Algoritmos y Estructuras de Datos I

Guía Práctica 9
Búsqueda y ordenamiento

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ejercicio 1.
Escribir un programa que dada una matriz cuadrada de enteros y un entero x, retorne la fila y columna que contiene al elemento x (o $\langle -1, -1 \rangle$ si no existe ese elemento). Calcular el tiempo de ejecución de peor caso.

Ejercicio 2.
Se tiene una secuencia de $N-1$ elementos enteros (con $N \geq 2$) en la cual se encuentran los valores 0 a $N-1$ ordenados de manera creciente y sin repetidos, a excepción de un único elemento faltante.

a) Escribir un programa que tome como parámetro una secuencia como la descripta y encuentre cuál es el elemento faltante. El tiempo de ejecución de peor caso del programa propuesto debe pertenecer a $\mathcal{O}(log\ N)$. Ejemplos: para $\langle 0, 1, 2, 4, 5, 6, 7 \rangle$ debe devolver 3, mientras que para $\langle 0 \rangle$ debe devolver 1.

b) Suponiendo que haya más de un elemento faltante pero no el primero ni el último. Escribir un programa que los devuelva todos. ¿Cuál es el tiempo de ejecución de peor caso?

c) Suponiendo que haya más de K elementos faltantes pero no el primero ni el último. Implementar un algoritmo que devuelva el $K-$esimo elemento que no pertenezca a dicha secuencia. ¿Cuál es el tiempo de ejecución de peor caso?

d) Suponiendo que haya como máximo $M << N$ elementos faltantes pero no el primero ni el último. Escribir un programa que devuelva todos los elementos faltantes con tiempo de ejecución de peor caso $\mathcal{O}(M\ log\ N)$.

Ejercicio 3.
Escribir dos programas que calculen la **raíz cuadrada** de un número n con tiempo de ejecución en peor caso $\mathcal{O}(\sqrt{n})$ y $\mathcal{O}(log\ n)$ respectivamente. Si n no es un cuadrado perfecto, devolver $\lfloor \sqrt{n} \rfloor$.

Ejercicio 4.
Escribir algoritmos que resuelvan cada uno de los siguientes problemas:

a) Dada una secuencia cuyos elementos son todos cero o uno, calcular la suma de los elementos de la secuencia.

b) Resolver el mismo problema que en el inciso anterior en tiempo logarítmico si se sabe que la secuencia está ordenada.

c) Resolver el mismo problema en tiempo logarítmico si se sabe que la secuencia está ordenada, y que en lugar de cero y uno, los posibles elementos de la secuencia son 15 y 22.

Ejercicio 5.
Dada una matriz de números enteros $mat \in \mathbb{Z}^{N \times M}$ con los elementos de cada fila ordenados de manera creciente y los elementos de cada columna ordenados de manera creciente. Escribir un programa que cuente la cantidad de veces que aparece un elemento dado en la matriz:

a) Suponiendo que no hay elementos repetidos en la matriz. El tiempo de ejecución debe pertenecer a $\mathcal{O}(N + M)$

b) Suponiendo que puede haber repetidos en las columnas (pero no en las filas). El tiempo de ejecución debe pertenecer a $\mathcal{O}(N + M)$

c) Suponiendo que tanto las filas como las columnas pueden tener elementos repetidos. Calcular el tiempo de ejecución de peor caso.

Ejercicio 6.
Escribir un programa para resolver los siguientes problemas que toman como entrada una secuencia de enteros v. En cada caso calcular el tiempo de ejecución de peor caso.

a) Contar la cantidad de veces que aparece un número e en v.

b) Encontrar la diferencia entre $max(v)$ y $min(v)$.

c) Encontrar el número que más veces aparece en la secuencia.

d) Resolver el item anterior en $\mathcal{O}(|v|)$, bajo la suposición de que los valores de v se encuentran acotados en el rango $[r_1, r_2]$. ¿Podrían utilizar el algoritmo anterior modificado para v no acotado? ¿Cuál sería el tiempo de ejecución de peor caso?

e) Para cada uno de los incisos anteriores, ¿Cómo podemos aumentar la eficiencia del algoritmo si suponemos v ordenado como entrada?

f) Resolver los tres primeros incisos para la secuencia $\langle 3,\ 1,\ -2,\ 0,\ 2,\ -2,\ -2,\ -2,\ 3,\ 10,\ 0,\ 4\rangle$. Si tenemos que resolver los tres ejercicios de manera consecutiva, ¿Es conveniente ordenar primero la secuencia?

Ejercicio 7.
Definimos a una secuencia de enteros como *"de cuentas"* si sus elementos aparecen tantas veces en la secuencia como su valor lo indique. Por ejemplo, $\langle 4,\ 1,\ 5,\ 5,\ 4,\ 4,\ 4,\ 5,\ 5,\ 5\rangle$ es una secuencia "de cuentas".

Escribir un programa que determine si una secuencia de enteros cualquiera s es o no "de cuentas" con tiempo de ejecución de peor caso perteneciente a $\mathcal{O}(|s|)$. Justificar por qué este algoritmo cumple con dicha complejidad.

Ejercicio 8.
Escribir un programa que devuelva la versión ordenada de un string s con costo de ejecución perteneciente a $\mathcal{O}(|s|)$. Por ejemplo, si $s =$ ``hola Homero!'', el resultado deberá ser ``!Haehlmooor''. Suponer que se cuenta con la función $ord(c)$ que dado un caracter indica su código en el standard Unicode[1].

