Estructura de datos y de la información
Boletín de problemas - Tema 8

1. Dada la siguiente función recursiva:

```
void F(char c) {
  if (('A' <= c) && (c <= 'H') ) {
    F(c-1);
    cout << c;
  }
  else
    cout << endl;
}
```

   Indicar la salida por pantalla obtenida por las siguientes llamadas:

   a) `F('C')`

   b) `F('G')`

   c) `F('3')`

   d) `F('C')` si `c - 1` se sustituye con `c +1`.

   e) `F('C')` si se intercambia el orden de la escritura y la llamada recursiva.

   f) `F('C')` si se añade una escritura antes de la llamada recursiva.

2. Dada la siguiente función recursiva:

```
int F2(char c1, char c2) {
  if (c1 > c2 )
    return 0;
  else if (c1+1 == c2)
    return 1;
  else
    return F2(c1+1, c2-1) + 2;
}
```

   Realizar una traza de la misma con las siguientes llamadas

   a) `F2('a','e')`

   b) `F2('h','c')`

3. Dada la siguiente función recursiva:

```
void F3(char c, int n) {
  if ( n <= 0 )
    cout << endl;
  else {
    F3(c-1, n-1);
```

```
    cout << c;
    F3(c+1, n-1);
  }
}
```

Determinar la salida producida por pantalla por las siguientes llamadas

a) `F3('M', 2)`, `F3('M', 3)` y `F3('M', 4)`.

b) ¿Cuantas letras se escriben como resultado de la llamada `F3('M', 10)`?

c) Si movemos la escritura antes de la primera llamada recursiva, ¿qué salida produce la llamada `F3('M', 4)`?

4. Determinar qué calculan las siguientes funciones recursivas y cuál es su coste:

a)
```
int misterio1(int n) {
  if ( n == 0 )
    return 0;
  else
    return n * misterio1(n-1);
}
```

b)
```
int misterio2(double x, int n) {
  if ( n == 0 )
    return 0;
  else
    return n + misterio2(x, n-1);
}
```

c)
```
int misterio3( int n) {
  if ( n < 2 )
    return 0;
  else
    return 1 + misterio3(n/2);
}
```

d)
```
int misterio4( int n) {
  if ( n == 0 )
    return 0;
  else
    return misterio4(n/10) + n % 10;
}
```

e)
```
int misterio5( int n) {
  if ( n < 0 )
    return misterio5(-n);
  else if (n < 10)
```

```
    return n;
  else
    return misterio5(n/10);
}
```

5. Escribir una versión iterativa de las funciones del ejercicio anterior.

6. Escribir una función recursiva que devuelva el número de dígitos de un entero no negativo en base 10.

7. Escribir una función iterativa que resuelva el problema del ejercicio anterior.

8. Escribir una función recursiva que muestre en orden inverso (de menor a mayor peso) los dígitos de un entero no negativo en base 10.

9. Escribir una función iterativa que resuelva el problema del ejercicio anterior.

10. Escribir una función recursiva que invierta el orden de los elementos de un array de caracteres y utilice la siguiente cabecera:
    `void Invertir(char a[], int primero, int ultimo);`

11. Escribir una función recursiva que busque un carácter en un array de caracteres y devuelva su posición entre `primero` y `ultimo`. La función devolverá `-1` si no encuentra el carácter buscado. Utilizar la siguiente cabecera:
    `int Buscar(char a[], int primero, int ultimo, char buscado);`

12. Utilizar las operaciones básicas de `string` (por ejemplo, `length` o `substr`) para implementar una función recursiva que lo invierta.

13. Se define el *máximo común divisor* de dos números enteros `a` y `b`, `MCD(a,b)`, no ambos cero a la vez, como el mayor entero positivo que es divisor de ambos. El *algoritmo de Euclides* para calcularlo funciona del siguiente modo: Se divide `a` y `b` y se obtiene el cociente `q` y el resto `r`, tales que `a=bq+r` (Si `b=0`, `MCD(a,b)=a`). Dado que se cumple `MCD(a,b)=MCD(b,r)`, se sustituye `a` por `b` y `b` por `r` y se repite la operación. Como los restos obtenidos van decreciendo, eventualmente el resto valdrá `0`. El último resto distinto de cero es el valor buscado: `MCD(a.b)`.
    Nota: Si a o b son negativos, se sustituyen por su valor absoluto.
    Implementar una función recursiva que calcule MCD(a,b).

