

# VC1 – Introducción a los algoritmos

## 03MIAR – Algoritmos de Optimización

# Agenda

1. Definición de Algoritmo
2. Un poco de historia
3. Mi primer algoritmo
4. Tipos, propiedades, características de los algoritmos
5. Algoritmos de Optimización
6. Modelar
7. Diseñar
8. Analizar Algoritmos
9. Análisis de la complejidad temporal

## ¿Qué vamos a hacer en la asignatura?

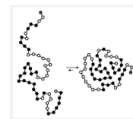
- Aprender a resolver problemas que pueden resolverse “computacionalmente”.
- Conocer diferentes técnicas deterministas.
- Analizar la complejidad
- Conocer técnicas metaheurísticas( las deterministas no son suficientes)

## ¿Por qué estudiar sobre algoritmos? (I)

**Internet:** Búsqueda web, enrutamiento de paquetes, intercambio de archivos distribuidos, ...



**Biología:** Proyecto genoma humano, plegamiento de proteínas, ...



**Ordenadores:** Diseño de circuitos, sistema de archivos, compiladores, ...



**Gráficos:** Películas, videojuegos, realidad virtual, ...



**Seguridad:** Teléfonos móviles, comercio electrónico, ...

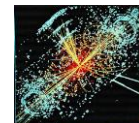


**Multimedia:** MP3, JPG, DivX, HDTV, reconocimiento facial, ...



**Redes sociales:** Recomendaciones, noticias, anuncios, ...

**Física:** Simulación de cuerpo N, simulación de colisión de partículas, ...



...

## ¿Por qué estudiar sobre algoritmos? (II)

- Nuevas oportunidades.
- Para la estimulación intelectual
- Para convertirse en un programador competente.

*"For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing." — Francis Sullivan*

[https://en.wikipedia.org/wiki/John\\_G.\\_F.\\_Francis](https://en.wikipedia.org/wiki/John_G._F._Francis)



*"An algorithm must be seen to be believed." — Donald Knuth*

[https://en.wikipedia.org/wiki/Donald\\_Knuth](https://en.wikipedia.org/wiki/Donald_Knuth)



*"I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships."*

*— Linus Torvalds (creator of Linux)*

[https://en.wikipedia.org/wiki/Linus\\_Torvalds](https://en.wikipedia.org/wiki/Linus_Torvalds)



# ¿Por qué estudiar sobre algoritmos? (III)

Computational models are replacing math models in scientific inquiry.

- Para descubrir los secretos de la vida y el universo.

$$E = mc^2$$

$$F = ma \quad F = \frac{Gm_1m_2}{r^2}$$

$$\left[ -\frac{\hbar^2}{2m} \nabla^2 + V(r) \right] \Psi(r) = E \Psi(r)$$

20<sup>th</sup> century science  
(formula based)

```
for (double t = 0.0; true; t = t + dt)
for (int i = 0; i < N; i++)
{
    bodies[i].resetForce();
    for (int j = 0; j < N; j++)
        if (i != j)
            bodies[i].addForce(bodies[j]);
}
```

21<sup>st</sup> century science  
(algorithm based)

*“Algorithms: a common language for nature, human, and computer.” — Avi Wigderson*

[https://en.wikipedia.org/wiki/Avi\\_Wigderson](https://en.wikipedia.org/wiki/Avi_Wigderson)

- Para divertirse y ganar dinero



## ¿Qué es un algoritmo?

- **Definición:** Un algoritmo es un conjunto de reglas ordenadas y finitas para realizar una actividad mediante la sucesión de pasos claramente establecidos(\*).
- Cada día usamos algoritmos sin darnos cuenta:
  - Operaciones numéricas sencillas(sumas, restas,...)
  - Poner la mesa o cocinar una receta de cocina
  - Elegir una película y buscar el asiento en el cine.
  - Conducir,...
- Centraremos nuestro estudio en los problemas de cálculo y más específicamente a los de optimización.

## ¿Por qué estudiar acerca de algoritmos? Razones

- **Prácticas:**
  - Conocer los algoritmos estándar para resolver ciertos problemas.
  - Ser capaces de diseñar nuevos algoritmos o evolucionar los existentes.
  - Analizar su eficacia para comparar entre algoritmos.
- **Teóricas:**
  - Es uno de los pilares de las ciencias de la computación.
  - Desarrollar habilidades analíticas para resolver problemas en general.



## Un poco de historia. al-Juarismi (780 dc.)

- La palabra algoritmo proviene del nombre **al-Juarismi** (y otros nombres traducidos similares) . Matemático persa que vivió del 780 al 850 dc. al que se le considera el padre del Álgebra(\*)
- Rompió con las bases geométricas griegas para resolver problemas y comenzó a usar los métodos algebraicos pero sin la notación que conocemos hoy.
- Método de reducción para resolver ecuaciones de 2º grado.  
<https://es.slideshare.net/SabrinaDechima/al-khwarizmi-17263168>

(\*) **Álgebra**: del árabe, significa reducción pero proviene del arte de recolocar los huesos que se rompían o se salían de su sitio.

