Universidad Nacional Autónoma de México

Facultad de Ciencias

Análisis y Diseño de Algoritmos

Tarea 4

Martínez Avila Santiago

Reynoso Sánchez Arturo Yitzack

19 de noviembre de 2021

1

Peso: 10 puntos. Sea $S = s_1, s_2, ..., s_n$ una secuencia de n elementos. Decimos que la pareja s_i, s_j está *ordenada* si $i < j$ y $s_i \leq s_j$. Por ejemplo, en la secuencia 2,3,5,1,3, el número 2 forma parejas ordenadas con los números 3, 5 y 3; el primer 3 forma parejas ordenadas con los números 5 y (el segundo) 3.

Problema (número de parejas ordenadas). Entrada: una secuencia S de n elementos. Salida: el número de parejas ordenadas en S. Es evidente que el problema se puede resolver por fuerza bruta en tiempo $O(n^2)$. Dá un algoritmo que lo resuelva en tiempo $O(n \log n)$. Instancias de ejemplo. Entrada: 2, 3, 3. Salida: 3. Entrada: 1, 6, 3, 2. Salida: 3.

Escribe el algoritmo, analiza su complejidad y justifica que el algoritmo es correcto.

Tip: La clave es observar que, si ordenamos los primeros $n/2$ elementos (parte inicial de la secuencia), y aparte ordenas el resto de los elementos (parte final de la secuencia), entonces, (con una variante del algoritmo de fusión de arreglos ordenados), puedes calcular en tiempo $O(n)$ el número de parejas ordenadas tal que uno de los elementos de la pareja pertenece a la parte incial, y el otro es un elemento de la parte final.

Algorithm 1 Pares Ordenados

procedure $\textsc{parOrd}(A, n)$
    $B \leftarrow [0] * n$
    return $\textsc{mergeSort}(A, B, 0, n-1)$
end procedure

Algorithm 2 Merge Sort

procedure $\textsc{mergeSort}(A, B, izq, der)$
    $c \leftarrow 0$
    if $izq < der$ **then**
        $mid \leftarrow (izq + der)//2$
        $c \leftarrow c + \textsc{mergeSort}(A, B, izq, mid)$
        $c \leftarrow c + \textsc{mergeSort}(A, B, mid+1, der)$
        $c \leftarrow c + \textsc{merge}(A, B, izq, mid, der)$
    end if
    return c
end procedure

Correctitud

A continuación demostratremos por qué el algoritmo $\textsc{merge}$ es correcto.

Proponemos el **invariante:** En la $t - \acute{e}sima$ iteración, c es la suma del número de parejas ordenadas que se pueden formar con $A[izq : mid]$ y $A[mid+1 : der]$ de las últimas $t - 1$

Algorithm 3 Merge

```
procedure MERGE(A, B, izq, mid, der)
    i ← izq
    j ← mid + 1
    k ← izq
    c ← 0
    while i ≤ mid and j ≤ der do
        if A[i] ≤ A[j] then
            B[k] ← A[i]
            c ← der − j + 1
            k ← k + 1
            i ← i + 1
        else
            B[k] ← A[j]
            k ← k + 1
            j ← j + 1
        end if
    end while
    while i ≤ mid do
        B[k] ← A[i]
        k ← k + 1
        i ← i + 1
    end while
    while j ≤ der do
        B[k] ← A[j]
        k ← k + 1
        j ← j + 1
    end while
    for w in range(izq, der + 1) do
        A[w] ← B[w]
    end for
    return c
end procedure
```

iteraciones.

- *Inicialización*: Como no ha habido ninguna iteración, no se han encontrado parejas ordenadas. Por lo que $c = 0$ y ese es su valor antes de entrar al ciclo.

- *Mantenimiento*: Al inicio de la $t - ésima$ iteración, supongamos que c es la suma del número de parejas ordenadas que se pueden formar con $A[izq : mid]$ y $A[mid + 1 : der]$ de las últimas $t - 1$ iteraciones. Y tenemos los subarreglos:

$$\big[A[izq] \quad \ldots \quad A[i] \quad \ldots \quad A[mid]\big]$$

$$\big[A[mid + 1] \quad \ldots \quad A[j] \quad \ldots \quad A[der]\big]$$

  Primero haremos una observación, debido a que este algoritmo es básicamente MERGE SORT y este ya ha sido explicado en clase, daremos por hecho algunas cosas. MERGE SORT divide el arreglo original en subarreglos de tamaño 1, una vez hecho esto, MERGE va uniendo y ordenando los subarreglos. Por lo que, la entrada de MERGE siempre son subarreglos ordenados.
  Si $A[i] > A[j]$, no se modifica c pues como los subarreglos están en orden creciente los pares ordenados que incluyen a $A[j]$ ya están contemplados en c
  Si $A[i] \leq A[j]$, como los subarreglos están en orden creciente $A[i] \leq A[j] \leq \cdots \leq A[der]$. Es decir, $A[i]$ es menor o igual que todos los elementos de la posición j a la posición der. Por lo tanto hay $der - j + 1$ parejas ordenadas compuestas de i y los elementos de $A[mid + 1 : der]$.
  Como para el primer caso no se encontraron más parejas; y para el segundo caso, se tiene que $c = c + der - j + 1$; en ambos casos c es la suma del número de parejas ordenadas que se pueden formar con $A[izq : mid]$ y $A[mid + 1 : der]$ de las últimas t iteraciones.

