Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Análisis de algoritmos
Parcial 1 - Marzo 7 de 2025 (6pm-8pm)
Nombre: Solución

Reglas y recomendaciones

Durante el parcial, deben observarse las siguientes reglas:

1. El parcial es estrictamente individual y se debe desarrollar en el salón de clase.
2. **Las únicas respuestas válidas serán las contenidas el reverso de esta hoja y la hoja en blanco dada.**
3. Está absolutamente prohibido el uso de cualquier herramienta de comunicación (digital o análoga) entre usted y cualquier otro ser humano.
4. **Toda pregunta acerca del enunciado del parcial se debe hacer desde el puesto en voz alta.**

Diámetro de una secuencia

El diámetro D de una secuencia S se define como la mayor diferencia absoluta entre dos elementos cualquiera de S. Por ejemplo, para la secuencia:

$$S = \langle -37, 85, -12, 5, -98, 23, 67, -52, 11, -7 \rangle$$

su diámetro es $D = 183$, donde los valores de referencia son -98 y 85. El algoritmo de fuerza bruta para calcular el diámetro de una secuencia es:

Algoritmo 1 Algoritmo de fuerza bruta para calcular el diámetro de una secuencia.

```
procedure DIAMETRO(S)
    D ← 0
    for a ∈ S do
        for b ∈ S do
            if D < |a − b| then
                D ← |a − b|
            end if
        end for
    end for
    return D
end procedure
```

En este parcial usted debe comparar este algoritmo con uno basado en la estrategia "dividir-y-vencer". Para esto, usted debe escribir formalmente dicho algoritmo y analizar su complejidad. Recuerde que la escritura formal de un algoritmo se compone de la siguiente información:

1. (10%) Análisis del problema.
2. (20%) Diseño del problema.
3. (40%) Descripción de la estrategia algorítmica.
4. (10%) Seudo-código.
5. (20%) Análisis de complejidad y comparación con el algoritmo de fuerza bruta.

NOTA: se espera que el desarrollo que usted haga de este parcial sea ordenado y con buena ortografía. La falta de rigor en estos aspectos será penalizado con un 10% de la nota final del parcial.

1 (10%) Análisis del problema.

Evaluación:

- ***5%***: Explicar que hay que encontrar la pareja de números con la diferencia más grande, es decir, son los valores máximo y mínimo de la secuencia de entrada.
- ***5%***: formalizar las restricciones sobre los elementos de la secuencia.

En escencia, el problema se puede parafrasear como encontrar la pareja (min, max) e informar $max - min$.

1.1 Algoritmo que se apoya en otros algoritmos "dividir-y-vencer"

Si la secuencia estuviera ordenada, el diámetro (diferencia de los valores extremos) se calcula como la diferencia absoluta entre el primer y el último elemento (es decir, la diferencia entre el mayor y el menor).

1.2 Algoritmo puro "dividir-y-vencer"

Es importante garantizar que el conjunto al cual pertenecen los elementos de la secuencia tenga definidas las operaciones resta y valor absoluto, para calcular la distancia, y la relación de orden total para poder maximizar las distancias que se calculan.

2 (20%) Diseño del problema.

Evaluación:

- ***10%***: Definir formalmente las entradas.
- ***10%***: Definir formalmente las salidas.

Contenido esperado:

- ***Entradas***: $S = \langle s_i \in \mathbb{T} \rangle$, una secuencia de elementos pertenecientes a un conjunto $\mathbb{T}$ donde están definidas las operaciones resta $(-)$, valor absoluto $(|\cdot|)$ y la relación de orden total $(<)$.
- ***Salidas***: $D \in \mathbb{T}$, el diámetro de la secuencia S.

3 (40%) Descripción de la estrategia algorítmica.

Evaluación:

- ***10%***: Describir el caso base.
- ***15%***: Describir la estrategia de pivoteo y división.
- ***15%***: Describir la estrategia de mezcla.