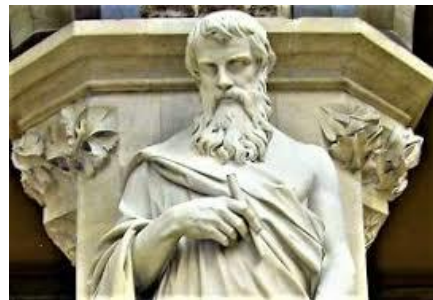
## Un poco de historia. Algoritmo de Euclides (325 ac.) (I)

- Encontrara el máximo común divisor de dos enteros.
- Se basa en aplicar iterativamente la igualdad :

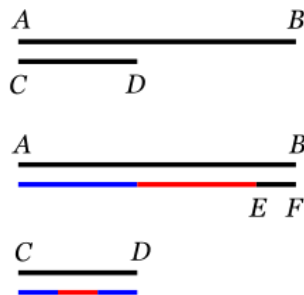
$$\text{m.c.d}(m,n) = \text{m.c.d}(n, \text{resto}(m,n) )$$

Hasta que el resto(segundo número) sea cero

- Ejemplo:  $\text{m.c.d}(60,24) = \text{m.c.d}(24,12) = \text{m.c.d}(12,0)$
- Ahora conocemos el álgebra pero Euclides lo hizo con geometría



### Algoritmo de Euclides

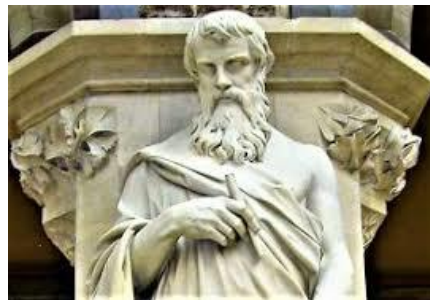
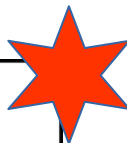


*EF* es la mayor medida común

## Un poco de historia. Algoritmo de Euclides (325 ac.) (II)

- Ahora podemos implementarlo:
  - Si  $n=0$  devuelve  $m$ . Si no ir a paso 2
  - Dividir  $m$  entre  $n$  y calcula el resto  $r$
  - Asigna el valor  $n$  a  $m$  y el valor  $r$  a  $n$ . Ir a al paso 1.
- Pseudocódigo(\*)

```
while  $n \neq 0$  do  
     $r \leftarrow m \bmod n$   
     $m \leftarrow n$   
     $n \leftarrow r$   
return  $m$ 
```



## Un poco de historia. Otros algoritmos

- **Método Herón** para calcular a través de iteraciones la raíz cuadrada de un número. Se conoce desde el siglo I dC.

Iterar

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

Donde **p** es el número del que queremos obtener la raíz

$n$	$x_n$ ( $x_0 = 1$ )	$ x_n - \sqrt{3} ^*$
1	2.0000000000000000	0.267949192431123
2	1.7500000000000000	0.017949192431123
3	1.732142857142857	0.000092049573980
4	1.732050810014727	0.000000002445850
5	1.732050807568877	0
6	1.732050807568877	0
7	1.732050807568877	0
8	1.732050807568877	0
9	1.732050807568877	0
10	1.732050807568877	0

## Un poco de historia. Otros algoritmos

- **Criba de Eratóstenes** para encontrar números primos. Se conoce desde el siglo II aC.

1. Crear una lista desde 2 a  $n$
2. Para  $i$  desde 2 hasta  $\text{int}(\sqrt{n})$   
Si  $i$  no está marcado  
Para  $j$  desde  $i$  hasta  $n \div i$ :  
Marcar  $i \times j$
3. El resultado es la lista no marcada

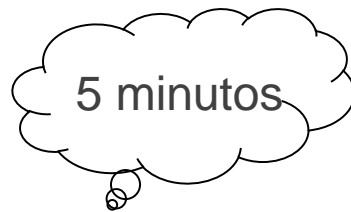
N=100

1	②	③	4	⑤	6	⑦	8	9	10
⑪	12	⑬	14	15	16	⑰	18	⑲	20
21	22	⑳	24	25	26	27	28	㉑	30
③1	32	33	34	35	36	③7	38	39	40
④1	42	④3	44	45	46	④7	48	49	50
51	52	⑤3	54	55	56	57	58	⑤9	60
⑥1	62	63	64	65	66	⑥7	68	69	70
⑦1	72	⑦3	74	75	76	77	78	⑦9	80
81	82	⑧3	84	85	86	87	88	⑧9	90
91	92	93	94	95	96	⑨7	98	99	100

## Mi primer algoritmo

- Podemos intentar pensar en un algoritmo que resuelva el siguiente problema:
- Encontrar los elementos comunes a dos listas de números  
 $A=[1,2,6,7,12,13,15]$   $B=[2,3,4,7,13]$

Debe devolver la lista (2,7,13)



## Mi primer algoritmo

- Podemos intentar pensar en un algoritmo que resuelva el siguiente problema:
- Encontrar los elementos comunes a dos listas de números  
A=[1,2,6,7,12,13,15] B=[2,3,4,7,13]

Debe devolver la lista (2,7,13)

