Hablemos de las prácticas

- Viernes de 11:00 a 12:**30**
- Se debe de terminare en la hora y media de laboratorio.
- Si no acaban se obtiene 50% de la calificación, y 50 % al entregar el mismo viernes a mas tardar 11:59 pm.
- Se evaluará la correcta codificación (comentarios, indentación, modularidad, nombres de variables).
- De preferencia se usarán las máquinas de laboratorio.

Introducción al análisis de algoritmos

MAT-151

Dr. Alonso Ramirez Manzanares
CIMAT A.C

e-mail: alram@cimat.mx
web: **www.cimat.mx/~alram/comp_algo/**

Algoritmos

- métodos **bien definidos** para resolver problemas. Deben ser finitos, escritos en forma adecuada para ser implementados como un programa de computadora.

- un **algoritmo** es un procedimiento computacional que toma un valor, o conjunto de valores como **entrada** y produce un valor o secuencia de valores como **salida**. Es la secuencia de pasos computacionales que transforman la entrada en la salida.

  Por ejemplo, entrada: $<a_1,a_2,...,a_N>$ y la salida es una permutación $(a'_1,a'_2,...,a'_N)$ tal que $a'_1 <= a'_2 <= ... <= a'_N$

  Ejemplo, Algoritmo de Euclides para determinar el máximo común divisor:

  1. Dados dos segmentos *AB* y *CD* (con *AB*>*CD*), restamos *CD* de *AB* tantas veces como sea posible. Si no hay residuo, entonces *CD* es la máxima medida común.
  2. Si se obtiene un residuo *EF*, éste es menor que *CD* y podemos repetir el proceso: restamos *EF* tantas veces como sea posible de *CD*. Si al final no queda un residuo, *EF* es la medida común. En caso contrario obtenemos un nuevo residuo *GH* menor a EF.
  3. El proceso se repite hasta que en algún momento no se obtiene residuo. Entonces el último residuo obtenido es la mayor medida común.

- muchos algoritmos de interés involucran métodos para organizar los datos involucrados en los cálculos. Los objetos creados de esta forma se llaman **estructuras de datos**.

