Laboratorio de Computación I

Tabla de Contenidos

Elementos informáticos

El Software y sus características

El software en sus comienzos era la parte insignificante del hardware, lo que venía como añadidura, casi como regalo. Al poco tiempo adquirió una entidad propia.
En la actualizad, el software es la tecnología individual más importante en el mundo. Nadie en la década de 1950 podría haber predicho que el software se convertiría en una tecnología indispensable en los negocios, la ciencia, la ingeniería; tampoco podría preverse que una compañía de software podría volverse más grande e influyente que la mayoría de las compañías de la era industrial; que una red construida con software, llamada Internet cubriría y cambiaría todo, desde la investigación bibliográfica hasta las compras de los consumidores y los hábitos de las personas. Nadie podría haber imaginado que estaría relacionado con sistemas de todo tipo: transporte, medicina, militares, industriales, entretenimiento, automatización de hogares.

Una definición formal de software según la IEEE (Instituto de Ingeniería Eléctrica y Electrónica) es la siguiente:

Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación.

El software puede definirse como "el alma y cerebro de la computadora, la corporización de las funciones de un sistema, el conocimiento capturado acerca de un área de aplicación, la colección de los programas, y los datos necesarios para convertir a una computadora en una máquina de propósito especial diseñada para una aplicación particular, y toda la información producida durante el desarrollo de un producto de software". El software viabiliza el producto más importante de nuestro tiempo: la información.

Características del software:

1. El software es intangible, es decir, que se trata de un concepto abstracto.
2. Tiene alto contenido intelectual.
3. Su proceso de desarrollo es humano intensivo, es decir que la materia prima principal radica en la mente de quienes lo crean.
4. El software no exhibe una separación real entre investigación y producción.
5. El software puede ser potencialmente modificado, infinitamente.
6. El software no se desgasta
7. La mayoría del software, en su mayoría, aún se construye a medida.
8. El software no se desarrolla en forma masiva, debido a que es único.

Estructura Interna de una Computadora

Una computadora moderna consta de uno o más procesadores, una memoria principal, discos, impresoras, un teclado, un ratón, una pantalla o monitor, interfaces de red y otros dispositivos de entrada/salida. En general es un sistema complejo. Si todos los programadores de aplicaciones tuvieran que comprender el funcionamiento de todas estas partes, no escribirían código alguno. Es más: el trabajo de administrar todos estos componentes y utilizarlos de manera óptima es una tarea muy desafiante. Por esta razón, las computadoras están equipadas con una capa de software llamada sistema operativo, cuyo

trabajo es proporcionar a los programas de usuario un modelo de computadora mejor, más simple y pulcro, así como encargarse de la administración de todos los recursos antes mencionados.

La mayoría de las computadoras, grandes o pequeñas, están organizadas como se muestra en la siguiente figura. Constan fundamentalmente de tres componentes principales: Unidad Central de Proceso (UCP) o procesador, la memoria principal o central.

Si a los componentes anteriores se les añaden los dispositivos para comunicación con la computadora, aparece la estructura típica de un sistema de computadora: dispositivos de entrada, dispositivos de salida, memoria externa y el procesador/memoria central.