Una solución

```
A=[1,2,6,7,12,13,15]
B=[2,3,4,7,13]

SALIDA = []

for i in A:
    for k in B:
        if i==k :
            SALIDA.append(i)

print(SALIDA)
```

📄 [2, 7, 13]

## Tipos de Algoritmos



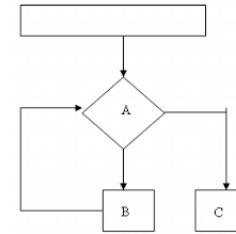
- Atendiendo a **la forma** en la que calculan la solución:
  - Algoritmos **deterministas**: Siempre la misma solución para los mismos datos de entrada (instancia).
  - Algoritmos **probabilistas**: Pueden generar soluciones diferentes para los mismos datos de entrada (instancia).
- Atendiendo al **tipo de solución** que proporcionan:
  - Algoritmos **exactos**: Proporcionan la solución óptima.
  - Algoritmos **aproximados**: Proporcionan buenas soluciones con un grado de aproximación al óptimo
  - Algoritmos **heurísticos**: Proporcionan buenas soluciones pero no podemos determinar si es la óptima.



# Propiedades de los Algoritmos

- **Finitud**

- El algoritmo debe finalizar después de un numero finito de pasos.



- **Precisión:**

- Cada etapa debe estar claramente especificada. No hay lugar para ambigüedades o interpretaciones(\*)

(\*) Los algoritmos probabilistas no siempre proporcionan la misma solución pero eso no quiere decir que sus pasos no estén claramente establecidos



## Partes de los Algoritmos

- **Entrada**
  - Es la información dada al algoritmo o los valores con los que va a trabajar.
- **Proceso:**
  - Son los cálculos y operaciones necesarios para que a partir de los valores de entrada se pueda llegar a un resultado.
- **Salida:**
  - Son los resultados finales o la transformación de los valores de entrada a través del proceso.



## Algoritmos de Optimización(I)

- Existen muchos problemas para los que se busca la mejor solución.
- **Optimización:** Obtención el mejor valor para alguna función objetivo dentro de un dominio de soluciones posibles.
- Las diferentes tipos de funciones objetivo y los diferentes tipos de dominio dan lugar a diferentes técnicas de optimización.
- En el capítulo de Problemas tipo veremos la clasificación de los problemas de optimización.

## Algoritmos de Optimización(II)

### Componentes:

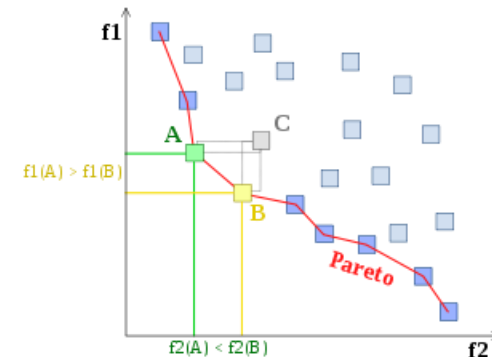
- **Función objetivo**: Medida cuantitativa de la bondad de un sistema a optimizar.
- **Variables**: Decisiones que afectan a la función objetivo.
- **Restricciones**: Conjunto de relaciones que las variables deben cumplir.

### Resolución:

- Encontrar el valor que deben tener las **variables** para optimizar(maximizar/minimizar) la **función objetivo** cumpliendo el conjunto de **restricciones**.

## Algoritmos de Optimización(III)

- **Optimización multiobjetivo:** En los problemas multiobjetivo se trata de optimizar más de una función objetivo
- En general en estos problemas las soluciones entran en conflicto. Si trato de mejorar una función penalizo a la otra.
- Por ejemplo: minimizar el tiempo de un trayecto(velocidad) y el combustible gastado.
- No existe una solución única sino un conjunto de soluciones para las que nos se pueda encontrar otra mejor en alguna función. **Óptimo de Pareto.**



## Desarrollo e implementación de algoritmos



- **Modelar**
  - ✓ Determinar los supuestos del problema y los datos de inicio.
  - ✓ Determinar la estructura de datos que represente la información del problema
  - ✓ No siempre es posible ajustarse al 100% de la realidad. Será necesario evaluar si con el modelo elegido podemos llegar a soluciones deseadas.
- **Diseñar**
  - ✓ Aplicar técnicas conocidas para la resolución de problemas.
- **Analizar**
  - ✓ Determinar las cantidades de operaciones que realizará el diseño realizado

# Desarrollo e implementación de algoritmos. Modelar

## • Modelo

Es una representación abstracta de una parte de la realidad compuesta por elementos que tienen una determinada estructura.

## • Tipos de Modelos

✓ Icónico

✓ Analógico

✓ Simbólico

✓ (\*) Matemático (lógica matemática, relaciones, ecuaciones, ...)

...



(\*)



Max  $3X + 8Y$

sujeto a:

$$2X + 4Y \leq 1.600 \quad (R1)$$

$$6X + 2Y \leq 1.700 \quad (R2)$$

$$Y \leq 350 \quad (R3)$$

$$X \geq 0$$

$$Y \geq 0$$

# Desarrollo e implementación de algoritmos. Modelar

## Riesgos

- Por exceso(muy cerca de la realidad):
  - Dificultad para encontrar un diseño de algoritmo que lo resuelva
  - Posible exceso de complejidad
- Por defecto(muy apartado de la realidad):
  - Las soluciones obtenidas pueden no ser las que se corresponden con el problema real.