14. Implementa un algoritmo iterativo que calcule MCD(a,b) utilizando el *algoritmo de Euclides*.

15. Un número de combinaciones de n elementos de k en k puede definirse como sigue:

$$\binom{n}{k} = \begin{cases} 1 & \text{si } k = 1 \text{ o } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{si } 0 < k < n \end{cases}$$

    a) Escribir una función recursiva que calcule $\binom{n}{k}$.

b) Realizar una traza de la función para $\begin{pmatrix} 4 \\ 3 \end{pmatrix}$.

16. Implementa un algoritmo recursivo que determine si una cadena de caracteres es o no palíndroma.

17. Implementa un algoritmo recursivo que calcule x^n, siendo x un número real y n un entero. Calcular el coste del algoritmo implementado.

18. ¿Cómo podríamos aprovechar el hecho de que $x^n = x^{n/2} * x^{n/2}$ para resolver el problema del ejercicio anterior? ¿Cuál sería el coste del algoritmo recursivo resultante?
Nota: Cuando n es impar, $x^n = x * x^{n/2} * x^{n/2}$

19. Realizar una traza del algoritmo mergesort con los siguientes vectores:

    a) {13, 57, 39, 85, 70, 22, 64, 48 }
    b) {13, 57, 39, 85, 99, 70, 22, 48, 64 }
    c) {13, 22, 57, 99, 39, 64, 57, 48, 70 }
    d) {13, 22, 39, 48, 57, 64, 70, 85 }
    e) {85, 70, 65, 57, 48, 39, 22, 13 }

20. Realizar una traza del algoritmo quicksort con los siguientes vectores de caracteres:

    a) {E, A, F, D, C, B }
    b) {A, B, C, F, E, D }
    c) {F, E, D, C, B, A }
    d) {A, B, C, D, E, F }

21. Una de los vectores del ejercicio anterior demuestra la necesidad de la condición `(izq < der)` en la búsqueda desde la izquierda del algoritmo de división ¿De qué vector se trata? ¿Qué ocurre si omitimos dicha condición?

22. La función quicksort de las transparencias ordena en primer lugar el subvector izquierdo y después el derecho. Puede comprobarse que el uso de la pila del sistema es menor si se ordena en primer lugar el subvector de menor longitud. Modifica el algoritmo quicksort para que tenga en cuenta esta mejora.

23. Puede demostrarse que para ordenar vectores de pequeña dimensión es mejor usar algoritmos como la ordenación por inserción. Modificar el algoritmo quicksort para que se detenga el proceso de división cuando los subvectores sean menores que un tamaño dado por la constante `UMBRAL`. En ese punto se ordenará el subvector correspondiente usando el método de la burbuja y se iniciará el retorno de la recursión. ¿Qué problema plantea si usamos el algoritmo de ordenación por inserción con centinela en lugar del método de la burbuja?

24. Implementar una función que calcule la mediana de tres números a, b y c, es decir, el número con un valor intermedio de los tres.

25. Modificar el algoritmo de división para que el pivote seleccionado sea la mediana de tres componentes del vector elegidas aleatoriamentes ¿Cómo se modificaría el coste del algoritmo quicksort si realizamos esta modificación?

26. La mediana de un número impar de elementos es el valor que se encuentra en medio tras ordenarlos. Un algoritmo eficiente para encontrar la mediana que no requiere la ordenación previa de los elementos, se obtiene mediante la siguiente modificación del algoritmo quicksort:
El algoritmo de división permite determinar la posición del pivote en un vector de elementos. Si el pivote se encuentra en la posición central, es la mediana que buscamos. En otro caso, el subvector de mayor dimensión contiene la mediana, y podemos encontrarla aplicando recursivamente el mismo esquema. Implementar un algoritmo recursivo que calcule la mediana de un vector de elementos utilizando este método.

27. El algoritmo recursivo del ejercicio anterior puede modificarse fácilmente para encontrar el k-ésimo elemento más pequeño de un vector. Implementar dicha función recursiva.