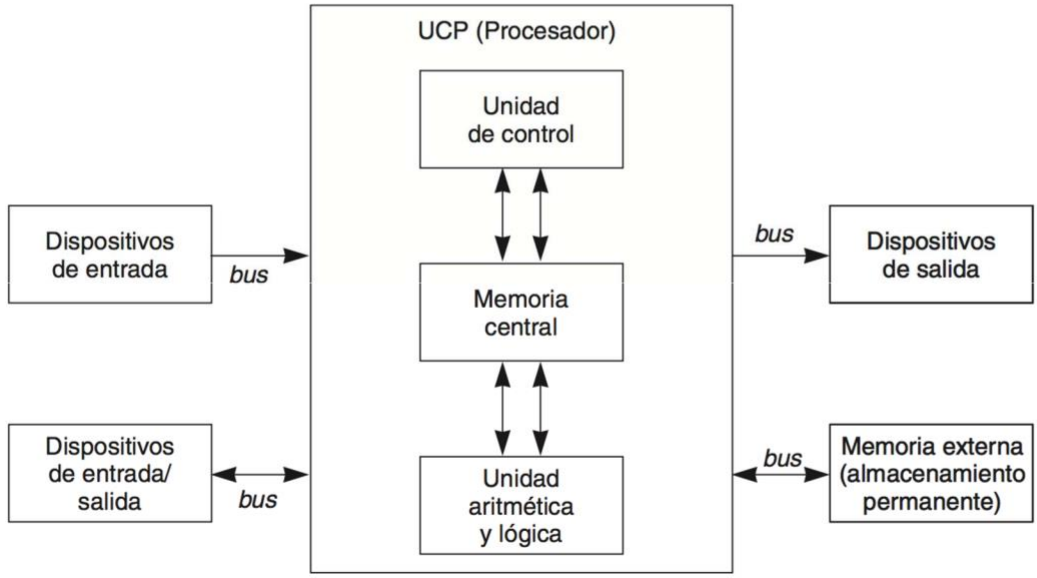


Figura 1: Estructura interna de un sistema de computadora

Los dispositivos de Entrada/Salida (E/S) (en inglés, Input/Output I/O) o periféricos permiten la comunicación entre la computadora y el usuario.

Los dispositivos de entrada, como su nombre indica, sirven para introducir datos en la computadora para su proceso. Los datos se leen de los dispositivos de entrada y se almacenan en la memoria central o interna. Los dispositivos de entrada convierten la información de entrada en señales eléctricas que se almacenan en la memoria central. Dispositivos de entrada típicos son teclados, lápices ópticos, joysticks, lectores de códigos de barras, escáneres, micrófonos, lectores de tarjetas digitales, lectores RFID (tarjetas de identificación por radio frecuencia), etc. Hoy, tal vez el dispositivo de entrada más popular es el ratón (mouse) que mueve un puntero gráfico (electrónico) sobre la pantalla, o más recientemente las pantallas táctiles, que facilitan la interacción usuario-máquina.

Los dispositivos de salida permiten representar los resultados (salida) del proceso. El dispositivo de salida típico es la pantalla o monitor. Otros dispositivos de salida son: impresoras (imprimen resultados en papel), trazadores gráficos (plotters), reconocedores (sintetizadores) de voz, parlantes, entre otros.

Los dispositivos de entrada/salida y dispositivos de almacenamiento masivo o auxiliar (memoria externa) son: unidad de discos (disquetes, CD-ROM, DVD, discos duros, etc.), videocámaras, memorias flash, USB, etc.

La memoria central o simplemente memoria (interna o principal) se utiliza para almacenar información (RAM, del inglés Random Access Memory). En general, la información almacenada en la memoria puede ser de dos tipos: instrucciones de un programa y datos con los que operan las instrucciones. Por

ejemplo, para que un programa se pueda ejecutar (correr, funcionar..., en inglés, run), debe ser situado en la memoria central, en una operación denominada carga (load) del programa. Después, cuando se ejecuta el programa, cualquier dato a procesar se debe llevar a la memoria mediante las instrucciones del programa. En la memoria central, hay también datos diversos y espacio de almacenamiento temporal que necesita el programa cuando se ejecuta a fin de poder funcionar.

La memoria central de una computadora es una zona de almacenamiento organizada en centenares o millares de unidades de almacenamiento individual o celdas. La memoria central consta de un conjunto de celdas de memoria (estas celdas o posiciones de memoria se denominan también palabras, aunque no guardan analogía con las palabras del lenguaje). El número de celdas de memoria de la memoria central, depende del tipo y modelo de computadora; hoy día el número suele ser millones (512, 1.024, etc.). Cada celda de memoria consta de un cierto número de bits (normalmente 8, un byte).

La unidad elemental de memoria se llama byte. Un byte tiene la capacidad de almacenar un carácter de información, y está formado por un conjunto de unidades más pequeñas de almacenamiento denominadas bits, que son dígitos binarios que pueden asumir como valor un 0 o un 1.