Objetivo: Obtener un compromiso entre las dos opciones



## Desarrollo e implementación de algoritmos. Modelar

- **Modelo Matemático**

Ventajas:

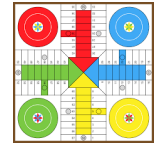
- ✓ Poner en evidencias relaciones no descubiertas
- ✓ Obtener propiedades y relaciones de los elementos
- ✓ Representar complejidad no posible en otros tipos de modelos
- ✓ Posibilidad de resolver el problema tanto desde el punto de vista analítico como numérico.
- ✓ Simular situaciones que en la realidad es difícil o imposible
- ✓ Posibilidad de reproducir para diferentes datos de entrada
- ✓ Ayudar a la toma de decisiones
- ✓ ...

## Desarrollo e implementación de algoritmos. Tipos de Modelos

- **Modelos Matemáticos según su función**
  - ✓ Predictivos: Como se comportará una variable. Técnicas estadísticas
  - ✓ Evaluativos: Encontrar todas las posibilidades. Árboles de decisión
  - ✓ **Optimización:** Identificar la mejor solución. Técnicas de optimización

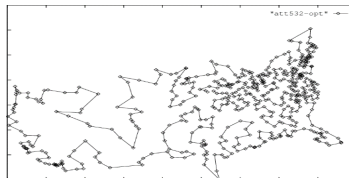
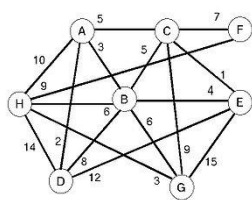
## Desarrollo e implementación de algoritmos. Tipos de Modelos

- Modelos Matemáticos según **los datos que se conocen**
  - **Deterministas**: Todos los datos del problema se conocen
  - **Estocásticos**: Hay datos del problema que nos se conocen
- Modelos Matemáticos según **cambia la realidad**
  - **Estáticos**: El paso del tiempo no influye en el comportamiento de otras variables.
  - **Dinámico**: El tiempo es una variable que influye en el resto de las variables.



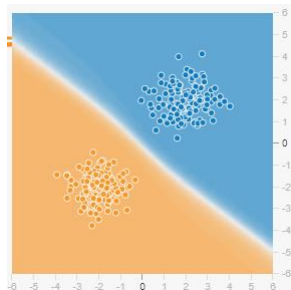
## Desarrollo e implementación de algoritmos. Modelar

- Características del nivel de resolubilidad del modelo
  - ✓ Tamaño del problema :  $n^{\circ}$  de elementos (variables, condiciones, ...)

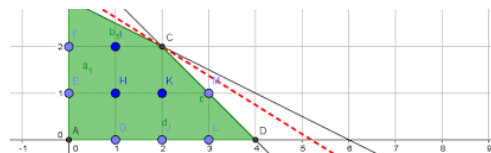


- ✓ Clase de problema: lineal, entero(y/o binarios), no lineal

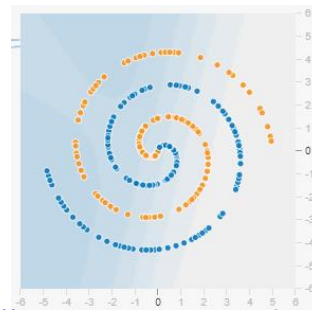
**lineal**



**entero**



**no lineal**



## Desarrollo e implementación de algoritmos. Fases

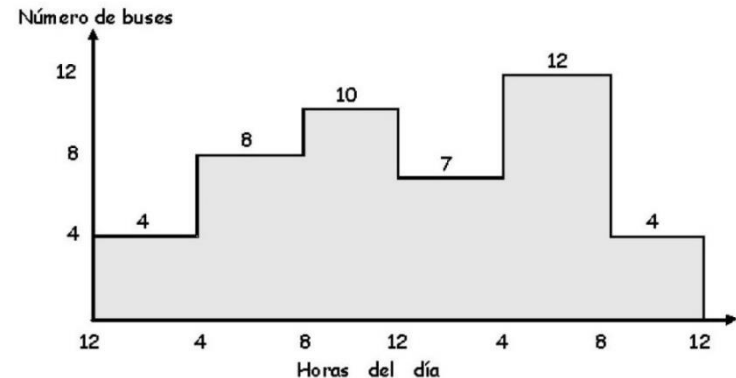
- **Conceptualización**
  - Conocer el problema para ser capaces de representar la realidad sin contradicciones o errores.
- **Formalización**
  - Establecer matemáticamente las relaciones entre los elementos.
- **Evaluación**
  - Establecer el procedimiento para resolver el problema

## Ejemplo. Desarrollo e implementación de algoritmos.

Una pequeña ciudad estudia introducir un sistema de transporte urbano de autobuses. Nos encargan estudiar el número mínimo de autobuses para cubrir la demanda. Se ha realizado un estudio para estimar el número mínimo de autobuses por franja horaria. Lógicamente este número varía dependiendo de la hora del día. Se observa que es posible dividir la franja horaria de 24h en tramos de 4 horas en los queda determinado el número de autobuses que se necesitan. Debido a la normativa cada autobús debe circular 8 horas como máximo y seguidas en cada jornada de 24h.



