Universidad Simón Bolívar
Departamento de Computación
y Tecnología de la Información
Laboratorio de Computación II
Ene-Mar 2019

> 124. *¿Qué es querer?*—Nos reímos del que sale al umbral de su puerta en el momento en el que el sol asoma al de la suya, y dice: "Quiero que salga el sol"; y de aquel que no pudiendo detener una rueda exclama: "Quiero que ruede"; y de aquel que derribado al suelo en la lucha, dice: "Estoy en el suelo porque quiero." Pero, bromas aparte, ¿nos conducimos alguna vez de diferente modo que esos tres hombres, cuando decimos: quiero?
>
> Aurora, Federico Nietzsche.

Lab. 2—Sem. 3—Algoritmos Recursivos, Inserción y Mezcla.

En este laboratorio tenemos varias tareas que cumplir, a saber:

1. Entender cómo se escribe y cuándo es conveniente usar un algoritmo recursivo.
2. Codificar algunos algoritmos recursivos. En particular codificar el algoritmo de búsqueda binaria recursivamente.
3. Escribir versiones iterativas y recursivas de ciertos problemas.
4. Usar dichos algoritmos para resolver problemas concretos.
5. Escribir y probar el algoritmo de búsqueda binaria recursivo e iterativo.
6. Escribir y probar el algoritmo de inserción ordenada: recursivo e iterativo.
7. Escribir y probar el algoritmo de mezcla.

Tareas

1. *Mis primeros programas recursivos.* Una función $f(x)$ se llama recursiva cuando usa valores anteriores de sí misma para calcular su valor para un x determinado. Básicamente una función recursiva consta de uno o varios valores básicos para los cuales se conoce la función y de una o varias fórmulas que permiten calcular el valor de $f(x)$. Un ejemplo de una definición recursiva es:

$$n! = \begin{cases} 1 & \text{si } n = 0, \\ n(n-1)! & \text{si } n > 0. \end{cases}$$

   a) Esto se transcribe en C como sigue:

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

   Si se desea un código no recursivo, basta con declarar una variable r para acumular el producto y otra para controlar el ciclo:

```
int fac(int n){
    int r = 1, k = 0;
    while (k<n) { r = r*(k+1); k++;} // o while (k<n) r = r*(++k);
    return r;
}
```

   Escriba un programa que permita chequear estos códigos. (Tiempo estimado: 10min.) Consulte lo que no entienda.

   b) *Máximo Común Divisor.* El máximo común divisor de dos enteros no negativos y no ambos nulos se puede determinar usando la siguiente fórmula recursiva.

$$\operatorname{mcd}(a,b) = \begin{cases} a & \text{si } b = 0, \\ \operatorname{mcd}(b, a \bmod b) & \text{si } b \neq 0 \end{cases}$$

Escriba dos funciones: una iterativa y una recursiva que permitan calcular el máximo común divisor positivo de dos enteros no ambos nulos a, b. Sus nombres deben ser mcd y $mcdR$ respectivamente. Además, si el tiempo lo permite resuelva los siguientes ejercicios:

1) Escriba un programa que lea un par de enteros a, b, con b no nulo que representan una fracción, y dé como salida la fracción irreducible correspondiente.
2) Use que $\text{mcd}(a,b)\,\text{mcm}(a,b) = ab$ para escribir una función que permita calcular el mínimo común múltiplo entre dos enteros no negativos a, b no ambos nulos.

2. *Búsqueda Binaria Recursiva.*

a) Haga un **NUEVO** programa que permita probar el algoritmo de búsqueda binaria recursivo visto en clase: El algoritmo esta vez debe tener como interfaz: `void bbR(int E, int x, int y, int a[])` donde $[x, y)$ representa el intervalo cerrado-abierto de búsqueda, y E el elemento que se quiere buscar en el arreglo ordenado no decrecientemente a. Recuerde que el código iterativo es:

```
int bb(int E, int n, int a[]){
   int x = 0, y = n, m;
   while (x+1 != y){
      m = (x+y)/2;
      if (E < a[m]) y = m; else x = m;
   }
   return a[x] == E;
}
```

y que por lo tanto el programa que se quiere consiste básicamente de un if con el caso base cuyo else es el caso recursivo. Nota: Tal vez conviene que implemente ambas versiones en el mismo archivo y las pruebe.

