Escuela de ingenieros Julio Garavito

Ingeniería de inteligencia artificial

Laboratorio Semana 3

Ejercicios en clase

Diseño con D&Q del programa de los estudiantes.

Edward Alexander Francia Guzman

Diseño de datos y algoritmos

ID: 1000110781

Sergio Alejandro Ariza Ocampo

15/02/26

Adaptación del programa de los estudiantes con la técnica Dividir y Conquistar.

Descripción del problema

El problema consiste en recibir como entrada un número entero que representa la cantidad de estudiantes y, posteriormente, una secuencia de datos que alterna entre el nombre del estudiante y su respectiva nota (todo esto en una sola línea). El sistema debe identificar cuáles estudiantes aprobaron la asignatura, considerando que se aprueba con una nota mayor o igual a 3.0, debe mostrar los nombres de los estudiantes aprobados Aunque anteriormente, resolvimos este problema mediante un recorrido lineal, en este trabajo se usara la estrategia de Dividir y Conquistar.

Requerimientos (Historias de usuario)

RF-01. Leer cantidad de estudiantes. El sistema debe leer un número entero n que representa la cantidad de estudiantes a procesar.

RF-02. Capturar información de estudiantes. El sistema debe leer a continuación 2n valores que alternan entre nombre y nota.

RF-03. Almacenar datos del grupo. El sistema debe organizar los datos en una estructura que relacione cada estudiante con su nota.

RF-04. Dividir el conjunto de estudiantes. El sistema debe aplicar la técnica de Divide y Conquistar, dividiendo recursivamente el conjunto hasta llegar a casos base.

RF-05. Evaluar condición de aprobación. El sistema debe considerar aprobado a todo estudiante cuya nota sea mayor o igual a 3.0.

RF-06. Combinar resultados parciales. El sistema debe unir los resultados obtenidos de cada subgrupo preservando el orden original.

RF-07. Mostrar estudiantes aprobados. El sistema debe imprimir en una sola línea los nombres de los estudiantes aprobados separados por un espacio.

Historias de usuario

- HU-01. Ingresar datos del grupo

Ingresar el número de estudiantes y sus respectivos nombres y notas para que el sistema procese correctamente la información.

Caso correcto:

Entrada:

3 Ana 4.0 Luis 3.5 Maria 2.8

El sistema reconoce 3 estudiantes y asocia correctamente cada nombre con su nota.

Caso incorrecto:

Entrada:

3 Ana 4.0 Luis

El sistema detecta que faltan datos para completar los 3 pares nombre-nota.

- HU-02. Filtrar estudiantes aprobados usando Divide y Conquistar

Quiero que el sistema divida el grupo en subgrupos y filtre recursivamente los estudiantes con nota mayor o igual a 3.0 para conocer quiénes aprobaron.

Caso correcto:

Entrada:

4 Ana 4.5 Luis 2.0 Maria 3.0 Juan 3.8

Salida esperada:

Ana Maria Juan

El sistema divide la lista, evalúa cada caso base y combina correctamente los resultados.

Caso incorrecto:

Entrada:

4 Ana 4.5 Luis 2.0 Maria 3.0 Juan 3.8

Salida incorrecta:

Ana Juan

En este caso se omite a Maria, aunque su nota es 3.0, lo cual indica error en la condición de evaluación.

- HU-03. Mostrar resultados en el formato correcto

Visualizar únicamente los nombres de los estudiantes aprobados en una sola línea y separados por espacios.

Caso correcto:

Entrada:

2 A 3.0 B 5.0

Salida:

