Ayudantía I1

Correctitud
Complejidad
Sorting
Dividir para conquistar

Correctitud

Enunciado:

4) **Demostración de corrección**. Considera dos arreglos A y B de n elementos cada uno; ambos arreglos están ordenados. Demuestra que el siguiente algoritmo encuentra la mediana de la totalidad de los elementos de A U B en O(log n).

1) Calcula las medianas *m1* y *m2* de los arreglos ar1[] y ar2[] respectivamente.
2) If *m1* = *m2* → terminamos; return *m1* (o *m2*)
3) If *m1* > *m2*, entonces la mediana está en uno de los siguientes subarreglos:
   a) Desde el primer elemento de ar1 hasta *m1*
   b) Desde *m2* hasta el último elemento de ar2
4) If *m2* > *m1*, entonces la mediana está en uno de los siguientes subarreglos:
   a) Desde *m1* hasta el último elemento de ar1
   b) Desde el primer elemento de ar2 hasta *m2*
5) Repite los pasos anteriores hasta que el tamaño de ambos subarreglos sea 2.
6) If tamaño de ambos subarreglos es 2, entonces la mediana es:
   Mediana = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2

Complejidad

Enunciado

4) Considera el algoritmo QuickSort estudiado en clases para ordenar un arreglo A. Calcula la complejidad de la ejecución del algoritmo en cada uno de los siguientes casos:

a) Si el arreglo *A* viene ordenado (de menor a mayor) y siempre, en cada llamada a *Partition*, se elige como pivote el último elemento (el de más a la derecha) del subarreglo correspondiente.

b) Si el arreglo *A* viene ordenado (de menor a mayor) y siempre, en cada llamada a *Partition*, se elige como pivote el elemento del medio del subarreglo correspondiente.

Enunciado

Considera ahora una nueva versión del algoritmo QuickSort, llamada *QuickerSort*, para ordenar un arreglo A en que todos los elementos son distintos:

QuickerSort($A[i, f]$):
    if $i \leq f$:
        $(p, q) \leftarrow$ PartitioninThree($A[i, f]$)
        QuickerSort($A[i, p-1]$)
        QuickerSort($A[p+1, q-1]$)
        QuickerSort($A[q+1, f]$)

PartitioninThree elige dos pivotes, $A[i]$ y $A[f]$, y reorganiza el subarreglo $A[i, f]$ de modo que todos los elementos menores que el menor pivote queden en el extremo izquierdo del subarreglo, todos los elementos mayores que el mayor pivote queden en el extremo derecho del subarreglo, y todos los otros elementos queden en el subarreglo entre ambos pivotes; p y q son las posiciones finales de los pivotes al terminar la ejecución de *PartitioninThree*, tal que $p < q$ y $A[p] < A[q]$.

Enunciado

Para este algoritmo *QuickerSort*, responde:

c) Da un caso que sea lo más lento posible y argumenta por qué es así.

d) Da un caso que sea lo más rápido posible y argumenta por qué es así.

Sorting

Enunciado

1) Ordenar por dos criterios.

Como parte del proceso de postulación a las universidades chilenas que realiza el DEMRE, se cuenta con un gran volumen datos en que cada registro representa cada una de las postulaciones de cada estudiante a una carrera en una universidad. Estos registros son de la siguiente forma:

RUT_Postulante	codigo_universidad	codigo_carrera	puntaje_ponderado	…

Nota: Asume que codigo_universidad y codigo_carrera son valores numéricos enteros.

Originalmente las postulaciones se encuentran ordenadas por RUT_Postulante, y se requiere ordenarlas por codigo_universidad Y codigo_carrera para distribuir esta información por separado a cada universidad con los postulantes a cada una de sus carreras.

Preguntas

a) Propón un algoritmo para realizar el ordenamiento requerido. Puedes utilizar listas o arreglos —especifica. Usa una notación similar a la usada en clases.
b) Calcula su complejidad.
c) Determina su mejor y peor caso.

Dividir para conquistar

Enunciado

3) Estrategias algorítmicas. Sean X e Y conjuntos de datos **no** ordenados. Lo que se busca hacer es lo siguiente: Se calcula el producto cartesiano entre ambos conjuntos; o sea, todos los pares posibles con un elemento de X y otro de Y. Sea Z este nuevo conjunto.

Formalmente, producto cartesiano $Z = \{(x_i, y_i) \mid x_i \in X, y_i \in Y\}$.

Se pide ordenar dichos pares según el valor de la suma $z = x + y$. Y en caso de empates, dejar primero al que estaba primero con respecto a x. Asume que cada suma en Z es única.

a) Describe una forma de calcular las sumas de forma eficiente en un entorno altamente paralelizado
b) Dadas las sumas, describe una estructura que permita mantener los pares ordenados y la suma correspondiente
c) Finalmente, describe un algoritmo de ordenación que ordene eficientemente los pares ordenados basándose en el resultado de la suma

Hint: Recuerda estrategias algorítmicas vistas en clases.

Solución propuesta:
a,b.

```
Z_sum(arr, i,f):
  if(i==f){
    arr[i] = (arr[0]+arr[1],arr[0],arr[1]);
  }
  else if (i<f){
    m = i+(f-1)/2;
    Z_sum(arr,i,m);
    Z_sum(arr,m+1,f);
```

Solución propuesta:

c.

```
Z_sort(arr, i,f):
  if (i<f){
    m = i+(f-1)/2;
    Z_sort(arr,i,m);
    Z_sort(arr,m+1,f);
    merge(arr,i,m,f);}
```

```
merge(arr,i,m,f):
    l1= m-i+1;
    l2= f-m;
    arr_1[l1], arr_2[l2];
    for(int e=0;e<l1;e++){
        arr_1[e] = arr[i+e];}
    for(int e=0;e<l2;e++){
        arr_2[e] = arr[m+1+e];
    int a=0;
    int b=0;
    int k=i;
    while(a<l1 && b<l2){
        if(arr_1[a][0] <= arr_2[b][0]){
            arr[k] = arr_1[a];
            a++;}
        else{
            arr[k] = arr_2[b];
            b++;}
        k++;
    }
    while (a<l1){
        arr[k] = arr_1[a];
        a++;
        k++;
    }
    while (b<l2){
        arr[k] = arr_2[b];
        b++;
        k++;
    }
}
```