Análisis y Diseño de Algoritmos

Ejercicios Tema 4

Francisco Javier Mercader Martínez

1) Diseñar un algoritmo para calcular el mayor y el segundo mayor elemento de un array de enteros utilizando la técnica divide y venceras.
Calcular el número de comparaciones realizadas en el peor y el mejor caso suponenido `n` potencia de 2.
¿Sería el orden obtenido extrapolable a un `n` que no sea potencia de 2?

1) Algoritmo de Divide y Vencerás

   La idea principal es dividir el problema en subproblemas más pequeños, resolverlos y luego cominar las soluciones.

   Esquema General:

   1) División:
      - Divide el arreglo en dos mitades iguales.
   2) Conquista:
      - Encuentra el mayor y el segundo mayor en cada mitad de forma recursiva.
      - Combinación:
        - Combina las soluciones de las dos mitades:
          - El mayor de las dos mitades es el mayor global.
          - El segundo mayor será el mayor de:
            - El segundo mayor de la mitad que contiene el mayor global.
            - El mayor de la otra mitad

   Implementación

```python
def encontrar_mayores(arr):
    """
    Encuentra el mayor y el segundo mayor elemento de un arreglo utilizando divide y
        vencerás.

    - arr: Lista de enteros.
    Retorna: (mayor, segundo_mayor, comparaciones)
    """
    def dividir_y_vencer(arr):
        # Caso base: Si hay solo dos elementos, compara directamente
        if len(arr) == 2:
            if arr[0] > arr[1]:
                return arr[0], arr[1], 1 # mayor, segundo mayor, comparaciones
            else:
                return arr[1], arr[0], 1
```

```
        # Divide el arreglo en dos mitades
        mid = len(arr) // 2
        izq_mayor, izq_segundo, izq_comparaciones = dividir_y_vencer(arr[:mid])
        der_mayor, der_segundo, der_comparaciones = dividir_y_vencer(arr[mid:])

        # Combina las soluciones
        comparaciones = izq_comparaciones + der_comparaciones

        if izq_mayor > der_mayor:
            mayor = izq_mayor
            segundo_mayor = max(izq_segundo, der_mayor)
        else:
            mayor = der_mayor
            segundo_mayor = max(der_segundo, izq_mayor)

        comparaciones += 2 # Comparaciones para determinar mayor y segundo mayor
        return mayor, segundo_mayor, comparaciones

    # Llamar a la función recursiva
    return dividir_y_vencer(arr)

# Ejemplo de uso
if __name__ == '__main__':
    arr = [10, 3, 5, 7, 9, 2, 6, 8]
    mayor, segundo_mayor, comparaciones = encontrar_mayores(arr)
    print(f"Mayor elemento: {mayor}")
    print(f"Segundo mayor elemento: {segundo_mayor}")
    print(f"Total de comparaciones: {comparaciones}")
```

2) Análisis del número de comparaciones

Peor caso y mejor caso

El número de comparaciones es el mismo en el peor y el mejor caso, ya que cada división implica el mismo número de subproblemas y combinaciones.

1) Caso base:
   - Para $n = 2$: Se realiza una única comparación.
2) Generalización para $n = 2^k$:
   - Hay $n - 1$ comparaciones para encontrar el mayor (esto se debe a que cada elemento pierde contra el mayor exactamente una vez).
   - Hay $\log_2(n) - 1$ comparaciones adicionales para encontrar el segundo mayor, porque el segundo mayor es el que pierde contra el mayor en el "torneo".

     Total de comparaciones:

$$T(n) = (n-1)(log_2(n) - 1) = n\log_2(n) - 2.$$

3) Orden para n no potencia de 2

Cuando n no es potencia de 2, el algoritmo sigue siendo aplicable:

1) Se rellena el arreglo con elementos adiciones $(-\infty)$ para completar una potencia de 2.

2) Las comparaciones adicionales no afectan significativamente el orden del algoritmo, ya que $T(n) = O(n + \log_2(n))$ sigue siendo válido.

2)