Algoritmo de Euclides gráfico

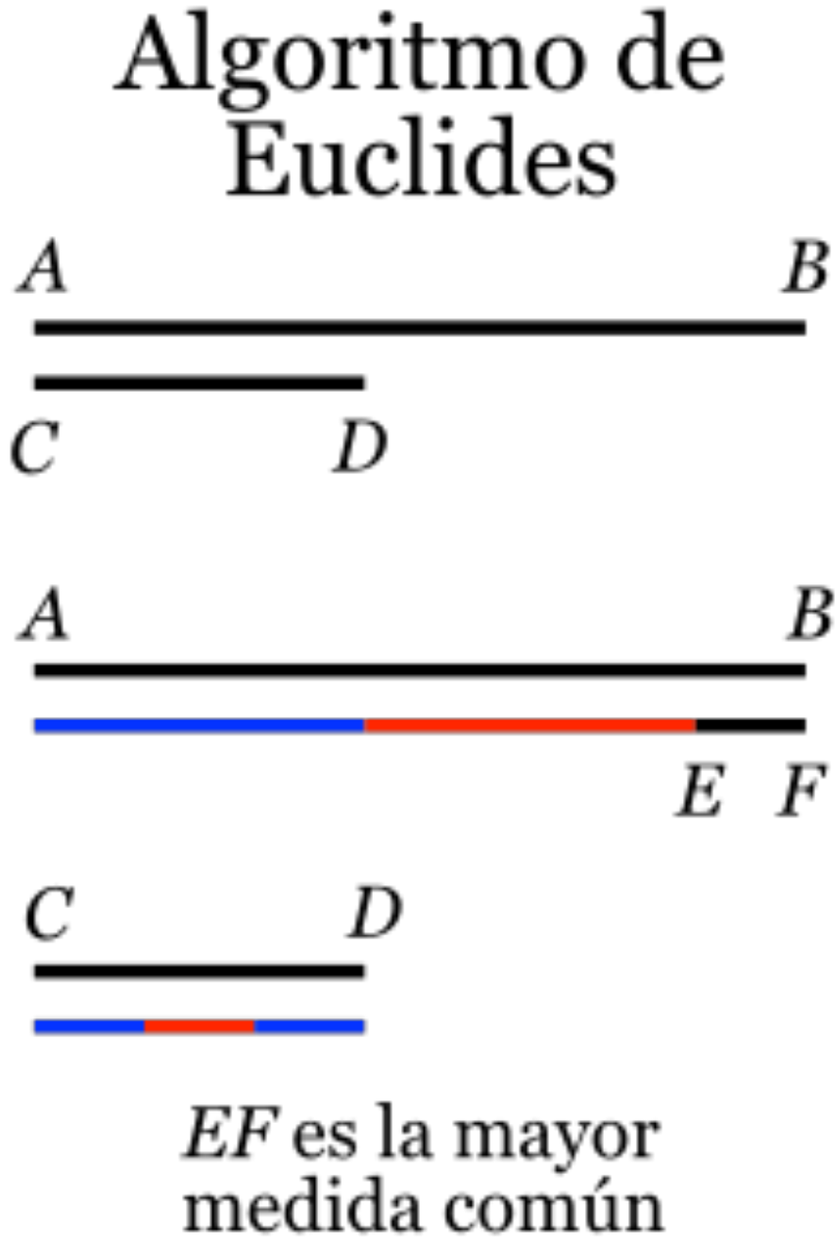


Diagrama de flujo de algoritmo

(Algoritmo para cambiar una lámpara)