Ejercicio 9.
Escribir un programa que, dado una secuencia de enteros v y dos enteros e y k (con $k \leq |v|$), devuelva los k números más cercanos a e. En caso de empates, considerar el de menor valor. Calcular el tiempo de ejecución de peor caso.

Ejercicio 10.
El algoritmo de ordenamiento *cocktail sort* es una variante del *selection sort* que consiste en buscar en cada iteración el máximo y el mínimo de la secuencia por ordenar, intercambiando el mínimo con i y el máximo con $|s| - i - 1$.

Escribir un programa implementando el algoritmo cocktail sort.

Ejercicio 11.
Consideremos el siguiente programa de ordenamiento, llamado ordenamiento por burbujeo (*bubble sort*):

```
int i = 0;
int j;
while (i < a.size()-1) {
    j = 0;
    while (j < a.size()-1) {
        if (a[j] > a[j+1]) {
            swap(a, j, j+1);
        }
        j++;
    }
    i++
}
```

a) Describir con palabras qué hace este programa.

b) Proponer un invariante para el ciclo principal (el más externo).

c) Proponer un invariante para el ciclo interno.

d) Calcular el tiempo de ejecución de peor caso.

Ejercicio 12.
El *bubble sort* puede ser modificado de manera que el burbujeo se realice en ambas direcciones (*cocktail shaker sort*). Una primera pasada para adelante en la lista, y una segunda pasada de regreso. Este patrón alternado se repite hasta que no sea necesario continuar.

a) Implementar un programa para el algoritmo descripto.

[1] El estandard Unicode 7.0 contiene 112956 caracteres

b) Calcular el tiempo de ejecución de peor caso.

c) Mostrar paso a paso como el algoritmo ordenaría la siguiente secuencia: $\langle 4,\ 2,\ 1,\ 5,\ 0\rangle$

Ejercicio 13.
Ordenar las siguientes secuencias utilizando los algoritmos de *ordenamiento por inserción*, *ordenamiento por selección* y *ordenamiento por burbujeo*. Para cada uno, indicar cuál (o cuáles) de los algoritmos utiliza una menor cantidad de operaciones[2] que los demás:

a) $\langle 1,\ 2,\ 3,\ 4,\ 5\rangle$

b) $\langle 5,\ 4,\ 3,\ 2,\ 1\rangle$

c) $\langle 1,\ 3,\ 5,\ 2,\ 4\rangle$

d) $\langle 1,\ 1,\ 1,\ 2,\ 2,\ 2\rangle$

e) $\langle 1,\ 2,\ 1,\ 2,\ 1,\ 2,\ 1,\ 2\rangle$

f) $\langle 1,\ 10,\ 50,\ 30,\ 25,\ 4,\ 6\rangle$

Ejercicio 14.
El algoritmo de ordenamiento *bingo sort* es una variante del *selection sort* que consiste en ubicar todas las apariciones del valor mínimo en la secuencia por ordenar, y mover todos estos valores al mismo tiempo (siendo efectivo cuando hay muchos valores repetidos).

Escribir un programa que implemente el algoritmo *bingo sort*.

Ejercicio 15. *Análisis de algoritmos.*
Decidir qué algoritmo de ordenamiento de los vistos en la materia es conveniente utilizar bajo las siguientes suposiciones. ¿Sería útil usar una versión modificada de dichos algoritmos?. Justificar:

a) Ordenar una secuencia que está ordenada.

b) Insertar k elementos en su posición en una secuencia v ordenada, con k significativamente más chico que $|v|$.

c) Encontrar los k elementos más chicos de la secuencia v, con k significativamente más chico que $|v|$.

d) Dadas dos secuencias ordenadas, devolver una secuencia que contenga sus elementos ordenados.

e) Ordenar una secuencia que está ordenada de forma decreciente.

f) Encontrar los k elementos más grandes de una secuencia v, con k significativamente más chico que $|v|$.

g) Ordenar una secuencia v en el que sus elementos están desordenados en a lo sumo k posiciones, con k significativamente más chico que $|v|$.

Ejercicio 16.
Dar un algoritmo que resuelva este problema:

`proc dosMitades` (inout a: $seq\langle\mathbb{Z}\rangle$) {
`Pre` $\{|a| \geq 2 \land (\exists i : \mathbb{Z})\big(0 \leq i < |a| \land ordenado(subseq(a, 0, i)) \land ordenado(subseq(a, i, |a|))\big)\}$
`Post` $\{ordenado(a)\}$
}

Ejercicio 17.
Escribir un programa que implemente este problema:

`proc reconstruye` (in a: $seq\langle\mathbb{Z}\rangle$, out b: $seq\langle\mathbb{Z}\rangle$) {
`Pre` $\{|a| = \sum_{i=0}^{|a|-1} a[i]\}$
`Post` {
$|a| = |b| \land ordenado(b) \land$
$(\forall i : \mathbb{Z})\big(0 \leq i < |b| \rightarrow_L 0 \leq b[i] < |a|\big) \land$
$(\forall j : \mathbb{Z})\big(0 \leq j < |a| \rightarrow_L a[j] = \#apariciones(b, j)\big)$
}
}

Ejercicio 18.
Sea n la longitud de una secuencia de enteros cuyos elementos están en el rango $[0 \ldots n]$, escribir un programa para ordenar la secuencia en forma descendente cuyo tiempo de ejecución de peor caso pertenezca a $\mathcal{O}(n)$

[2]Cuando decimos cantidad de operaciones, nos referimos a cantidad de comparaciones y asignaciones que realiza el algoritmo. En caso de ser una cantidad grande, buscamos una respuesta aproximada con respecto a la longitud de la secuencia.