A B

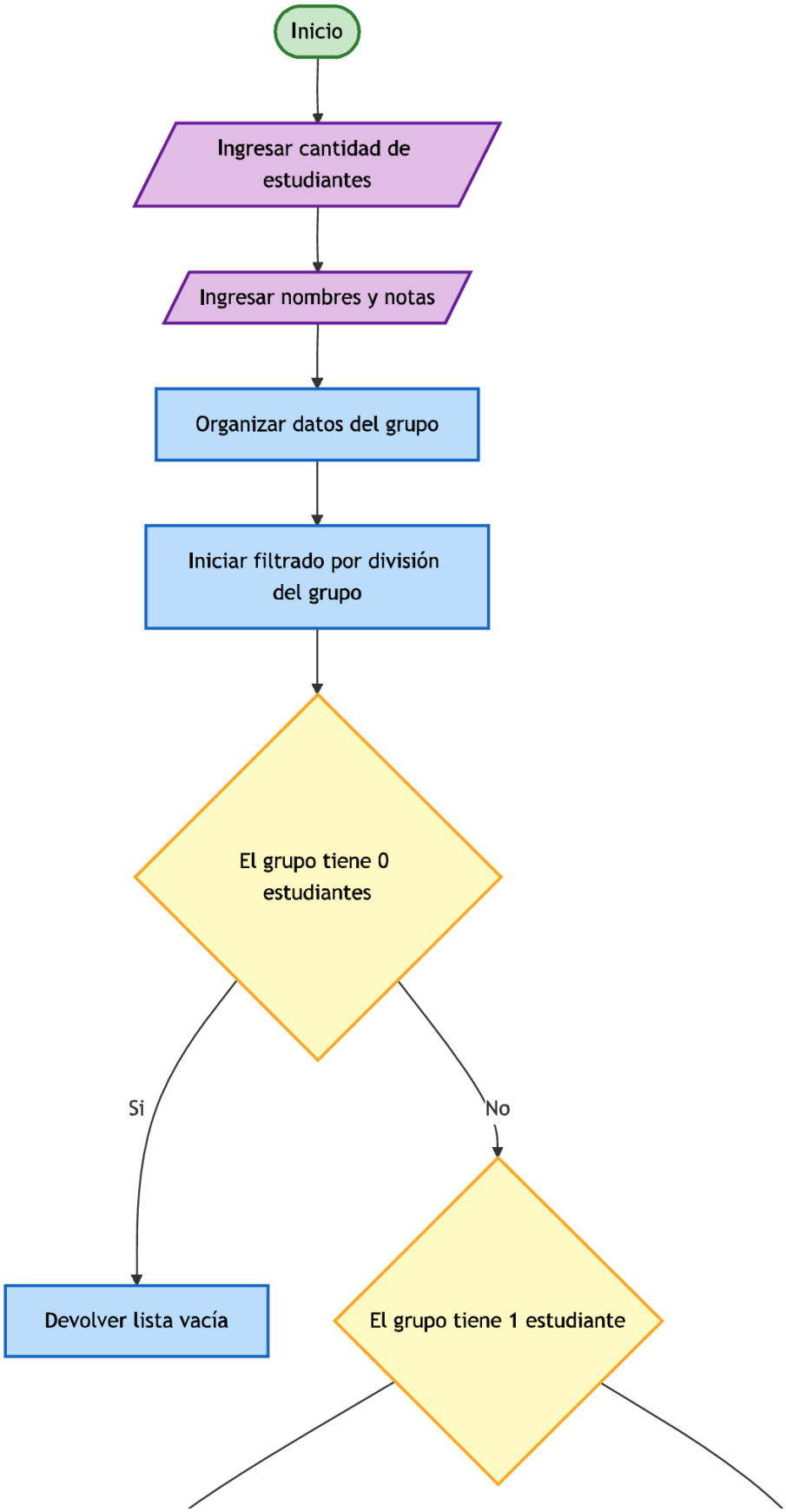
Caso incorrecto:

Salida incorrecta:

A, B

La salida no debe contener comas ni otros símbolos, solo espacios entre nombres.

Figura 1. Diagrama de flujo.



