Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos 2'2022

Guía 1
Algoritmos de ordenación

1. Ejercicios de práctica

A continuación se presenta una selección de diversos problemas. Aquellos que han sido recogidos de fuentes, se encuentran referenciados para poder ser rastreados. Es importante notar que varios de estos problemas han sido modificados bajo el contexto del curso.

1. (**2.1-3**[CLRS09]) Considere el **problema de búsqueda**:

Input:	Secuencia de n números $A[0 \dots n-1]$ y valor v
Output:	Índice i tal que $v = A[i]$ o valor especial `NIL` si v no aparece en A

   Escriba el pseudocódigo para la **búsqueda líneal**, que resuelve el problema de búsqueda escaneando la secuencia buscando v. Demuestre la correctitud de su algoritmo.

2. (**2.1-2**[CLRS09]) Reescriba el algoritmo `InsertionSort` para ordenar de manera no creciente en lugar de no decreciente.

3. Los algoritmos `SelectionSort` e `InsertionSort` fueron vistos de forma iterativa en clases.

   a) Proponga un pseudocódigo recursivo para cada uno.

   b) Para cada uno, deduzca una relación de recurrencia para el tiempo $T(n)$ que toma ordenar n datos y obtenga la complejidad asintótica.

   c) En el mejor caso de `InsertionSort`, ¿cómo cambia el análisis?

4. (**2.2-4**[CLRS09]) ¿Cómo se puede modificar casi cualquier algoritmo para tener un buen tiempo en el mejor caso? Considere que el algoritmo efectivamente tiene un *mejor caso* que corresponde a un input con características favorables.

5. (**2-1**[CLRS09]) Aunque `MergeSort` tiene tiempo $\Theta(n \log(n))$ en el peor caso e `InsertionSort` es $\Theta(n^2)$ en el peor caso, los factores constantes en este último pueden hacerlo más rápido en la práctica para problemas pequeños. Por lo tanto, tiene sentido *podar* las hojas del árbol de recursión de `MergeSort` usando `InsertionSort` cuando los subproblemas se vuelven suficientemente pequeños.
   Considere una modificación a `MergeSort` en que n/k sub-secuencias de largo k se ordenan usando `InsertionSort` y luego son *mergeadas* usando `Merge`, donde k es un valor a optimizarse.

   a) Demuestre que `InsertionSort` puede ordenar las n/k secuencias, cada una de largo k, en tiempo $\Theta(nk)$ en el peor caso.

   b) Muestre cómo *mergear* las sub-secuencias en tiempo $\Theta(n \log(n/k))$ en el peor caso.

c) Dado que el algoritmo modificado se ejecuta en tiempo $\Theta(nk + n\log(n/k))$ en el peor caso, ¿cuál es el máximo valor de k en función de n para el cual el algoritmo modificado tiene el mismo tiempo asintótico de `MergeSort`?

d) ¿Cómo se debiera escoger k en la práctica?

6. (**2-2**[CLRS09]) Bubblesort es un algoritmo de ordenación popular, pero ineficiente. Opera intercambiando repetidamente elementos adyacentes que están desordenados.

input : Secuencia $A[0,\ldots,n-1]$
output: $\emptyset$

`BubbleSort` (A)**:**
1 **for** $i = 0 \ldots n-2$ **:**
2 **for** $j = n-1 \ldots i+1$ **:** Observe que este **for** es descendente
3 **if** $A[j] < A[j-1]$ **:**
4 $A[j] \rightleftarrows A[j-1]$

a) Sea A' la secuencia A luego de ejecutar `BubbleSort`(A). Para probar la correctitud de `BubbleSort` necesitamos probar que termian y que cumple la propiedad de orden

$$A'[0] \leq A'[1] \leq \cdots \leq A'[n-1]$$

Para probar que `BubbleSort` efectivamente ordena, ¿qué más debemos probar? *Hint:* es algo trivial de demostrar.

b) Demuestre la correctitud de `BubbleSort`.

c) ¿Cuál es la complejidad en el peor caso de `BubbleSort`? ¿Es mejor o peor que `InsertionSort`en términos asintóticos?

7. (**2-4**[CLRS09]) Considere la definición restrictiva de ***inversión***: dada una secuencia $A[0\ldots n-1]$ de n números distintos, si $i < j$ y $A[i] > A[j]$, entonces el par (i,j) se dice una **inversión** de A.

a) Liste las 5 inversiones de la secuencia 2,3,8,6,1.

b) ¿Qué secuencia de elementos del conjunto $\{1,2,\ldots,n\}$ tiene el máximo de inversiones? ¿Cuántas tiene?

c) Entregue un algoritmo que determine el número de inversiones de cualquier permutación de n elementos en tiempo $\mathcal{O}(n\log(n))$ en el peor caso. *Hint:* modifique `MergeSort`.

8. (**7.1-2**[CLRS09]) Considere el pseudocódigo de `Partition` visto en clases.

input : Arreglo $A[0,\ldots,n-1]$, índices i, f
output: Índice del pivote aleatorio en la secuencia ordenada

`Partition` (A,i,f)**:**
1 $x \leftarrow$ índice aleatorio en $\{i,\ldots,f\}$
2 $p \leftarrow A[x]$
3 $A[x] \rightleftarrows A[f]$
4 $j \leftarrow i$
5 **for** $k = i \ldots f-1$ **:**
6 **if** $A[k] < p$ **:**
7 $A[j] \rightleftarrows A[k]$
8 $j \leftarrow j+1$
9 $A[j] \rightleftarrows A[f]$
10 **return** j

¿Qué valor retorna `Partition` cuando todos los elementos de $A[0\ldots n-1]$ tienen el mismo valor? Modifique `Partition` para que en tal caso se entregue $\lfloor n/2 \rfloor$ como resultado.