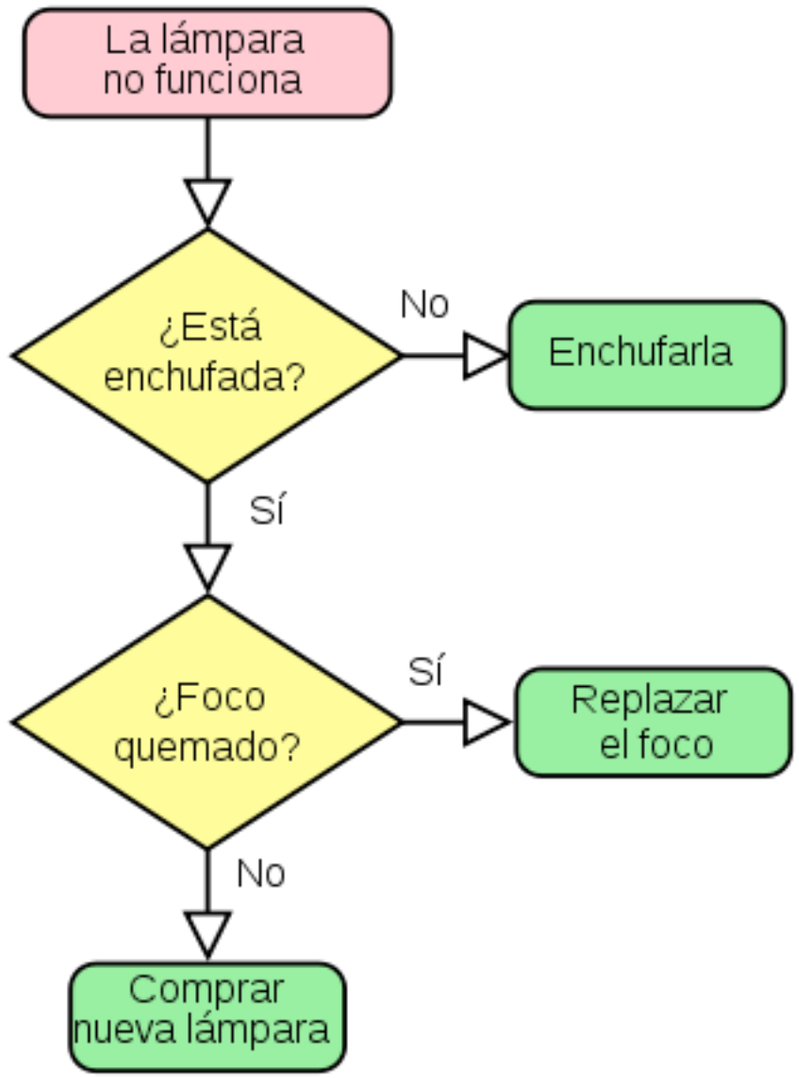
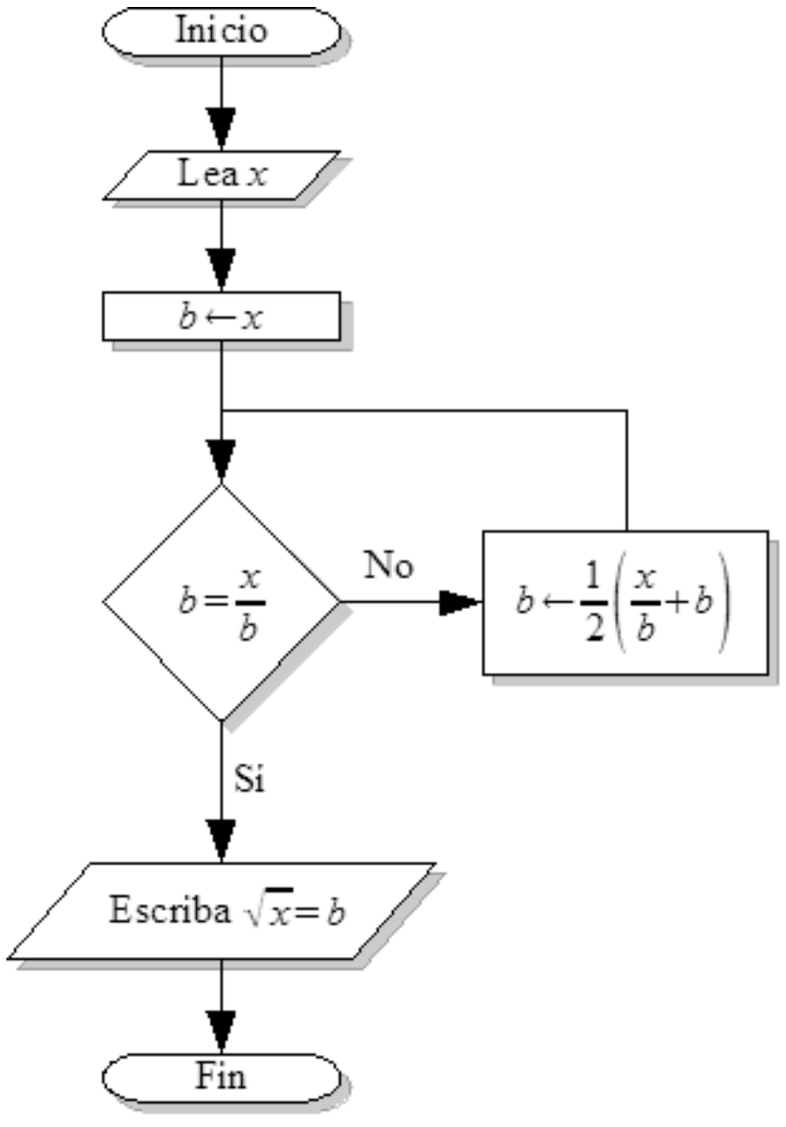


Diagrama de flujo de algoritmo

(Algoritmo de Babylonia para calcular la raíz cuadrada)