3.1 Algoritmo que se apoya en otros algoritmos "dividir-y-vencer"

Como el problema se soluciona con la diferencia entre los elementos extremos, el problema se puede solucionar con dos búsquedas (de los valores mínimo y máximo) basadas en la estrategia "dividir-y-vencer":

1. ***Caso base***: el tipo de secuencia más sencilla sobre la cual se puede calcular los extremos son las unitarias $S = \langle a \rangle$, cuyos extremos son a (al tiempo es el mínimo y máximo).
2. ***Estrategia de "pivoteo" y división***: el pivote h se puede poner en la mitad de la secuencia o usar la partición aleatoria para reducir recurrentemente el problema a encontrar los extremos de la subsecuencia izquierda E_i y la subsecuencia derecha E_d.
3. ***Estrategia de "mezcla" de soluciones***: con los extremos E_i y E_d calculados recurrentemente, se modifican comparándolos con el pivote h.

3.2 Algoritmo puro "dividir-y-vencer"

Como se pide formular un algoritmo basado en la estrategia "dividir-y-vencer", definimos los pasos básicos como:

1. ***Caso base***: el tipo de secuencia más sencilla sobre la cual se puede calcular un diámetro son las vacías $S = \emptyset$, cuyo diámetro es $D = 0$.

2. ***Estrategia de "pivoteo" y división***: el pivote h se pone en la mitad de la secuencia para reducir recurrentemente el problema a encontrar los diámetros de la subsecuencia izquierda D_i y la subsecuencia derecha D_d.

3. ***Estrategia de "mezcla" de soluciones***: con los diámetros D_i y D_d calculados recurrentemente, se calcula el diámetro que tiene al pivote h como referencia.

4 (10%) Seudo-código.

Evaluación:

- ***3%***: Implementar el caso base.
- ***3%***: Implementar estrategia de pivoteo.
- ***4%***: Usar correctamente las recurrencias.

4.1 Algoritmo que se apoya en otros algoritmos "dividir-y-vencer"

Algoritmo 2 Algoritmo "dividir-y-vencer" para calcular los extremos de una secuencia.

```
procedure MinValue(S, b, e)
    if b = e then
        return s_b
    else
        h ← ⌊(b + e) ÷ 2⌋
        E_i ← MinValue(S, b, h − 1)
        E_d ← MinValue(S, h + 1, e)
        return min (E_i, E_d, s_h)
    end if
end procedure
procedure MaxValue(S, b, e)
    if b = e then
        return s_b
    else
        h ← ⌊(b + e) ÷ 2⌋
        E_i ← MaxValue(S, b, h − 1)
        E_d ← MaxValue(S, h + 1, e)
        return max (E_i, E_d, s_h)
    end if
end procedure
procedure Diametro(S)
    x ← MinValue(S, 1, |S|)
    y ← MaxValue(S, 1, |S|)
    return y − x
end procedure
```

4.2 Algoritmo puro "dividir-y-vencer"

Es importante recordar que la solución puede estar basada en el pivoteo por partición aleatoria. En este caso, los órdenes de complejidad son diferentes.

Algoritmo 3 Algoritmo "dividir-y-vencer" para calcular el diámetro de una secuencia.

```
procedure DiametroAux(S, b, e)
    if b > e then
        return 0
    else
        h ← ⌊(b + e) ÷ 2⌋
        D_i ←DiametroAux(S, b, h − 1)
        D_d ←DiametroAux(S, h + 1, e)
        D_h ← 0
        for i ← h downto 1 do
            if D_h < |s_h − s_i| then
                D_h ← |s_h − s_i|
            end if
        end for
        for i ← 1 to |S| do
            if D_h < |s_h − s_i| then
                D_h ← |s_h − s_i|
            end if
        end for
        return max (D_i, D_d, D_h)
    end if
end procedure
```

5 (20%) Análisis de complejidad y comparación con el algoritmo de fuerza bruta.

Evaluación:

- ***2%***: Mostrar y explicar el orden de complejidad del algoritmo de fuerza bruta.
- ***3%***: Mostrar y explicar el orden de complejidad del algoritmo de "dividir-y-vencer".
- ***15%***: Comparar ambos órdenes de complejidad y concluir argumentadamente.

5.1 Algoritmo que se apoya en otros algoritmos "dividir-y-vencer"

5.1.1 Pivoteo por la mitad del rango

Para este análisis se define $n = |S|$, la cardinalidad (tamaño) de la secuencia S.