9. (**7.1-4**[CLRS09]) Modifique `Quicksort` para ordenar de forma no creciente.

10. (**7.2-2**[CLRS09]) ¿Cuál es el tiempo de `Quicksort` cuando todos los elementos tienen el mismo valor?

11. En clase vimos la versión de `Partition` que escoge el pivote de forma aleatoria, no obstante, podríamos escogerlo de manera determinista. Considere una versión modificada de `Partition` que siempre toma como pivote el último elemento de la secuencia.

    a) ¿Cuál es el peor caso de ambos enfoques? ¿Es el mismo?

    b) ¿Cuál es la complejidad de peor caso en ambos casos? ¿Se gana algo escogiendo un pivote específico?

2. Preguntas de pruebas

A continuación se presentan las preguntas de la Interrogación 1 del semestre 2022-1. El objetivo es que puedan enfrentarse a preguntas tipo prueba simulando el contexto de evaluación.

1. **Ordenar por dos criterios.** Cimo parte del proceso de postulación a las universidades chilenas que realiza el DEMRE, se cuenta con un gran volumen de 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: Asuma 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.

   a) Proponga un algoritmo para realizar el ordenamiento requerido. Puede utilizar listas o arreglos –especifique. Use una notación similar a la usada en clases.

   b) Calcule su complejidad.

   c) Determine su mejor y peor caso.

   Hint: Puede referirse al valor del atributo de un elemento como `elemento.atributo`. Por ejemplo, si `P` es uno de los registros de postulaciones, entonces

   - `P.codigo_universidad` retorna el valor de `codigo_universidad` de la postulación
   - `P.codigo_carrera` retorna el valor de `codigo_carrera` de la postulación

2. **Ordenación estable.** Un algoritmo de ordenación es **estable** si al ordenar dos elementos con el mismo valor los deja en el mismo orden relativo que tenían antes de ordenarlos. Por ejemplo, considere el siguiente arreglo de nombres de personas y las edades correspondientes:

   ```
   [(yo,23),(tú,20),(él,2),(ella,10),(nosotros,25),(vosotros,15),
    (ellos,25),(ellas,30),(ustedes,10),(vosotras,5),(todos,12)]
   ```

   Si al ordenar este arreglo por edades, `ella` queda antes que `ustedes` y `nosotros` antes que `ellos`, entonces el algoritmo de ordenación es estable; en otro caso, inestable.

   a) Ejecute el algoritmo `SelectionSort` estudiado en clases sobre el arreglo anterior y muestre que es estable.

b) Sugiera una forma de hacer que `SelectionSort` sea inestable, pero mantenga su propiedad de ordenar.

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, el producto cartesiano es

$$Z = \{(x, y) \mid x \in X, y \in Y\}$$

Se pide ordenar dichos pares según el valor de la suma $z = x + y$ para el par $(x, y) \in Z$. En caso de empates, dejar primero al que estaba primero con respecto a x. Asuma que cada suma en Z es única.

a) Describa una forma de calcular las sumas de forma eficiente en un entorno altamente paralelizado.

b) Dadas las sumas, describa una estructura que permita mantener los pares ordenados y la suma correspondiente.

c) Finalmente, descibra un algoritmo de ordenación que ordene eficientemente los pares ordenados basándose en el resultado de la suma.

Hint: recuerde estrategias algorítmicas vistas en clases.

4. **Demostración de correctitud.** Considere dos arreglos A y B de n elementos cada uno, ambos ordenados. Demuestre que el siguiente algoritmo encuentra la mediana de la totalidad de los elementos de $A \cup B$ en $\mathcal{O}(\log(n))$.

(1) Calcula las medianas m_1 y m_2 de los arreglos $Ar1[]$ y $Ar2[]$ respectivamente
(2) If $m_1 = m_2$: **return** m_1
(3) If $m_1 > m_2$, entonces la mediana está en uno de los siguientes subarreglos:
  1) Desde el primer elemento de $Ar1$ hasta m_1
  2) Desde m_2 hasta el último elemento de $Ar2$
(4) If $m_2 > m_1$, entonces la mediana está en uno de los siguientes subarreglos:
  1) Desde m_1 hasta el último elemento de $Ar1$
  2) Desde el primer elemento de $Ar2$ hasta m_2
(5) Repetir los pasos anteriores hasta que el tamaño de ambos subarreglos sea 2
(6) If tamaño de ambos subarreglos es 2, entonces la mediana es

$$\frac{\text{máx}(Ar1[0], Ar2[0]) + \text{mín}(Ar1[1], Ar2[1])}{2}$$

Especificamente, demuestre -es decir, dé un argumento lógico, bien razonado- que

a) El algoritmo termina (observe que el algoritmo es iterativo, que en cada iteración el tamaño de los subarreglos disminuye, y que la condición de término de la iteración es que el tamaño de ambos subarreglos sea 2).

b) El algoritmo efectivamente encuentra la mediana en tiempo $\mathcal{O}(\log(n))$ (recuerde que dado que el número total de elementos es par, la mediana es el promedio aritmético de los dos elementos que quedan *al medio* si todos los elementos estuvieran ordenados). Con respecto a los pasos 3 y 4, basta que demuestre uno de los dos.

3. Referencias

Referencias

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.