8 horas x día

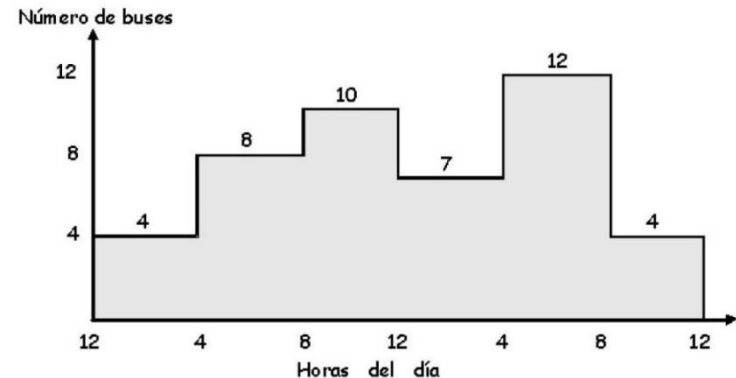


## Desarrollo e implementación de algoritmos. Ejemplo

Una pequeña ciudad estudia introducir un sistema de transporte urbano de autobuses. Nos encargan estudiar el número mínimo de autobuses para cubrir la demanda. Se ha realizado un estudio para estimar el número mínimo de autobuses por franja horaria. Lógicamente este número varía dependiendo de la hora del día. Se observa que es posible dividir la franja horaria de 24h en tramos de 4 horas en los que queda determinado el número de autobuses que se necesitan. Debido a la normativa cada autobús debe circular 8 horas como máximo y seguidas en cada jornada de 24h.



8 horas x día



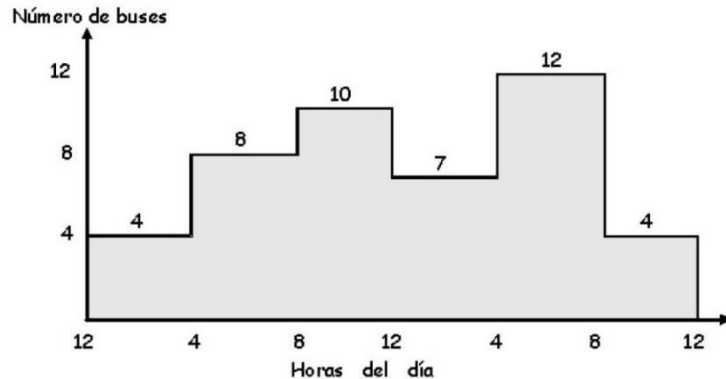
## Desarrollo e implementación de algoritmos. Ejemplo

- **Conceptualización**

- ✓ ¿Entendemos el problema?
- ✓ ¿Entendemos la función a optimizar?
- ✓ ¿Entendemos las restricciones?
- ✓ ¿Sabemos establecer las relaciones entre autobuses y franjas horarias?
- ✓ ¿Por qué 12 autobuses no es la solución?



8 horas x día



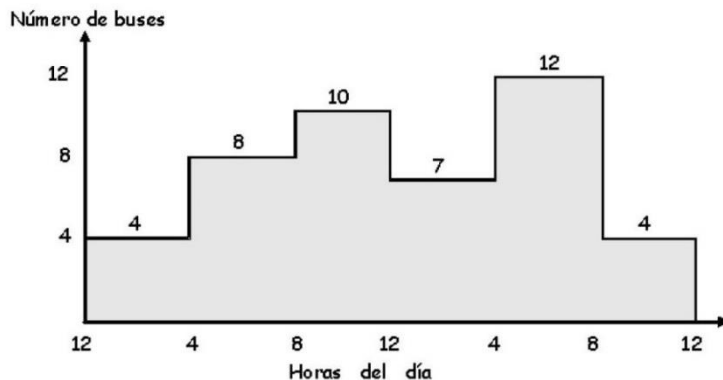


## Desarrollo e implementación de algoritmos. Ejemplo

- **Formalizar(I)**
  - ✓ ¿Qué variables elegimos?
  - ✓ ¿Cuál es la función objetivo?
  - ✓ ¿Cuáles son las restricciones?



8 horas x día



## Desarrollo e implementación de algoritmos. Ejemplo

- Formalizar(II)

- ✓ Definición de Variables

Cada  $X_j$  representa el nº de buses en cada tramo  $j = \{1,2,3,4,5,6\}$

$j = 1$  es el turno que empieza a las 0 am

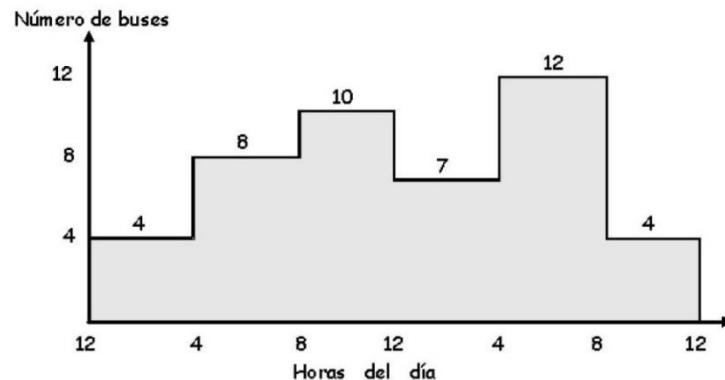
$j = 2$  es el turno que empieza a las 4 am

...

