par de números de la lista cuya suma sea la más cercana posible al objetivo dado. Dada la lista y el número objetivo como entradas a la función, hay dos posibles retornos esperados:

- `None`, cuando el arreglo está vacío o contiene un solo elemento.
- Una tupla `(num1, num2)`, cuando el arreglo tiene dos o más elementos. La tupla contiene el par de números cuya suma se aproxima más al objetivo. En el caso de que existan múltiples pares cuya suma sea igualmente más cercana al objetivo, se retorna cualquiera de ellos. El orden de aparición de `num1` y `num2` en la tupla es irrelevante.

- **Ejemplo 1** – Arreglo vacío:

Entrada: `arr = [], objetivo = 10`
Salida esperada: `None`

- **Ejemplo 2** – Arreglo de un elemento:

Entrada: `arr = [1], objetivo = 3`
Salida esperada: `None`

- ***Ejemplo 3*** – *Arreglo con un solo par:*

Entrada: `arr = [-2, 7], objetivo = 4`
Salida esperada (cualquiera de las dos es válida): `(-2, 7)` ó `(7, -2)`

- **Ejemplo 4** – Arreglo con un solo par cumpliendo la suma más cercana:

Entrada: `arr = [1, 9, -3, 10], objetivo = 6`
Salida esperada (cualquiera de las 2 es válida): `(-3, 9)` ó `(9, -3)`

[3,1,9,10]

- **Ejemplo 5** – Arreglo con dos pares empatando en cercanía:

Entrada: `arr = [2, 5, 11, -2, 7], objetivo = 9`
Salida esperada (cualquiera de las 4 es válida): `(2, 7)` ó `(7, 2)` ó `(-2, 11)` ó `(11, -2)`

Restricciones:

- No es válida una solución de complejidad temporal cuadrática o superiores. Esto es lo más importante. Si se entrega una solución de complejidad superior a cuadrática, básicamente la calificación es 0.0.
- No es válida una solución que requiera estructuras de datos adicionales al arreglo recibido y a la tupla, la cual se debe usar únicamente para retornar un resultado distinto a `None`.
- La solución debe usar la estructura `array_list` usada en el curso (ver página 2).
- No es válido el uso de métodos de ordenamiento nativos de Python (por ejemplo: `sorted()` o `list.sort()`). Si necesita alguno, puede usar los métodos vistos en clase (ver página 2).
- No puede definir restricciones, condiciones o suposiciones adicionales.

La siguiente, es la rúbrica de calificación que se usaría para calificar la pregunta 1*:

Descripción	Excelente	Bien	Regular	Insuficiente
40 pts si el plan de implementación cumple con todas las restricciones, es correcto y completo. 30 pts si el plan de implementación cumple con todas las restricciones, es completo, pero tiene errores muy leves. 15 pts si el plan de implementación cumple con todas las restricciones, pero tiene omisiones y/o errores leves. 0 pts si el plan de implementación incumple restricciones o tiene errores u omisiones graves. *Aplica también si no se responde o se responde con letra ilegible.*				
Total sobre 40pts:				

Observaciones del profesor:

*** La nota mínima posible sería 0.0.**

Pregunta 1 (40 pts). Escriba de forma muy clara y muy concisa el plan de implementación en texto plano para solucionar el problema: *Par con suma más cercana al objetivo.* Evite descripciones redundantes y verbosas. Si el texto es ilegible, la calificación en el parcial real sería de 0.0.

Pregunta 2 (40 pts). Implemente en Python una función que solucione el problema: *Par con suma más cercana al objetivo*. No es necesario documentar la función ni crear doctests.