- *Terminación*: La condición para que el **while** loop termine es que $i > mid$ o $j > mid$.
  Caso 1: $i > mid$. Sabemos que en el cuerpo del while, i sólo puede aumentar en el caso en que $A[i] \leq A[j]$ y sólo puede aumentar una unidad. Así, al incio de la última iteración, $i = mid$. Por lo que a todos los elementos del arreglo $A[izq : mid]$ ya se les contó sus parejas ordenadas con $A[mid + 1 : der]$ y si todavía quedan elementos de $A[mid + 1 : der]$, sus parejas ordenadas ya fueron contadas.
  Caso 2: $j > der$,j sólo puede aumentar en el caso en que $A[i] > A[j]$ y sólo puede aumentar una unidad. Así, al incio de la última iteración, $j = der$. Por lo que a todos los elementos del arreglo $A[mid + 1 : der]$ ya se les contó sus parejas ordenadas con $A[izq : mid]$ y si todavía quedan elementos de $A[izq : mid]$, no pueden tener parejas ordenadas porque quiere decir que son más grandes que cualquier elemento de $A[mid + 1 : der]$.
  Así, de ambos casos, el algoritmo cuenta todas las parejas ordenadas de elementos de $A[izq : mid]$ y $A[mid + 1 : der]$, y se va sumando en c. Y como MERGE regresa c, concluimos que es correcto. Por otra parte, observamos que el arreglo B se va actualizando ordenando los elementos que provienen de los dos subarreglos.

Con el algoritmo `Merge` podemos proceder con el algoritmo `Merge Sort`, que recibe un arreglo A y otro arreglo B donde se irán ordenando los elementos de A en la función `Merge`. Además, recibe dos enteros izq y der que son los índices extremos en A del subarreglo $A[izq, izq + 1, ..., der-1, der]$ que el algoritmo dividirá en dos partes, $A[izq, ..., mid]$ y $A[mid+1, ..., right]$. Hacemos recursión en cada una de las partes, donde supondremos que el algoritmo contará correctamente el número de parejas ordenadas en cada parte y tanto A como B ya están ordenados.

El algoritmo `Merge` lo que hace es contar el número de parejas ordenadas que existen, donde cada pareja toma un elemento del subarreglo ordenado $A[izq, ..., mid]$ y un elemento del subarreglo ordenado $A[mid + 1, ..., der]$, en B se ordenan ambas partes y se copia a A.

Demostración: `Merge Sort es correcto`. Al terminar el algoritmo `Merge Sort` devuelve el número de parejas ordenadas en el arreglo A que recibe y ordena a A.

Por inducción sobre la longitud n del arreglo A.

- **Caso base:** Cuando $n < 2$, el algoritmo funciona porque no hay parejas ordenadas, no entra en el ciclo del `if` y devuelve el valor de c igual a 0, y el arreglo A ya está ordenado.
- **Hipótesis de inducción:** supongamos que para arreglos de tamaño m, con $m < n$ y $n > 2$, el algoritmo devuelve el número de parejas ordenadas en el arreglo que recibe y lo ordena.
- **Paso inductivo:** El algoritmo recibe un arreglo A de tamaño $n > 2$, un arreglo B de tamaño n inicializado a 0 y con los índices 0 y n-1. Inicializa la variable c a cero que contará el número de parejas ordenadas. Procede a partir A en dos subarreglos, uno desde el índice 0 a mid y el otro desde el índice mid a n-1.

  Aplicamos el algoritmo al subarreglo izquierdo de A, con tamaño menor a A, devolviendo el número correcto de parejas ordenadas en ese subarreglo y actualizamos c, actualizamos B con los elementos ordenados del subarreglo izquierdo de A y lo copiamos a A. Luego Aplicamos el algoritmo al subarreglo derecho de A, actualizamos c al sumarle las parejas ordenadas que nos devolvió el algoritmo en la parte derecha, actualizamos B ordenando la parte derecha de B y lo copiamos a A.

  En este punto tenemos la suma de las parejas ordenadas del subarreglo izquierdo de A y del subarreglo izquierdo de A. A eso, hay que sumarle las parejas ordenadas que hay cuando se toma un elemento del subarreglo izquierdo y un elemento del subarreglo derecho de A. Ese número nos lo devuelve el algoritmo `Merge`, que se lo sumamos a c. Por lo tanto, el algoritmo `MergeSort` nos devuelve el número de parejas ordenadas de A.

Ahora,

Complejidad

Como mergeSort divide repetidamente el arreglo en subarreglos de la mitad del tamaño, hasta obtener subarreglos de tamaño 1, obtenemos un árbol binario de $log_2(n)$ niveles, en donde el nivel i tiene 2^i vértices que son subarreglos de tamaño $\frac{n}{2^i}$. merge toma dos subarreglos y recorre cada elemento de ambos subarreglos una vez y hace una cantidad constante de instrucciones, ya sea el while en el que se comparan las entradas o en el while en el que se unen los elementos restantes (en caso de que sobren). Por lo que, por cada dos vértices del árbol toma O($\frac{n}{2^i}+\frac{n}{2^i}$)=O($\frac{n}{2^{i-1}}$) y al ser 2^i vértices, se hace $\frac{2^i}{2} = 2^{i-1}$ veces; así por nivel del árbol toma O($\frac{n}{2^{i-1}} * 2^{i-1}$)=O($n$). Por último, como son $log_2(n)$ niveles, toma O($n * log_2(n)$).