$j = 6$  es el turno que empieza a las 8 pm



8 horas x día



## Desarrollo e implementación de algoritmos. Ejemplo

- Formalizar(III)

✓ Relación de Variables con restricciones.

Horario de la demanda	Turnos de 8 horas, empezando a las 12 de la noche						Número de buses necesarios
	X <sub>1</sub> 0-8	X <sub>2</sub> 4-12	X <sub>3</sub> 8-16	X <sub>4</sub> 12-20	X <sub>5</sub> 16-0	X <sub>6</sub> 20-4	
12 - 4	✓					✓	4
4 - 8	✓	✓					8
8 - 12		✓	✓				10
12 - 4			✓	✓			7
4 - 8				✓	✓		12
8 - 12					✓	✓	4

## Desarrollo e implementación de algoritmos. Ejemplo

- Formalizar(IV)

✓ Formalización matemática

$\text{Min } X_1 + X_2 + X_3 + X_4 + X_5 + X_6$   $\longrightarrow$  Función Objetivo

$$X_1 + X_2 \geq 8$$

$$X_2 + X_3 \geq 10$$

$$X_3 + X_4 \geq 7$$

$$X_4 + X_5 \geq 12$$

$$X_5 + X_6 \geq 4$$

$$X_1 + X_6 \geq 4$$

$$X_1, X_2, X_3, X_4, X_5, X_6 \geq y \text{ enteros}$$

$\longrightarrow$  Restricciones

## Desarrollo e implementación de algoritmos. Ejemplo

- Evaluación

- ✓ Elegir el procedimiento para resolver el problema

$$\text{Min } X_1 + X_2 + X_3 + X_4 + X_5 + X_6$$

$$X_1 + X_2 \geq 8$$

$$X_2 + X_3 \geq 10$$

$$X_3 + X_4 \geq 7$$

$$X_4 + X_5 \geq 12$$

$$X_5 + X_6 \geq 4$$

$$X_1 + X_6 \geq 4$$

$$X_1, X_2, X_3, X_4, X_5, X_6 \geq y \text{ enteros}$$

## Desarrollo e implementación de algoritmos. Diseñar

- ✓ Es la parte creativa del proceso
- ✓ Necesitaremos conocer las **técnicas y heurísticas** que se demuestran en la práctica ser útiles para resolver problemas.
- ✓ Debemos demostrar formalmente que las respuestas son correctas para todas las posibles entradas.

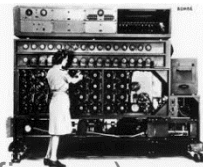
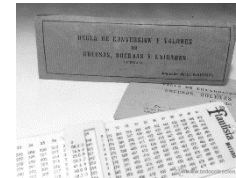
## Desarrollo e implementación de algoritmos. Analizar(I)

- Determinar los recursos necesarios. ¿Cómo de bueno es nuestro algoritmo?
  - Eficiencia en **tiempo**.
  - Eficiencia en espacio
- Análisis **teórico** / Análisis práctico (independencia del procesador)
- Comparación. ¿Existen mejores algoritmos?
- Clasificación de los problemas:

Años 30 : Problemas computables y no computables.

Años 50 : Complejidad de los problemas computables(búsqueda de algoritmos más eficaces).

Años 70 : Clasificación de los problemas computables: P y NP.



## Desarrollo e implementación de algoritmos. Analizar(II)

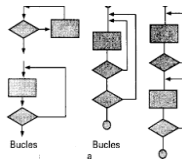
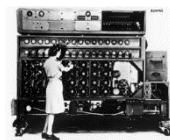
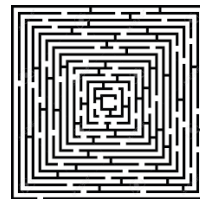
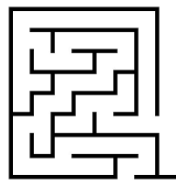
- El tiempo de ejecución de un algoritmo dependerá de :

- Tamaño de entrada (n) del problema

- Calidad del código del programa

- Procesador donde se ejecuta

- La complejidad del algoritmo





## Desarrollo e implementación de algoritmos. Analizar(III)

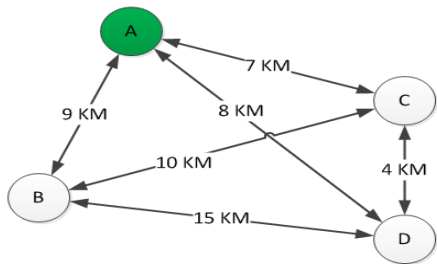
- **Análisis teórico temporal.** Analizamos el número de **operaciones elementales** en función del **tamaño de entrada**.