Por inspección de código, el algoritmo de fuerza bruta tiene un orden de complejidad $O\left(n^2\right)$. Para el algoritmo "dividir-y-vencer" exhibe el polinomio temporal:

$$T\left(n\right) = 2T\left(\frac{n}{2}\right) + O\left(1\right)$$

que se debe analizar con el teorema maestro:

- ***Caso 1***: $1 = n^0 = n^{\log_2(2-\epsilon)} \implies 0 = \log_2\left(2-\epsilon\right) \implies 2^0 = 2 - \epsilon \implies \epsilon = 1$
- ***Caso 2***: $1 = n^{\log_2 2} \implies n^0 = n^1$
- ***Caso 3***: $1 = n^0 = n^{\log_2(2-\epsilon)} \implies 0 = \log_2\left(2-\epsilon\right) \implies 2^0 = 2 - \epsilon \implies \epsilon = -1$

Mientras los casos 2 y 3 no son válidos a luz del teorema maestro, el ***caso 1*** se mantiene; entonces el algoritmo tiene un orden de complejidad $\Theta\left(n\right)$.

Únicamente observando los órdenes de complejidad, el algoritmo basado en la estrategia "dividir-y-vencer" es más rápido para solucionar el problema porque un crecimiento lineal es más lento que un crecimiento cuadrático.

5.1.2 Pivoteo por partición aleatoria

Para este análisis se define $n = |S|$, la cardinalidad (tamaño) de la secuencia S.

Por inspección de código, el algoritmo de fuerza bruta tiene un orden de complejidad $O\left(n^2\right)$. Para el algoritmo "dividir-y-vencer" exhibe el polinomio temporal:

$$T\left(n\right) = T\left(\frac{n}{2}\right) + O\left(1\right)$$

que se debe analizar con el teorema maestro:

- ***Caso 1***: $1 = n^0 = n^{\log_2(1-\epsilon)} \Longrightarrow 0 = \log_2(1-\epsilon) \Longrightarrow 2^0 = 1-\epsilon \Longrightarrow \epsilon = 0$
- ***Caso 2***: $1 = n^{\log_2 1} \Longrightarrow n^0 = n^0$
- ***Caso 3***: $1 = n^0 = n^{\log_2(1-\epsilon)} \Longrightarrow 0 = \log_2(1-\epsilon) \Longrightarrow 2^0 = 1-\epsilon \Longrightarrow \epsilon = 0$

Mientras los casos 1 y 3 no son válidos a luz del teorema maestro, el ***caso 2*** se mantiene; entonces el algoritmo tiene un orden de complejidad $\Theta(\log_2 n)$.

Únicamente observando los órdenes de complejidad, el algoritmo basado en la estrategia "dividir-y-vencer" es más rápido para solucionar el problema porque un crecimiento logarítmico es más lento que un crecimiento cuadrático.

5.2 Algoritmo puro "dividir-y-vencer"

Para este análisis se define $n = |S|$, la cardinalidad (tamaño) de la secuencia S.

Por inspección de código, el algoritmo de fuerza bruta tiene un orden de complejidad $O(n^2)$. Para el algoritmo "dividir-y-vencer" exhibe el polinomio temporal:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

que se debe analizar con el teorema maestro:

- ***Caso 1***: $n = n^{\log_2(2-\epsilon)} \Longrightarrow 1 = \log_2(2-\epsilon) \Longrightarrow 2^1 = 2-\epsilon \Longrightarrow \epsilon = 0$
- ***Caso 2***: $n = n^{\log_2 2} \Longrightarrow n = n^1$
- ***Caso 3***: $n = n^{\log_2(2+\epsilon)} \Longrightarrow 1 = \log_2(2+\epsilon) \Longrightarrow 2^1 = 2+\epsilon \Longrightarrow \epsilon = 0$

Mientras los casos 1 y 3 no son válidos a luz del teorema maestro, el ***caso 2*** se mantiene; entonces el algoritmo tiene un orden de complejidad $\Theta(n \log_2 n)$.

Únicamente observando los órdenes de complejidad, el algoritmo basado en la estrategia "dividir-y-vencer" es más rápido para solucionar el problema porque un crecimiento lineal-logarítmico es más lento que un crecimiento cuadrático.