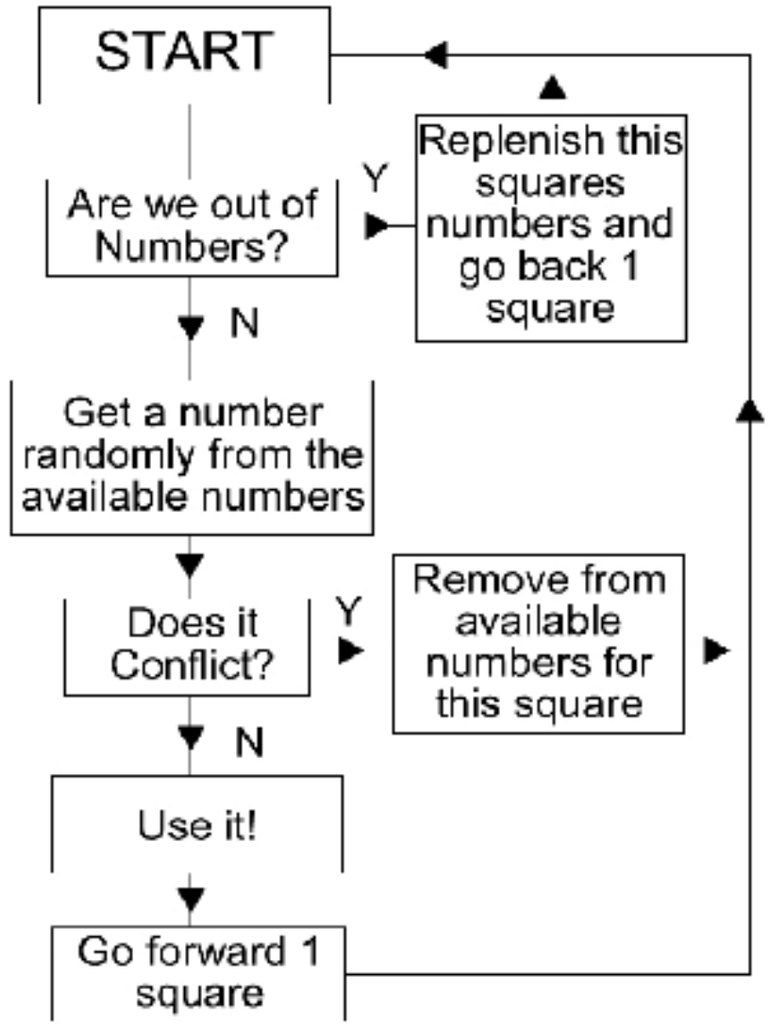
$$x_0 \approx \sqrt{S}.$$

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right),$$

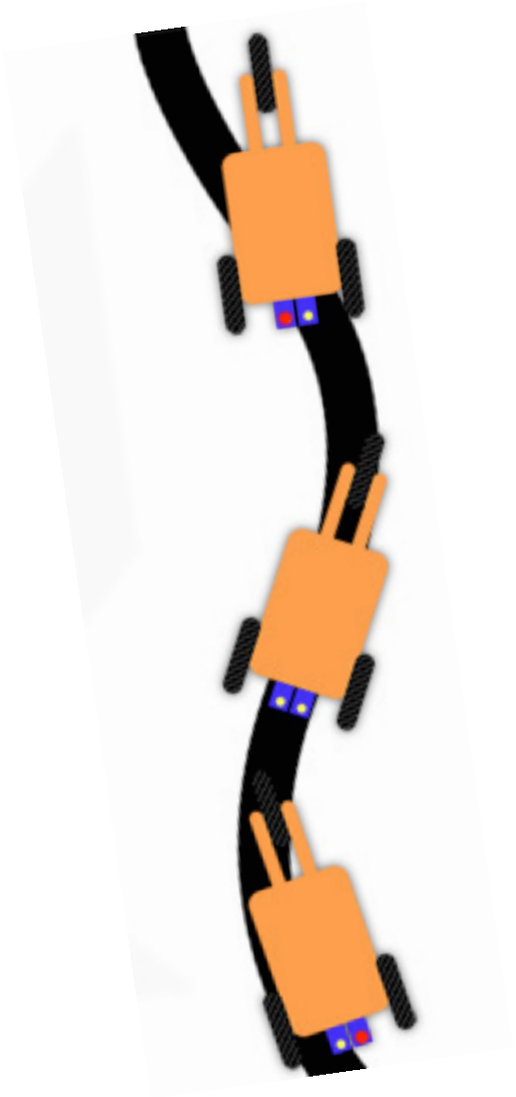
$$\sqrt{S} = \lim_{n \to \infty} x_n.$$

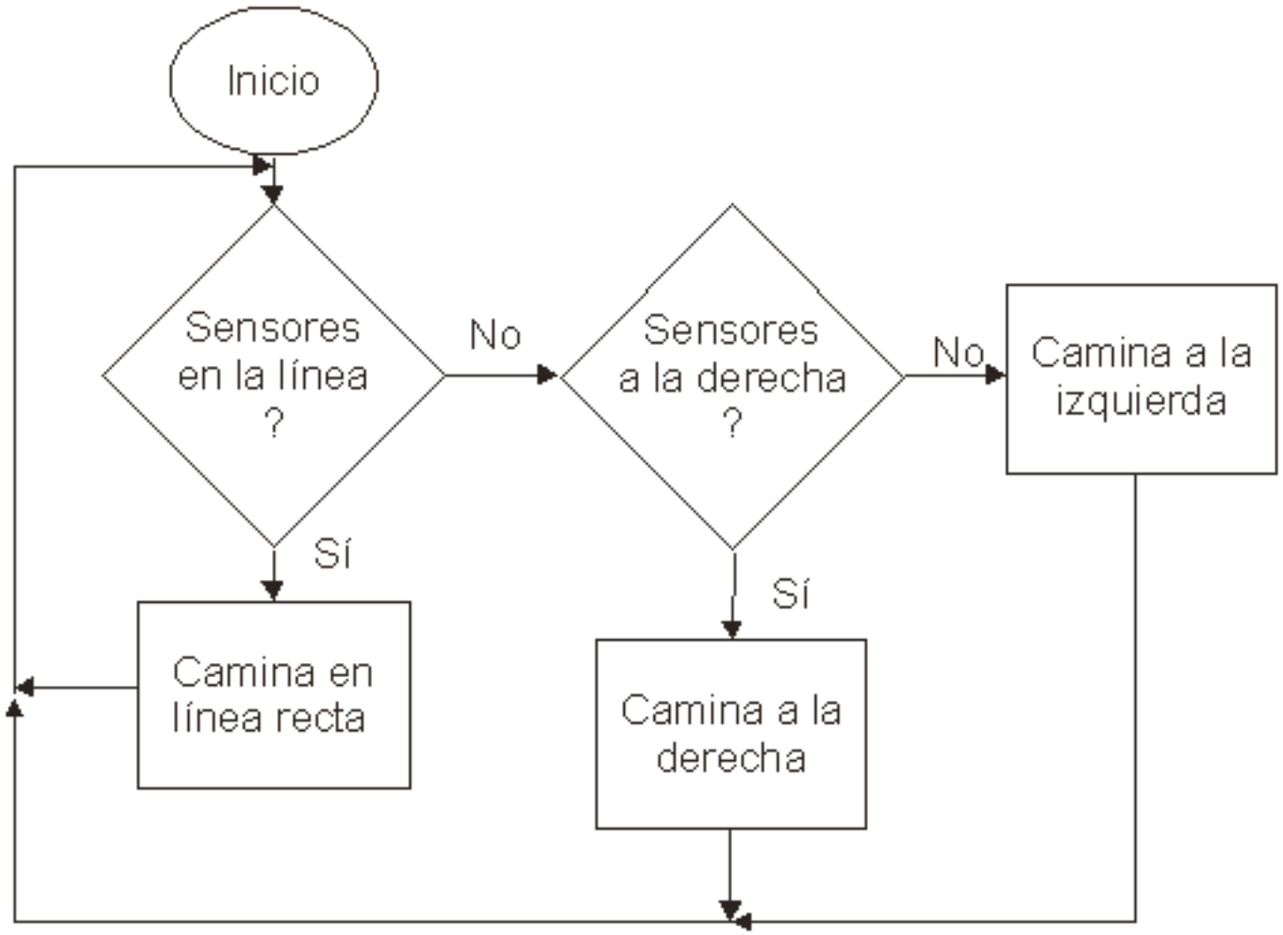
(Algoritmo para llenar un tablero de sudoku)

8	3	5	4	1	6	9	2	7
2	9	6	8	5	7	4	3	1
4	1	7	2	9	3	6	5	8
5	6	9	1	3	4	7	8	2
1	2	3	6	7	8	5	4	9
7	4	8	5	2	9	1	6	3
6	5	2	7	8	1	3	9	4
9	8	1	3	4	5	2	7	6
3	7	4	9	6	2	8	1	5



Algoritmo de un seguidor de lineas





Algoritmos

- Concepto similar a *receta, proceso, método, técnica, procedimiento* o *rutina*. Es un conjunto **finito** de reglas que da una secuencia de operaciones para resolver un problema específico. Tiene 5 características importantes.