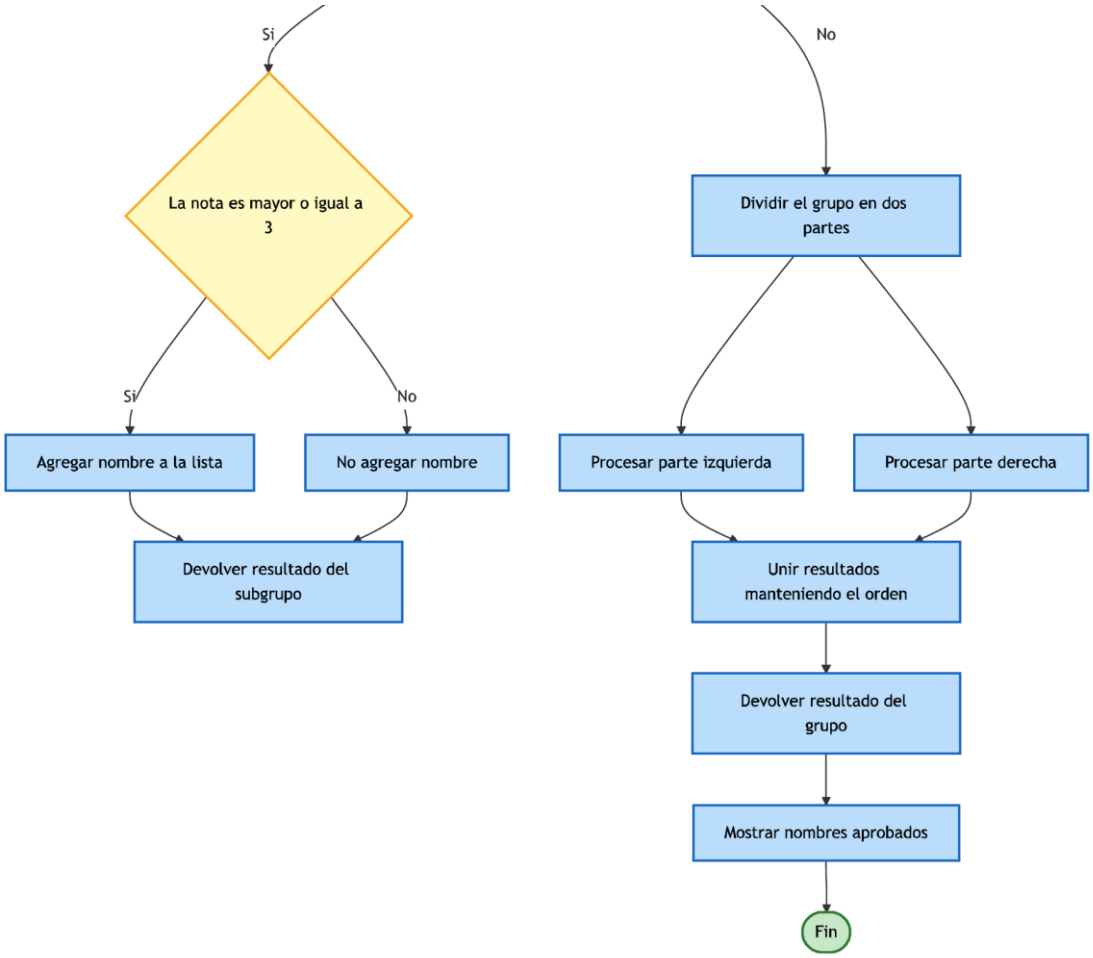


Figura 2. Diagrama de secuencia Usuario–Sistema

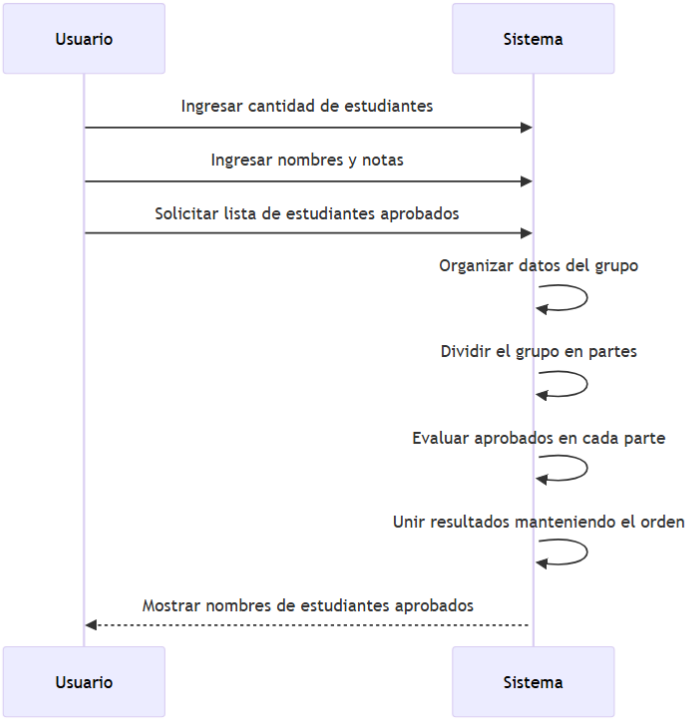
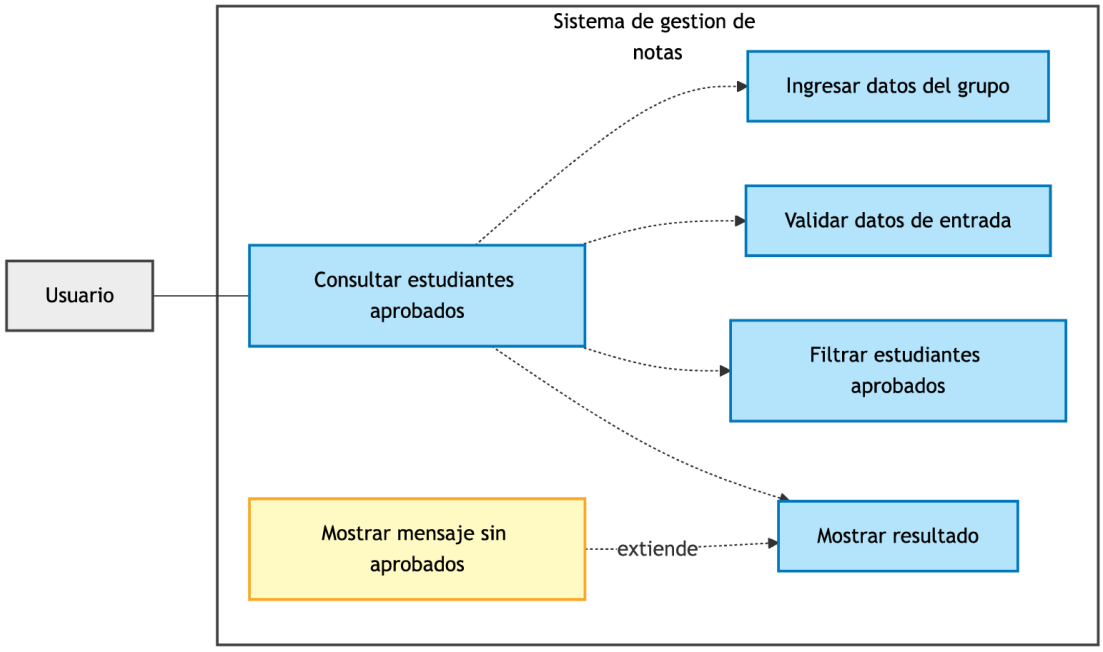


Figura 3. Diagrama de caso de uso



Análisis de complejidad

Análisis del Código

El algoritmo divide el grupo de estudiantes en dos mitades, resuelve cada mitad recursivamente y luego combina los resultados manteniendo el orden.

Código:

```
def filtrar_aprobados(pares, l, r):
    """Retorna lista con los nombres de estudiantes aprobados (nota >= 3.0)"""
    # Casos base
    if r - l == 0:
        return []
    if r - l == 1:
        nombre, nota = pares[l]
```

```
        return [nombre] if nota >= 3.0 else []
    # Dividir
    m = (l + r) // 2
    # Conquistar
    izquierda = filtrar_aprobados(pares, l, m)
    derecha = filtrar_aprobados(pares, m, r)
    return izquierda + derecha
def main():
    datos = input().split()
    n = int(datos[0])
    tokens = datos[1:]
    pares = []
    for i in range(0, 2 * n, 2):
        nombre = tokens[i]
        nota = float(tokens[i + 1])
        pares.append((nombre, nota))
    aprobados = filtrar_aprobados(pares, 0, n)
    print(*aprobados, sep = " ")
main()
```

