UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERIAS
DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

MATERIA:
ALGORITMIA

MAESTRA:
DAVID ALEJANDRO GOMEZ ANAYA

TITULO DE INVESTIGACIÓN:
Actividad Posterior 7: Recursividad y Operaciones Barometricas

FECHA ENTREGA:
DOMINGO 20 DE MAYO 2018

ALUMNO:
FELIPE DE JESUS RUIZ GARCIA

CODIGO:
214522077

CARRERA: INGENIERIA INFORMATICA (INNI)

SECCION: D10

CALIFICACIÓN Y OBSERVACIONES:

Requerimientos:
Para cada ejercicio defina
- ***a)*** *las operaciones barométricas,*
- ***b)*** *la regla de recursividad*
- ***c)*** *y la eficiencia del algoritmo.*

Problema 1 *)*
Diseñe un algoritmo recursivo que permita hacer la división por restas sucesivas.

Problema 2)
Diseñe, forma recursiva, una función que reciba una cadena de caracteres y devuelva la cadena con las posiciones de los caracteres invertidos, si la cadena es un palíndromo, la entrada y la salida coincidirán.

Problema 3)
Implemente, de forma recursiva, una función que devuelva el máximo común divisor de dos números enteros utilizando los pasos sugeridos. Dados dos números enteros positivos m y n, tal que m > n, para encontrar su máximo común divisor (es decir, el mayor entero positivo que divide
a ambos):
- Dividir m entre n para obtener el resto r.
- Si r = 0, el MCD es n.
- Si no, el máximo común divisor es MCD(n,r).

LOS CODIGOS ESTAN DISPONIBLES EN MI REPOSITORIO:
https://github.com/CUCEI-TAREAS/ALGORITMIA-2018A

compile with:
gcc version 7.3.1 20180303 (Red Hat 7.3.1-5) (GCC)

Problema 1 *)*
Diseñe un algoritmo recursivo que permita hacer la división por restas sucesivas.

NOTA: Bien, desarrolle el algoritmo y lo codifique en lenguaje C.
Cabe destacar que el algoritmo no toma en cuenta los numeros racionales ; por ejemplo la division 10/3 el resultado sera 3 y no 3.333 ...

```c
#include <stdio.h>

int divisionRecursiva(int, int, int);

int main(){

        int resultado = divisionRecursiva(10, 3, 0);
        printf("El resultado de la division recursiva es %d", resultado);
        return 0;
}

int divisionRecursiva(int dividendo, int divisor, int resultado){

        if ( dividendo < divisor ){
                return  resultado;
        }
        else {
                return divisionRecursiva(dividendo - divisor, divisor, resultado + 1);
        }
}
```

a) Podemos observar que la que la condicion de paro del algortimo recursivo es :
* si el dividendo es menor que el divisor, ya no puedo restar otra unidad del divisor, entonces retorna todas las veces que pude restar el divisor.
* De lo contrario, si el dividendo es mayor o igual al divisor, se le resta el divisor al dividendo, y se suma al resultado la cual indica que fue posible realizar una resta.

Por ello, defino como operaciones barometricas :
1) a la comparativa de dividendo < divisor , linea 14
2) a la llamada recursiva del algoritmo, linea 18

b) La regla de recursividad esta denotada por:

$$T(n) \stackrel{\text{def}}{=} \begin{Bmatrix} n<divisor \Rightarrow 1 \\ n \geq divisor \Rightarrow 1+T(n-divisor) \end{Bmatrix}$$

c) En el peor de los casos, el tamano del divisor seria tan pequeno como la unidad y esto haria que se ejecutara el algoritmo n veces. Por ejemplo 10/10 = 1, la T(n) seria T(n) = n +1
Siendo es evidente que:

$$T(n)=O(n)$$

Problema 2)

Diseñe, forma recursiva, una función que reciba una cadena de caracteres y devuelva la cadena con las posiciones de los caracteres invertidos, si la cadena es un palíndromo, la entrada y la salida coincidirán.

NOTA: Bien, desarrolle el algoritmo y lo codifique en lenguaje C.

```c
#include<string.h>
#include<stdio.h>

void invertirCadena(char[], char[], int, int);

int main(){

        char cadena_original[] = "felipe";
        int numero_caracteres = sizeof(cadena_original) / sizeof(char);

        printf("%s \n", cadena_original);
        printf(" el tamanio del arreglo es de %d elementos\n \n", --numero_caracteres);

        char cadena_invertida[numero_caracteres];

        invertirCadena(cadena_original, cadena_invertida, 0, numero_caracteres);
        printf("%s \n", cadena_original);
        printf("%s \n", cadena_invertida);
        return 0;
}

void invertirCadena(char normal[], char invertido[], int pos, int size){

        if(pos == size ){
                invertido[size + 1] =  '\0';
                return;
        } else {
                printf("%c", invertido[size - pos]); // current char
                invertido[size - pos] = normal[pos];
                invertirCadena(normal, invertido, pos + 1, size);

        }
}
```

la T(n) seria T(n) = n +1

a) Podemos observar que la que la condicion de paro del algortimo recursivo es :
Definimos como operaciones barometricas, la condicion de la linea 24 y la recursividad de la linea 30.

b) La regla de recursividad esta denotada por:

$$T(n) \stackrel{\text{def}}{=} \begin{cases} n = elementos \Rightarrow 1 \\ n \geq elementos \Rightarrow 1 + T(n-1) \end{cases}$$

c) El numero de llamadas recursivas sera siempre n. la T(n) seria T(n) = n +1.

$$T(n) = O(n)$$

Problema 3)

Implemente, de forma recursiva, una función que devuelva el máximo común divisor de dos números enteros utilizando los pasos sugeridos. Dados dos números enteros positivos m y n, tal que m > n, para encontrar su máximo común divisor (es decir, el mayor entero positivo que divide a ambos):

- Dividir m entre n para obtener el resto r.

- Si r = 0, el MCD es n.

- Si no, el máximo común divisor es MCD(n,r).

```
1 #include<stdio.h>
2
3 int mcd( int m, int n){
4
5       int r =  m % n;
6
7       if ( r == 0 ){
8             return n;
9       } else {
10             return mcd(n, r);
11       }
12 }
13 int main(){
14
15       int test = 0;
16       test = mcd(10, 5);
17       printf("%d", test);
18       return 0;
19 }
20
```

a) Podemos observar que la que la condicion de paro del algortimo recursivo es : Definimos entonces como operaciones barometricas, la condicion de **la linea 7** y la **recursividad de la linea 10.**

b) La regla de recursividad esta denotada por:

$$T(n) \overset{\text{def}}{=} \begin{Bmatrix} r = 0 \Rightarrow 1 \\ n \geq elementos \Rightarrow 1 + T(n-1) \end{Bmatrix}$$

c) El numero de llamadas recursivas sera siempre n.

T(n) = n + 1

$$T(n) = O(n)$$