Diseño y análisis de algoritmos. Tarea 2.
Técnica de diseño: Divide y vencerás.
Fecha de entrega: 20 de septiembre

9 de septiembre de 2022

1. Resumen

1.1. Divide y vencerás

Esta técnica consiste en que dada una instancia a resolver, reducir el problema (recursivamente) a la solución de varias subinstancias más simples o más pequeñas y combinar las soluciones de esas subinstancias para construir la de la instancia original, es decir:

1. Divide: Crea subinstancias más simples o más pequeñas.
2. Resuelve: Resuelve esas subinstancias.
3. Combina las soluciones.

2. Tarea

1. Considera el siguiente problema:

   Problema 1 (Inserción) *Dado un arreglo ordenado de n números enteros positivos, A, y un número entero positivo x, inserta el número x en el arreglo A de tal forma que A se mantenga ordenado y devuelve el nuevo arreglo A'.*

   Propiedad 1 *Un arreglo A tiene la propiedad de ordenamiento si para todo par de índices i, j en el arreglo A se cumple que $(i < j) \implies A[i] \leq A[j]$.*

   a) propón un algoritmo que resuelva este problema.

   b) Demuestra que el algoritmo es correcto.

   c) Obtén el orden asintótico del algoritmo.

2. A continuación se presenta un algoritmo que resuelve el siguiente problema:

   Problema 2 (Conteo de inversiones significativas) *Dado un arreglo A de números enteros no ordenado, devuelve el número de inversiones significativas.*

   Definición 1 *Dado un arreglo A de números enteros positivos, suponemos que todos los números de A son diferentes, definimos una inversión significativa como una pareja de índices $(i < j)$ del arreglo A, tal que $A[i] > 2A[j]$.*

   a) Demuestra que el algoritmo *mergeSortCount(A)* devuelve como segundo elemento de su valor de retorno, el número de inversiones significativas del arreglo A.

   b) Obtén el orden asintótico del algoritmo.

```
función mergeSortCount(A) :
    n = len(A)
    if n < 2: return (A[:], 0)
    L = A[: n//2]
    R = A[n//2 :]
    (L', siL') = mergeSortCount(L)
    (R', siR') = mergeSortCount(R)
    siR'L' = mergeCount(L', R')
    siTotal = siL' + siR' + siR'L'
    return ( merge(L', R'), siTotal)
```

```
función mergeCount(L, R) :
    i = len(L) − 1                      ▷ último índice de L
    j = len(R) − 1                      ▷ último índice de R
    countSI = 0                         ▷ número de inversiones significativas
    while i ≥ 0 and j ≥ 0:
        if 2 · R[j] < L[i]:
            countSI+ = j + 1
            i-=1
        else:
            j-=1
    return countSI
```

```
función merge(A, B) :                          ▷ visto en clase
    i = j = 0
    C = []
    while i < len(A) and j < len(B):
        if A[i] ≤ B[j]:
            C.append(A[i])
            i+=1
        else:
            C.append(B[j])
            j+=1
    while i < len(A):
        C.append(A[i])
        i+=1
    while j < len(B):
        C.append(B[j])
        j+=1
return C
```

3. Considera el siguiente algoritmo recursivo (visto en clase):

```
función GCD(a, b) :
    if a == 0:
        return b
    else:
        return GCD(b%a, a)
```

   a) Propón un algoritmo iterativo equivalente.

   b) Demuestra que tu algoritmo iterativo soluciona el problema del máximo común divisor.

   c) Argumenta sobre su complejidad.

4. Considera el siguiente problema.

   Problema 3 (¿Es palíndromo?) *Dado un arreglo A de caracteres de longitud n, tal que $n > 0$. Determina si el arreglo A es palíndromo.*

   Definición 2 *Un arreglo $A = [a_0, a_1, \ldots, a_i, \ldots, a_{n-1}]$ es un palíndromo si $[a_0, a_1, \ldots, a_i, \ldots, a_{n-1}] = [a_{n-1}, \ldots, a_i, \ldots, a_1, a_0]$.*

   Ejemplo de arreglos que son palíndromos:

   - $[a]$
   - $[a, a]$
   - $[a, b, a]$
   - $[r, o, t, o, m, o, t, o, r]$

   a) Propón un algoritmo que solucione este problema, con una complejidad de $O(n)$.

   b) Demuestra que tu algoritmo propuesto soluciona el problema planteado.

   c) Argumenta sobre su complejidad.

5. Considera el siguiente problema.

   Problema 4 (Búsqueda binaria) *Dados un entero x y un arreglo ordenado A de $n \geq 0$ números enteros, devuelve algún índice de A en el que se encuentra el número x, o -1 si x no está en A.*

   a) Propón un algoritmo de complejidad $O(\log n)$ que resuelva el problema.

   b) Demuestra que el algoritmo propuesto es correcto.

   c) Demuestra que el algoritmo propuesto tiene la complejidad solicitada.