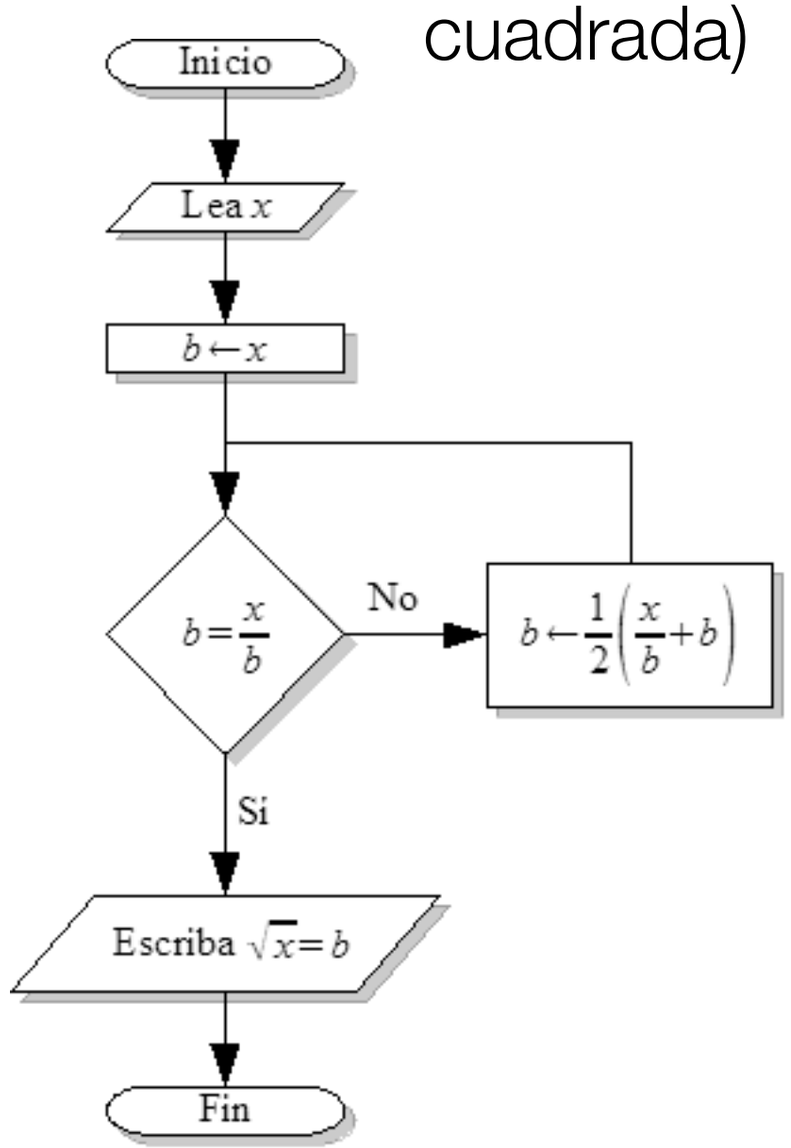
    - **finito**
    - **bien definido (debe de estar definido para todos los casos)**
    - **entradas definidas**
    - **salidas, factibles para todas las posibles entradas**
    - **factible**

Ejemplo de bien definido

Ejemplos de programación

- Calculo de la raíz cuadrada

(Algoritmo para calcular la raíz cuadrada)