### Problema del Viajante

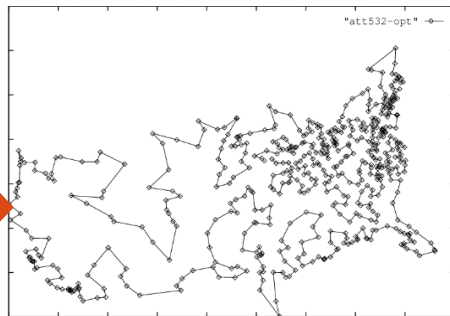
*(Recorrer todas las ciudades una sola vez)*

Algoritmo por fuerza bruta : explorar todas las posibilidades

#### 4 Ciudades



#### 532 Ciudades



¿Cuántas  
Rutas posibles?



## Desarrollo e implementación de algoritmos. Analizar(III)

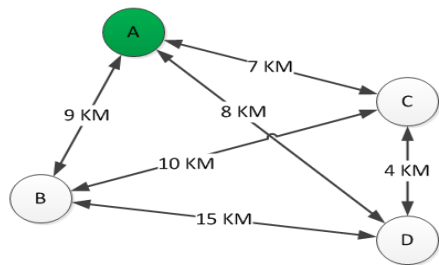
- **Análisis teórico temporal.** Analizamos el número de **operaciones elementales** en función del **tamaño de entrada**.

### Problema del Viajante

*(Recorrer todas las ciudades una sola vez)*

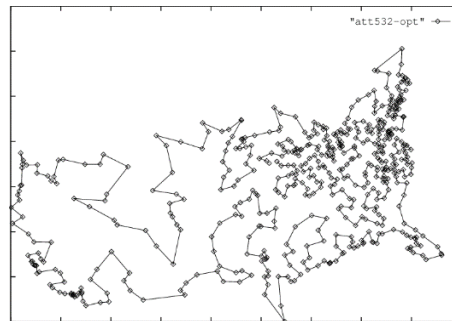
Algoritmo por fuerza bruta : explorar todas las posibilidades

#### 4 Ciudades



$4! : 2 = 12$  posibilidades

#### 532 Ciudades



$532! : 2$  posibilidades

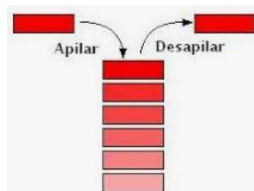
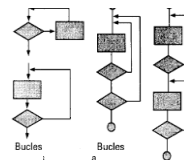
**Se necesita otra estrategia!**

## Análisis de la complejidad temporal (I)

- Mediremos la complejidad de un algoritmo por el **número de operaciones elementales(OE)** que realiza.
- ¿Cuáles son las operaciones elementales?
  - Suma, resta, multiplicación y división
  - Asignaciones
  - Condiciones , llamadas a funciones externas y retornos
  - Accesos a estructuras de datos



<code>a+=b</code>	<code>a = a + b</code>
<code>a-=b</code>	<code>a = a - b</code>
<code>a*=b</code>	<code>a = a * b</code>
<code>a/=b</code>	<code>a = a / b</code>
<code>a%=b</code>	<code>a = a % b</code>



## Análisis de la complejidad temporal (II)

- Orden de complejidad
  - ✓ Decimos que un algoritmo tiene orden de complejidad  $O(f)$  si su tiempo de ejecución en operaciones elementales está acotado, salvo por una constante, por  $f$  para todos los tamaños de entrada a partir de un cierto  $n_0$

## Análisis de la complejidad temporal (III)

- Ordenes de complejidad (de menor a mayor)



$O(1)$	Orden constante
$O(\log n)$	Orden logarítmico
$O(n)$	Orden lineal
$O(n \log n)$	Orden cuasi-lineal
$O(n^2)$	Orden cuadrático
$O(n^3)$	Orden cúbico
$O(n^k)$	Orden polinómico
$O(2^n)$	Orden exponencial
$O(n!)$	Orden factorial



**Tabla 1**

Comparación de los órdenes de complejidad según el tamaño del problema

$n$	$\log n$	$n^2$	$2^n$	$n!$
2	1	4	4	2
8	3	64	256	40.320
32	5	1.024	$4,3 \times 10^9$	$2,6 \times 10^{35}$
100	6	$10^4$	$1,2 \times 10^{27}$	$9,3 \times 10^{177}$

## Análisis de la complejidad temporal (IV)



- Ejemplo:  
n = tamaño de la lista A(el mayor)  
m = tamaño de la lista B
- Calcular las operaciones elementales:  
2 asignaciones                      2  
n asignaciones                      n  
n\*m asignaciones                      m\*n  
n\*m asignaciones                      m\*n

