PEC3: Tercera Prueba de Evaluación Continuada

Por: Álvaro Giménez Gorrís

Ejercicio 1: Conceptos básicos sobre búsqueda (10%)

Tarea: Di si son ciertas o no las sentencias siguientes. En caso de que no lo sean, especifica la razón:

I. El único algoritmo que permite hacer búsquedas eficientes en un vector no ordenado es el algoritmo de búsqueda binaria.

Falso: la búsqueda binaria se basa en descartar los elementos que sean mayores o menores que el elemento que está en posición central del vector y, por lo tanto, el vector debe de estar ordenado para que pueda funcionar

II. La recursividad indirecta ocurre cuando una función llama a una función diferente que a su vez llama a la primera.

Cierto.

III. La velocidad de ejecución de un algoritmo de búsqueda secuencial en un vector depende exclusivamente del tamaño del código.

Falso: La velocidad de ejecución de un algoritmo depende de los datos de entrada, la calidad del código generado por el compilador, la máquina donde se ejecuta el algoritmo y de la complejidad del algoritmo.

IV. Un conjunto de datos ordenado favorece la velocidad de ejecución de los algoritmos de búsqueda, de tal forma que el coste de ordenar el conjunto, así como mantenerlo ordenado es despreciable, sobre todo cuando crece el volumen de datos.

Falso: En el caso de que utilicemos un algoritmo de búsqueda secuencial, que el vector esté ordenado no favorece la velocidad de ejecución.

Ejercicio 2: Algoritmos de búsqueda (20%)

Tarea: Dado el siguiente vector, aplica el algoritmo indicado para buscar el elemento que se desea. Escribe el resultado y la secuencia de índices consultados en el proceso de búsqueda por el vector:

1	2	3	4	5	6	7	8	9	10	11	12
87	**80**	**71**	**65**	**63**	**42**	**33**	**20**	**17**	**16**	**6**	**4**

I. Aplica el algoritmo de búsqueda **lineal** o **secuencial** para encontrar el elemento 42

El elemento 42 está en la posición con índice 6 en el vector.
Índice 1: 87 ≠ 42
Índice 2: 80 ≠ 42
Índice 3: 71 ≠ 42
Índice 4: 65 ≠ 42
Índice 5: 63 ≠ 42
Índice 6: 42 = 42 Encontrado

II. Aplica el algoritmo de búsqueda **lineal** para encontrar el elemento 21.

El elemento 21 no se encuentra en el vector
Índice 1: 87 ≠ 21
Índice 2: 80 ≠ 21
Índice 3: 71 ≠ 21
Índice 4: 65 ≠ 21
Índice 5: 63 ≠ 21
Índice 6: 42 ≠ 21
Índice 7: 33 ≠ 21
Índice 8: 20 ≠ 21
Índice 9: 17 ≠ 21
Índice 10: 16 ≠ 21
Índice 11: 6 ≠ 21
Índice 12: 4 ≠ 21

III. Aplica el algoritmo de búsqueda **binaria** para encontrar el elemento 65.

Como el vector está ordenado de forma descendente, habrá que modificar el algoritmo para que, si la posición media del vector es mayor que el elemento, la variable izquierda sea igual a la variable “medio” + 1 y si no se cumple la condición, que la variable derecha sea igual a la variable “medio” – 1. El algoritmo quedaría de la siguiente manera:

```
function busquedaBinaria (v : vector of integer, elem : integer, vectorLength : integer) integer
    var
        i : integer;
        r : integer;
        middle : integer;
        founded : boolean;
    end var
    i := 1;
    r := vectorLength;
    founded := false;
    while (i <= r and not founded) do
        middle := (l + r) div 2;
        if (v[middle] = elem) then
            founded = true;
        else
            if (v[middle] > elem) then
                i := middle + 1;
            else
                r := middle – 1;
            end if
        end if
    end while
    if (founded) then
        return middle;
    else
        return -1;
    end if
end function
```

Con lo cual, la operación realizada por el algoritmo será la siguiente:
El elemento 65 está en el índice 4 del vector

I = 1, D = 12	(1 + 12) / 2 = 13 / 2 = 6	índice 6: 42 < 65
I = 1, D = 5	(1 + 5) / 2 = 6 / 2 = 3	índice 3: 71 > 65
I = 4, D = 5	(4 + 5) / 2 = 9 / 2 = 4	índice 4: 65 = 65 Encontrado

IV. Aplica el algoritmo de búsqueda **binaria** para encontrar el elemento 18.

Al igual que en el apartado anterior, modificamos el algoritmo de la misma manera. La operación realizada por el algoritmo es la siguiente:

El elemento 18 no se encuentra en el vector

I = 1, D = 12	(1 + 12) / 2 = 13 / 2 = 6	índice 6: 42 > 18
I = 7, D = 12	(7 + 12) / 2 = 19 / 2 = 9	índice 9: 17 < 18
I = 7, D = 8	(7 + 8) / 2 = 15 / 2 = 7	índice 7: 33 > 18
I = 8, D = 8	(8 + 8) / 2 = 16 / 2 = 8	índice 8: 20 > 18

Como I = D, finaliza el bucle y se demuestra que el elemento no se encuentra en el vector.