Análisis tiempo-veces

Análisis del main

Código	**Costo**	**Ejecuciones**
datos = input().split()	c1	1
n = int(datos[0])	c2	1
tokens = datos[1:]	c3	1
pares = []	c4	1
for i in range(0, 2*n, 2):	c5	n
nombre = tokens[i]	c6	n
nota = float(tokens[i + 1])	c7	n
pares.append((nombre, nota))	c8	n
aprobados = filtrar_aprobados(pares, 0, n)	T(n)	1
print(*aprobados, sep=" ")	c9	m

Calculo complejidad:

$$T_{main}(n) = c_1 + c_2 + c_3 + c_4 + (c_5 + c_6 + c_7 + c_8)n + c_9 m$$

Esto es:

$$T_{main}(n) = O(n + m) = O(n)$$

Análisis de la función filtrar_aprobados

Parte	**Costo**	**Ejecuciones**
if r - l == 0	d1	1
if r - l == 1	d2	1
pares[l] y comparación nota >= 3.0	d3	solo en caso base
m = (l + r) // 2	d4	1
Llamada izquierda	T(k/2)	1
Llamada derecha	T(k/2)	1
izquierda + derecha	d5·k	1

análisis complejidad:

Sea $T(n)$ el tiempo de filtrar_aprobados sobre n estudiantes.

- Divide en dos: $2T(n/2)$
- Combina: $O(n)$

$$T(n) = 2T(n/2) + c\,n$$

Resolución por Teorema Maestro

- $T(n) = aT(n/b) + f(n)$
- $a = 2$
- $b = 2$
- $f(n) = c\,n$

$$n^{\log_b a} = n^{\log_2 2} = n\text{^}1 = n$$

Como $f(n) = \Theta(n)$, Caso 2:

$$T(n) = \Theta(n\log\ n)$$

Complejidad total del programa

$$T_{total}(n) = T_{main}(n) + T(n)$$

Como:

- $T_{main}(n) = O(n)$
- $T(n) = O(n\log\ n)$

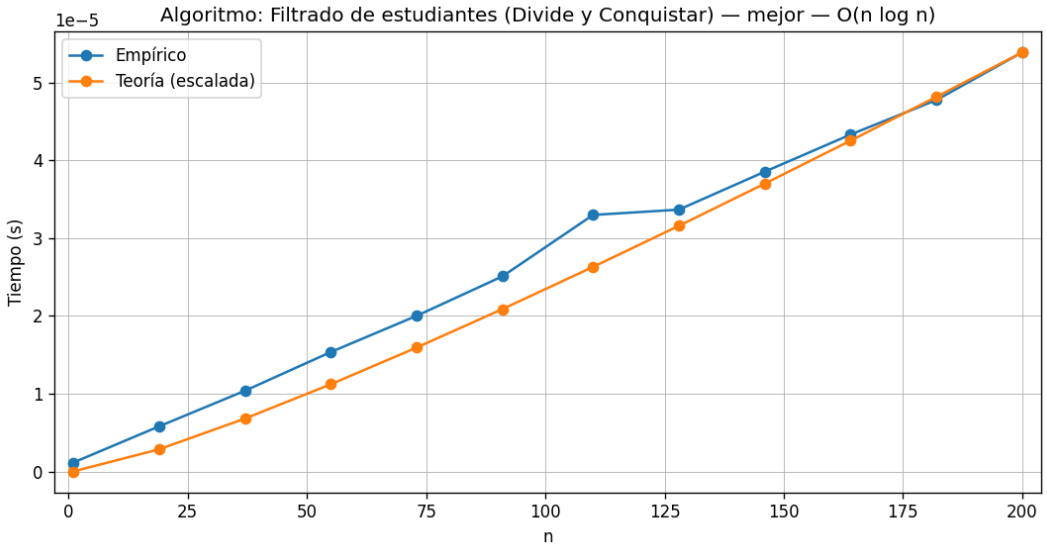
Entonces:

$$T_{total}(n) = O(n\log\ n)$$

Mejor caso y peor caso

- **Mejor caso:** Aunque nadie apruebe ($m = 0$), la recursión divide igual y combina igual.

$$T_{mejor}(n) = O(n\log\ n)$$



- **Peor caso**

Si todos aprueban ($m = n$), se copian más elementos en concatenaciones, pero el orden de crecimiento sigue siendo:

$$T_{peor}(n) = O(n\log\ n)$$