---

TOTAL

$$2(m*n) + n + 2$$



$$O(n^2)$$

```
A=[1,2,6,7,12,13,15]
B=[2,3,4,7,13]

SALIDA = []

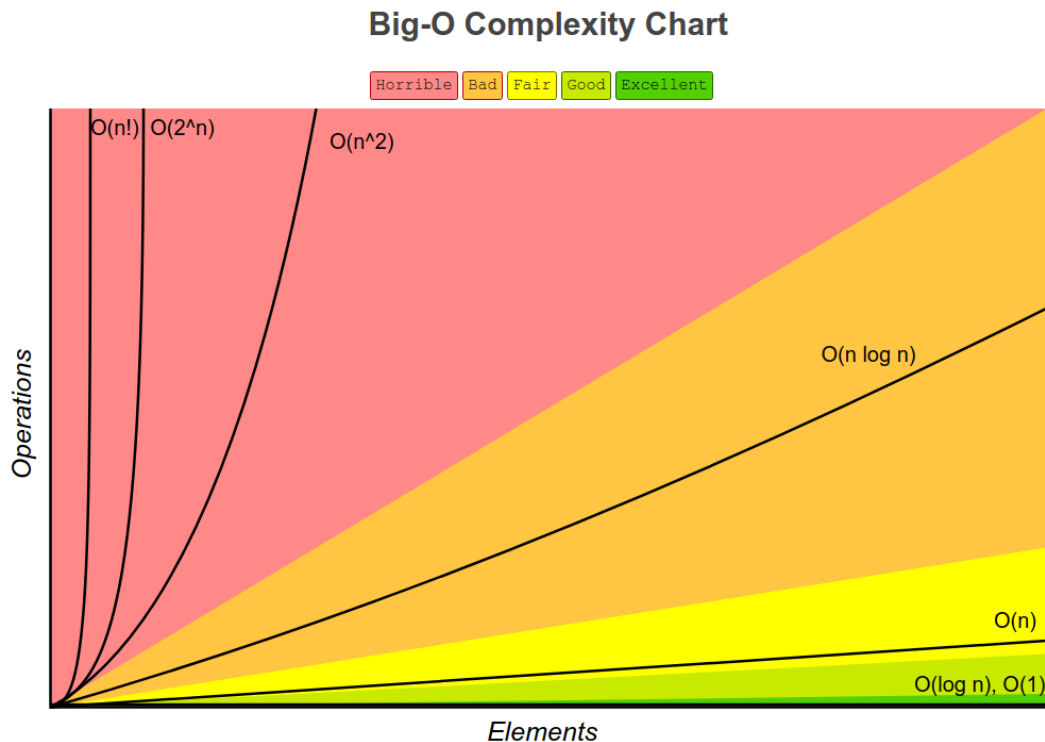
for i in A:
    for k in B:
        if i==k :
            SALIDA.append(i)

print(SALIDA)
```

[2, 7, 13]



# Análisis de la complejidad temporal (IV)



<https://www.bigocheatsheet.com/>



## Próxima clase. VC2 – Algoritmos de ordenación y Diseño de Algoritmos


1. Algoritmos de ordenación
2. Algoritmos voraces.
3. Búsqueda en Grafos (Prim y Kruskal)
4. Técnica de Vuelta atrás
5. Técnica de Divide y Vencerás
6. Técnica de Programación Dinámica

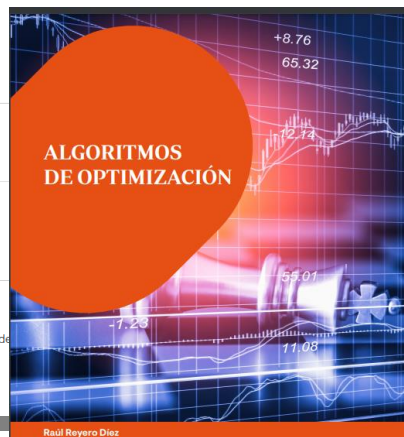
# Manual de la asignatura

**03MIAR**  
ALGORITMOS  
DE  
OPTIMIZACIÓN  
  
INICIO  
  
Bienvenida  
  
Calendario  
  
INFORMACIÓN

## 01. Materiales docentes

**Manual de la asignatura**  
Archivos adjuntos:  03MIAR\_RReyero\_nueva\_imagen.pdf (5,02 MB)

**03MIAR | ALGORITMOS DE OPTIMIZACIÓN**  
Explicación y práctica de las técnicas y métodos para diseñar y analizar algoritmos orientados a resolver problemas de optimización en el ámbito de...

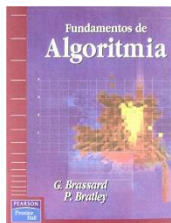


Raul Reyero Diez

MÁSTER UNIVERSITARIO EN INTELIGENCIA  
ARTIFICIAL  
Módulo de Matemáticas



## Bibliografía



### **Fundamentos de algoritmia: Una perspectiva de la ciencia de los computadores**

*Paul Bratley , Gilles Brassard*

ISBN 13: 9788489660007



### **Introducción al diseño y análisis de algoritmos**

*R.C.T. Lee,...*

ISBN 13: 9789701061244



### **Técnicas de diseño de algoritmos**

*Guerequeta, R., y Vallecillo, A.*

<http://www.lcc.uma.es/~av/Libro/indice.html>

## Bibliografía(en la biblioteca de la VIU)



***Una introducción a las matemáticas para el análisis y diseño de algoritmos***

Pérez Aguila, R. (2012).

<https://tinyurl.com/yzlt5oed>

# Gracias

[juanfrancisco.vallalta@campusviu.es](mailto:juanfrancisco.vallalta@campusviu.es)