# Temario

- Conocimientos previos
- Necesidad de un lenguaje común
- Análisis de algoritmo, conceptos

# Capítulo 1 - Introducción

- Principio DRY (Don't repeat yourself)
  - Si hay alguien que ya lo ha hecho, no repetir.
  - Ya está probado, y hay una comunidad de usuarios que lo utilizan.
  - ¿Lo podemos hacer mejor?
  - Race conditions.

# ¿Sabemos hacer un huevo frito?

Analogía

- Arrays
- Funciones
- DRY

# Algoritmo

> Conjunto finito y ordenado de instrucciones para resolver un problema dado

En definitiva: una receta

# Ejemplos

- Calcular el mayor elemento de un array
- Ordenar una serie de números
- Calcular el camino óptimo para llegar del punto A al B...

# Qué se le debe pedir:

- Correcto
- Eficaz
- Fácil de entender
- Haga uso eficiente de los recursos

# Método de trabajo

- Se diseña un algoritmo sencillo
- Se hacen simulaciones y mediciones
- Se plantea un algoritmo más eficiente
- Análisis de mejora

# Porqué estudiar algoritmos:

- Importante para todas las ramas de la computación
    i. Redes (optimización de caminos)
    ii. Diseño (algoritmos de geometría)
    iii. Bases de datos (b-trees)
    iv. Criptografía
    v. ...
- Importante en la innovación tecnológica
- Nos hace más inteligentes (otra forma de resolver problemas)

# Porqué estudiar algoritmos:

- Mejora nuestras habilidades Matemáticas
- Te ayuda a posibles entrevistas
- Comienzas a pensar de "Algortímicamente"
- Nos hace mejores programadores

# Demostración

- Page Rank
- Apple compra SnappyLabs, la empresa creadora de SnappyCam (5-1-2014)

# Corrección (correspondería a otra asignatura)

- Componente matemático muy fuerte
- Se hace a través de:
    i. Precondiciones
    ii. Postcondiciones
    iii. Invariantes

# Ejemplo

Calcular el mayor elemento de un array

```
int max(int n,const int a[n]){
        int m = a[0];
        int i =1;
        while(i != n){
                if(m < a[i])
                m = a[i];
                ++i;
        }
        return m;
}
```

# Demostración

el programa es correcto:

```
int max(int n,const int * a){
        int m = a[0];
        // m es el mayor número en el subarray a[0...0]
        int i =1;
        while(i != n){
                // m es el mayor número en el subarray a[0...i-1]
                if(m < a[i])
                        m = a[i];
                        // m es el mayor número en el subarray a[0...i]
                ++i;
                // m es el mayor número en el subarray a[0...i-1]
        }
        // m es el mayor número en el subarray a[0...i-1], & i==n
        return m;
}
```

# Análisis de un programa

Frente al rendimiento

- Memoria
- Tiempo de ejecución
- Otros recursos

# Otros análisis

- Los recursos pueden estar fijados
  1. ¿Tamaño máximo que puedo resolver en T secs.?
  2. ¿Tamaño máximo con M megs?
- Referencias:
  Cyberpunk 2077

# Tiempo de ejecución de un programa

¿Cómo se puede calcular?

> Contando instrucciones y ponderando por el tiempo de ejecución de cada instrucción

# Ejemplo 1

```
// Inicialización
int max(int n,const int * a){
        int m = a[0];
        int i =1;
        while(i != n){
                if(m < a[i])
                m = a[i];
                ++i;
        }
        return m;
}
```

# Factores de los que depende

- Calidad del código que genera el compilador
- Instrucciones de la máquina
- Velocidad de la máquina
- Datos de entrada de un programa
    i. No dependen de la entrada (A veces)
    ii. Sólo dependen del tamaño del problema (Habitualmente)

# Casos mejor, peor, promedio

Por ejemplo, si el código tiene un condicionante en su ejecución, no se puede calcular el tiempo de ejecución sin saber los datos de entrada.

> Por ejemplo, determinar si existe un elemento en un array

```
for (int i = 0; i < sizeof(output); i++)
{
      if (input [count] == output[i][])
      {

            //.......
                        break;
      }

}
```

# ¿Tiene mucho sentido?

No es un tiempo exacto, porque:

- Depende del compilador
- Depende del tipo de máquina
- Depende la velocidad de la máquina
- Compite con los recursos del ordenador

# ¿Y los Algoritmos?

No se ejecutan en ninguna máquina

> Rendimiento de un algoritmo = Complejidad