Completar:

```
//Busca en el intervalo de índices enteros [x,y)
int bbR(int E, int x, int y, int a[]){
   int m;
   if (x+1 != y){
      m = (x+y)/2;

            COMPLETAR;
   }
   return COMPLETAR;
}
```

Además resuelva los siguientes problemas:

b) Modifique el algoritmo de búsqueda binaria **recursiva** que obtuvo antes para que devuelva el lugar donde lo encontró o n si no lo encontró.

3. *El Método de Bisección.* El método de bisección es un algoritmo que permite hallar los valores reales donde se anula una función continua en un intervalo cerrado $[a, b]$. Se basa en el Teorema del valor intermedio que establece que: "toda función continua en un intervalo cerrado $[a, b]$ toma todos los valores que se hallan entre $f(a)$ y $f(b)$." En otras palabras: todo valor entre $f(a)$ y $f(b)$ es imagen de algún número en $[a, b]$. Luego, como caso particular: *Si* $f(a)$ *y* $f(b)$ *tienen signo contrario, entonces existe* $x \in [a, b]$ *tal que* $f(x) = 0$.

El algoritmo consiste en evaluar a la función en el punto medio del intervalo $m = (a + b)/2$, si da cero hemos terminado, de lo contrario debemos evaluar que mitad del intervalo descartar. Si $f(a) * f(m) < 0$, la raíz está $[a, m]$ y debemos cambiar a b por m, de lo contrario si $f(a) * f(m) > 0$ debemos cambiar a a por m en caso contrario m era la raíz.

Use el esquema de búsqueda binaria para implementar este método completando el siguiente esqueleto.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

float f(float x){
   return x*x - 2; // Cambiar esto para probar con otra función...
}

///Pre: f(a)*f(b) < 0, a < b, er > 0 cecano a CERO
float biseccion( float a, float b, float er){
   float m;
   while ( COMPLETAR ){
      m = (b+a)/2;
      COMPLETAR;
    }
    return COMPLETAR;
}

int main(){
   float a = 1.0, b = 2.0;              //Cambie el intervalo de búsqueda ...??
   float r = biseccion(a,b,.000001);
   printf("\nPrueba del Metodo de Biseccion:\n");
   printf("\n%9.6f es una raiz de la FUNCION f.",r);
   return 0;
}
```

4. *Inserción Ordenada.* El problema de inserción ordenada en un arreglo consiste en agregar un nuevo elemento a un arreglo cuyos elementos están ordenados de forma no decreciente de tal manera que el nuevo arreglo siga ordenado. Se le pide que lleve a cabo esta tarea con un algoritmo recursivo y con uno iterativo.

```
viod insertOrdR(int x, int a[], int n){
   if(n == 0 || a[n-1] <= x) a[n] = x;
   else {
      Completa: mover a a[n-1] e invocar recursivamente.

   }
}

void insertOrdI(int x, int a[], int n){
   int i = n;
   //if(n == 0) a[n] = x; // No es necesario... por qué?
   while(i > 0 && x < a[i-1]){ //Note que es una búsqueda lineal acotada
      a[i] = a[i - 1];              //en la cual se hace algo mientras se avanza.
      i--;
   }
   a[i] = x;
}
```

Con el fin de probar sus códigos partiendo de $n = 0$ haga una secuencia de inserciones ordenada sobre un arreglo y luego emprima su contenido.

5. *El Algoritmo de Mezcla.* El algoritmo de mezcla consiste en combinar dos arreglos ordenados para producir un nuevo arreglo ordenado. Copie, complete, comente y pruebe el siguiente cógigo.

```
void merge(int a[], int b[], c[], int m, int n){
   int i = 0, j = 0, k = 0;
   while(i < m && j < n) if(a[i] <= b[j]) c[k++] = a[i++]; else c[k++] = b[j++];
   while(i < m) c[k++] = a[i++];
   while(j < n) Completar...;
}
```

Con el fin de probar este código, lea dos arreglos que tienen que llamarse a y b, ordene dichos arreglos, invoque el algoritmo de mezcla, y luego imprima el arreglo en el cual dejó el resultado de la mezcla—no tiene que llamarse c. Para ordenar use el algoritmo de inserción dado en clases.

Nota: Se recomienda seriamente que no se hagan *cut and paste*. Debe copiar su código para que lo entienda mejor!!!