A - Maximise The Score

**1. Reducción del Problema**

**Descripción en palabras:**

Dados 2n números enteros positivos en una pizarra, hay que realizar exactamente n movimientos. En cada movimiento, seleccionamos dos números, sumamos el mínimo de ellos a nuestra puntuación inicial (0) y los eliminamos de la pizarra. El objetivo es maximizar la puntuación final.

**Descripción en lenguaje matemático:**

Dado un conjunto A de 2n enteros positivos, encontrar un emparejamiento de los elementos en `n` pares `(x\_i, y\_i)` tal que la suma `Σ min(x\_i, y\_i)` sea máxima.

**2. Reducción de la Solución**

**Descripción en palabras:**

Para maximizar la suma de los mínimos de los pares, se debe emparejar los números más grandes posibles entre sí. Esto se logra ordenando el arreglo y luego sumando los elementos en las posiciones pares (0, 2, 4, etc.), ya que al estar ordenados, cada elemento en posición par será el mínimo de su par con el siguiente elemento (posición impar).

**Descripción en lenguaje matemático:**

1. Ordenar el arreglo `A` de forma ascendente:

2. La suma máxima es `Σ A'[2i]` para `i` desde `0` hasta `n-1`.

**3. Código Realizado y Análisis**

Codigo: <https://vjudge.net/solution/61070276/hlp0QYuoSdpwFIGtfLrL>

**4. Casos de Prueba**

**Caso 1: Caso mínimo posible**

**Entrada**

1

1

5 3

**Salida esperada**

3

**Justificación:** Este es el caso más pequeño válido (n = 1). Al ordenar [5, 3] a [3, 5], y tomar el índice 0, la suma es 3. Sirve para validar que la implementación maneja correctamente entradas mínimas.

**Caso 2: Todos los números iguales**

**Entrada**

1

3

4 4 4 4 4 4

**Salida**

12

**Justificación**

Verifica que el código no depende de diferencias entre los números. La suma de los índices pares de [4, 4, 4, 4, 4, 4] es 4 + 4 + 4 = 12.

**Caso 3: Números en orden descendente**

**Entrada**

1

2

10 8 6 4

**Salida**

12

**Justificación**

La lista [10, 8, 6, 4] se ordena como [4, 6, 8, 10]. Índices pares: 4 + 8 = 12. Verifica que el orden original no afecta la solución.

**Caso 4: Números negativos**

**Entrada**

1

2

-1 -3 -2 -4

Salida

-6

**Justificación:**

Al ordenar: [-4, -3, -2, -1], y sumar índices 0 y 2 => -4 + (-2) = -6. Asegura que el algoritmo maneja correctamente números negativos.

#### **Caso 5: Caso extremo (gran n)**

**Entrada(Parcial)**

1

100000

<200000 enteros aleatorios entre 1 y 1000000>

**Salida**

Un entero alrededor de 30 a 45 mil millones

**Justificación:**Este caso prueba la eficiencia y rendimiento del algoritmo ante un límite alto de entrada. Como el tiempo es O(n log n) y memoria O(n), debería ejecutarse sin errores en tiempo razonable.

Nota: para evitar mostrar 200,000 números aquí, se puede generar con un script o usar generadores**.**

**5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)**

En este ejercicio no lo hice a la primera, a continuación presento el error principal que tuve

**Código de la solución incorrecta:** <https://vjudge.net/solution/61013703/tloufnI2fkK17QFDPYEL>

**Descripción del error:**

La solución falló en el primer caso de prueba con un Runtime Error. Esto indica que el programa intentó ejecutar una operación inválida, como acceder a un índice fuera de rango o dividir por cero.

**Proceso de depuración:**

Revisé la lectura de entradas y el manejo de listas. Descubrí que estaba leyendo n valores cuando en realidad el problema requiere leer 2n enteros por cada caso. Esto causaba que la lista fuera más corta de lo necesario, y al acceder a índices fuera del rango se generaba un error.

**Solución corregida:**

En realidad lo que hice fue dar otro enfoque para dar la solución, así que hice el codigo prácticamente desde cero porque en el anterior no logre corregir el problema(la verdad es que tampoco lo intente mucho solo que cuando estaba haciendo este problema era de noche cuando ya acosté, me quede como pensando en ese problema y así se me ocurrió una idea nueva), además lo hice de una forma más fácil(en términos de líneas de código) y esta vez sí lo hice a la primera sin tener un error

**Estrategia:**

Lo que hice fue que a cada movimiento se sumarán el mínimo de dos enteros seleccionados. Por tanto, para maximizar la suma total de mínimos posibles, conviene emparejar los números más pequeños entre sí.

Para lograr esto, primero ordené la lista de 2n enteros de menor a mayor. Luego, seleccioné los elementos en las posiciones pares (índices 0, 2, 4, ..., 2n−2). Estos elementos son los menores de cada pareja consecutiva en la lista ordenada, por lo que al sumarlos se obtiene la suma máxima posible de mínimos.

**6. Preguntas de Aprendizaje**

**Temas aplicados:**

- Ordenamiento de arreglos.

- Greedy algorithms (algoritmos voraces).

**Dificultad de la implementación:**

La mayor dificultad fue asegurarse de que la estrategia de sumar los elementos en posiciones pares después de ordenar realmente maximiza la suma. Una vez entendido, la implementación fue sencilla.

**Recursos utilizados:**

Videos de youtube para saber los ordenamientos y cómo se implementan en este caso en Python.

**7. Feedback LLM (En este caso ChatGPT)**

**Código:** [**https://vjudge.net/solution/61083064/A9Y95HGSJnrFWG4p7UeY**](https://vjudge.net/solution/61083064/A9Y95HGSJnrFWG4p7UeY)

**Comparación con código generado por LLM:**

Ambos tienen la misma idea, solo que en este caso el mismo es más eficiente y **ChatGPT** se puede decir que está más organizado quizás sea mejor para testear y también es más fácil de reutilizar.

Para este taller el mejor seria el mio porque es más eficiente por temas en vjudge puede haber exceso de límite de tiempo, si ya sí algún momento algo así se aplicará en la vida real ahí sí el mejor seria el de ChatGPT porque es más fácil para reutilizar el mio tendria que hacer modificaciones para que se pueda reutilizar la verdad

**Feedback de LLM:** le pedí ChatGPT que evaluara mi código, fuera de lo que me dice y que bien y es funcional, estas son las observaciones que puso que según él eran menores

**lo que dijo ChatGPT**

| **Área** | **Comentario** |
| --- | --- |
| **Nombres de variables** | Usas a, n, t que son correctos pero podrían ser más semánticos (numeros, casos, cantidad, etc.) si el código fuera para un público más amplio. |
| **Documentación** | Aunque está comentado con complejidad, podrías añadir un breve comentario sobre la **idea del algoritmo** justo antes de a.sort() para quien no conozca el problema. |
| **Validación de entrada (opcional)** | No hay validación de que efectivamente se estén leyendo 2n valores. Esto está bien si confías 100% en el input, pero puede causar errores si el input llega corrupto o incompleto. |

Y en la recomendaciones me dijo

* que ponga la validación minina aunque dijo que es opcional
* que al inicio del codigo ponga la estrategia o lo que tengo que hacer por si estoy en un equipo de trabajo algo como esto:

**# Estrategia: ordenar los 2n números y sumar los menores de cada par consecutivo.**

* que ponga funcionalidades para si algun momento lo quiera reutilizar

y me manda una versión el código que me recomienda(no lo mandé vjudge porque el cambio fue mínimo)