GNU nano 2.2.6 File: square.cpp Modified

```
#include <iostream>
#include <cmath>

using namespace std;


int main(int narg, char * args[]){

        double x,b;


        cout << "dame el valor de x ";
        cin >> x;

        b = x;

        while( fabs(b- x/b)> 1e-6){
                b = 0.5*(x/b +b);
                cout  << endl <<  b;
        }

        cout << endl << "la raiz de " << x << " es " << b << endl << endl;

        return 0;
}
```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

```
  GNU nano 2.2.6                File: square.cpp                      Modified

#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;

int main(int narg, char * args[]){

        double x,b;

        if(narg < 2){
                cout << "\n Error, introduce el numero como parametro\n";
                return 1;
        }
        x = atof(args[1]);

        b = x;

        while( fabs(b- x/b)> 1e-6){
                b = 0.5*(x/b +b);
                cout  << endl <<  b;
        }

        cout << endl << "la raiz de " << x << " es " << b << endl << endl;

        return 0;
}
                              [ Read 27 lines ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

```
alram@teolt:~/tempPrograms/squareroot$ ./a.out
dame el valor de x 9.5

5.25
3.52976
3.11058
3.08234
3.08221
la raiz de 9.5 es 3.08221
```

```
alram@teolt:~/tempPrograms/squareroot$ ./a.out

 Error, introduce el numero como parametro
alram@teolt:~/tempPrograms/squareroot$ ./a.out 9.5

5.25
3.52976
3.11058
3.08234
3.08221
la raiz de 9.5 es 3.08221
```

Algoritmos

- no solo queremos algoritmos, queremos **buenos** algoritmos.
  - **tiempo** que toma su ejecución.
  - **uso de recursos** de la máquina (memoria, disco duro, cpu)
  - su **adaptabilidad** a diferentes computadoras.

- cuando tenemos varios algoritmos para resolver el mismo problema, debemos decidir cuál es el mejor. Esto nos lleva al campo del **análisis de algoritmos**: dado un algoritmo, el problema es determinar sus características de **desempeño**.

Análisis de algoritmos

- El estudio teórico del desempeño y utilización de recursos de un programa computacional.
- ¿Qué puede ser más importante que el desempeño?

– modularidad
– que sea mantenible
– funcionalidad
– robustez
– seguridad

– amigable al usuario (que sea claro)
– tiempo de programador
– simplicidad
– extensibilidad
– confiabilidad
– ...

Entonces... ¿para que estudiar el desempeño?

- Nos ayuda a entender la **escalabilidad** de un algoritmo.
- Nos traza una línea entre lo que es **realizable** y lo que es **imposible**.
- Las matemáticas algorítmicas proporcionan un **lenguaje** para hablar sobre el comportamiento de un programa.
- El desempeño es el tipo de **moneda** de la computación.
- Lo que se aprende acerca del desempeño se puede generalizar a otros recursos computacionales.
- Nos ayuda a **comparar** algoritmos para realizar la misma tarea.
- Nos permite establecer los valores de los parámetros del algoritmo.

“El poder de cómputo se duplica cada dos años”

Gordon Moore, co-fundador de Intel. 1965

Ley de Moore

- En realidad, hasta hace algunos años, no era cada dos años sino cada 18 meses.
- Como ejemplo, el último Sony Playstation rebasa completamente a la super-computadora más rápida del principio de los noventas.

¿Y por qué no solo comprar una mejor computadora?

- un algoritmo bien diseñado puede disminuir el tiempo de cálculo millones de veces, una computadora mejor puede acelerar en el orden de 10 o 100 veces.
- la miniaturización de los procesadores tiene un límite, confiar en la constante reducción del tiempo de cálculo es como confiar en la abundancia del petróleo.
- A) Imaginemos una nueva computadora 10 veces mas rápida y con capacidad de almacenamiento 10 veces mas grande, entonces antes procesábamos N y ahora queremos procesar 10N.
- B) Supongamos que tenemos un problema con complejidad cuadrática, entonces antes nos tardábamos N^2 unidades de tiempo.
- Entonces, la nueva computadora se va a tardar 10 veces mas en resolver el problema nuevo que lo que se va a tardar la viejita en resolver el problema viejo, porque:

$$(10N)^2/10 = 10N^2 \text{ unidades de tiempo}$$