Ejercicio 3: Uso de los algoritmos de búsqueda (20%)

Tarea: Volviendo al sistema de pedidos de la tienda de instrumentos musicales, en este ejercicio se utilizarán los tipos de datos presentados a continuación y que están basados en la solución del ejercicio 1 de la PEC1.

```
type
    tStatus = {high, medium, low, none};
    tInstrument = record
    id : integer;
    description : string;
    price : real;
    availability : tStatus;
    end record

    tMusicStore = record
    catalogue: pointer to tInstrument;
```

```
        numInst: integer;
        end record;
end type
```

Asumiendo que el conjunto de instrumentos esta ordenado por código en forma ascendente, se nos pide implementar en **lenguaje algorítmico** la función **getInstInfo** que a partir de la tienda (**m**) un código de instrumento (**id**) devuelve el precio del instrumento. Si el instrumento no existe se devolverá el valor -1.0. Para implementar la búsqueda debes utilizar el **método de búsqueda binaria.**

```
function getInstInfo (m: tMusicStore, id: integer) : real
        var
                i : integer;
                r : integer;
                middle : integer;
                founded : boolean;
        end var
                i := 1;
                r := m.numInst;
                founded := false;
        while (i <= r and not founded) do
                middle := (l + r) div 2;
                        if (m.catalogue[middle] = id) then
                                founded = true;
                        else
                                if (m.catalogue[middle] < id) then
                                        i := middle + 1;
                                else
                                        r := middle – 1;
                                end if
                        end if
        end while
        if (founded) then
                return m.catalogue[middle].price;
        else
                return -1.0;
                end if
end function
```

Ejercicio 4: Conceptos básicos sobre complejidad (20%)

Tarea: Responde las preguntas siguientes:

I. ¿La complejidad de un algoritmo, en función de que se define?

La complejidad de un algoritmo se define en función del tamaño de los datos de entrada.

II. Calcula la complejidad computacional de las siguientes funciones de tiempos de ejecución usando la notación asintótica:

a. $T(n) = n_2 + n$

$O(n^2)$ cuadrática

b. $T(n) = \log (n-100)$

$O(\log(n))$ logarítmica

c. $T(n) = 200.n_8 + n_5$

$O(n^8)$ polinómica

d. $T(n) = 1 + 5n.\log(2n)$

$O(n * \log(n))$ casi-lineal

e. $T(n) = 823$

$O(1)$ constante

f. $T(n) = 4.n + 4$

$O(n)$ lineal

III. Ordena las complejidades computacionales del apartado anterior de más eficiente a menos eficiente.

El orden de más eficiente a menos eficiente es el siguiente: e, b, f, d, a, c.

IV. ¿A qué se refiere el comportamiento asintótico de un algoritmo?

El comportamiento asintótico de un algoritmo se refiere al comportamiento mostrado por el algoritmo cuando el tamaño de la entrada de datos aumenta.

Ejercicio 5: Análisis de algoritmos (30%)

Tarea: Responde a las preguntas siguientes.

I. Dada la siguiente definición del tiempo de ejecución, calcula la función *T(n)* sin que aparezca ninguna definición recursiva y explica el proceso que has seguido para obtener el resultado:

$$T(n) = \begin{cases} c_1, & \text{si } n \leq 1 \\ c_2 + T(n/2), & \text{si } n > 1 \end{cases}$$

Aplicamos la definición recursiva de forma repetida un total de i veces para ver la forma que toma la expresión resultante:

$$T\left(\frac{n}{2}\right) + C_2 = T\left(\frac{n}{4}\right) + 2C_2 = T\left(\frac{n}{8}\right) + 3C_2 = T\left(\frac{n}{16}\right) + 4C_2 \ldots$$
$$T\left(\frac{n}{2^i}\right) + iC_2$$

Después, calculo el valor de i para que de lugar el caso base:

$$\frac{n}{2^i} = 1 \rightarrow n = 2^i \rightarrow i = \log(n)$$

Finalmente, completo el cálculo cuando i se transforma en T(1) que en este caso es C_1:

$$T\left(\frac{n}{2^{\log(n)}}\right) + \log(n) * C_2 = T\left(\frac{n}{n}\right) + \log(n) * C_2 = T(1) + \log(n) * C_2$$
$$= C_1 + \log(n) * C_2$$

II. Calcula la complejidad computacional de la función **mistery** y explica el proceso que has seguido para obtener el resultado.

```
function mistery (v: vector[N] of real, x: integer, y: integer): real
Pre: { 1 ≤ x ≤ y ≤ N }
var
        res: real;
        ind: integer;
end var
        res := 0.0; (1)
        ind := x; (2)
        while (ind ≤ y) do (3)
                if v[ind] < 0.0 then (4)
                    1.  res := res + v[ind]; (5)
                end if (6)
                ind := ind + 1; (7)
        end while (8)
        return res; (9)
end function
```

La función **místery** empieza con una asignación (línea 1) que es una operación elemental con un tiempo constante, k_1.

A continuación, tenemos la línea 2 que, al igual que n la línea 1 se trata de una operación elemental con un tiempo constante k2.

Después hay un bucle (líneas 3-8) cuyo tiempo de ejecución viene dado por el tiempo constante de evaluar la condición (k3) más el tiempo de ejecución de la condición (línea 4) sumado al tiempo de ejecución de las instrucciones de la sentencia condicional (línea 5) más el tiempo constante de ejecución de la asignación y suma (línea 7) que podemos expresar con la siguiente fórmula:

$$T_{3-8}(n) = T_3(n) + T_4(n) + T_5(n) + T_7(n)$$

Como el número de iteraciones totales es n, la expresión del bucle sería la siguiente:

$$T_{3-8}(n) = k_3 + n * (k_3 + k_4 + k_5 + k_7)$$

Por lo tanto, el tiempo total de ejecución de la función se puede expresar de la siguiente forma:

$$T(n) = k_1 + k_2 + k_3 + n * (k_3 + k_4 + k_5 + k_7)$$

Finalmente, podemos afirmar que la complejidad de la función es **lineal** O(n).