Siempre que se almacena una nueva información en una posición, se destruye (desaparece) cualquier información que en ella hubiera y no se puede recuperar. La dirección es permanente y única, el contenido puede cambiar mientras se ejecuta un programa.

La memoria central de una computadora puede tener desde unos centenares de miles de bytes hasta millones de bytes. Como el byte es una unidad elemental de almacenamiento, se utilizan múltiplos de potencia de 2 para definir el tamaño de la memoria central: Kilobyte (KB o Kb) igual a 1.024 bytes (2^{10}) —prácticamente se consideran 1.000—; Megabyte (MB o Mb) igual a 1.024 × 1.024 bytes = = 1.048.576 (2^{20}) —prácticamente se consideran 1.000.000; Gigabyte (GB o Gb) igual a 1.024 MB (2^{30}), 1.073.741.824 = prácticamente se consideran 1.000 millones de MB.

Byte	**Byte (B)**	*equivale a*	8 bits
Kilobyte	**Kbyte (KB)**	*equivale a*	1.024 bytes
Megabyte	**Mbyte (MB)**	*equivale a*	1.024 Kbytes
Gigabyte	**Gbyte (GB)**	*equivale a*	1.024 Mbytes
Terabyte	**Tbyte (TB)**	*equivale a*	1.024 Gbytes
1 **Tb** = 1.024 **Gb** = 1.024 × 1.024 **Mb** = 1.048.576 **Kb** = 1.073.741.824 **B**			

Tabla 1: Unidades de medida para el almacenamiento en la memoria

La Unidad Central de Proceso UCP, o procesador, dirige y controla el proceso de información realizado por la computadora. La UCP procesa o manipula la información almacenada en memoria; puede recuperar información desde memoria (esta información son datos o instrucciones de programas) y también puede almacenar los resultados de estos procesos en memoria para su uso posterior.

Más adelante veremos en profundidad cómo los programas hacen uso de la memoria para almacenar o leer datos a fin de utilizarlos para el desarrollo de sus funciones.

Redes de Computadoras

La fusión de las computadoras y las comunicaciones ha tenido una profunda influencia en cuanto a la manera en que se organizan los sistemas de cómputo. El concepto una vez dominante del “centro de cómputo” como un salón con una gran computadora a la que los usuarios llevaban su trabajo para

procesarlo es ahora totalmente obsoleto, (aunque los centros de datos que contienen miles de servidores de Internet se están volviendo comunes). El viejo modelo de una sola computadora para atender todas las necesidades computacionales de la organización se ha reemplazado por uno en el que un gran número de computadoras separadas pero interconectadas realizan el trabajo. A estos sistemas se les conoce como redes de computadoras.

Se dice que dos computadoras están interconectadas si pueden intercambiar información. La conexión no necesita ser a través de un cable de cobre; también se puede utilizar fibra óptica, microondas, infrarrojos y satélites de comunicaciones. Las redes pueden ser de muchos tamaños, figuras y formas, como veremos más adelante. Por lo general se conectan entre sí para formar redes más grandes, en donde Internet es el ejemplo más popular de una red de redes.

Imaginemos el sistema de información de una empresa como si estuviera constituido por una o más bases de datos con información de la empresa y cierto número de empleados que necesitan acceder a esos datos en forma remota. En este modelo, los datos se almacenan en poderosas computadoras denominadas servidores. A menudo estos servidores están alojados en una ubicación central y un administrador de sistemas se encarga de su mantenimiento. Por el contrario, los empleados tienen en sus escritorios máquinas más simples conocidas como clientes, con las cuales acceden a los datos remotos, por ejemplo, para incluirlos en las hojas de cálculo que desarrollan (algunas veces nos referiremos al usuario humano del equipo cliente como el “cliente”, aunque el contexto debe dejar en claro si nos referimos a la computadora o a su usuario). Las máquinas cliente y servidor se conectan mediante una red, como se muestra en la figura 2.

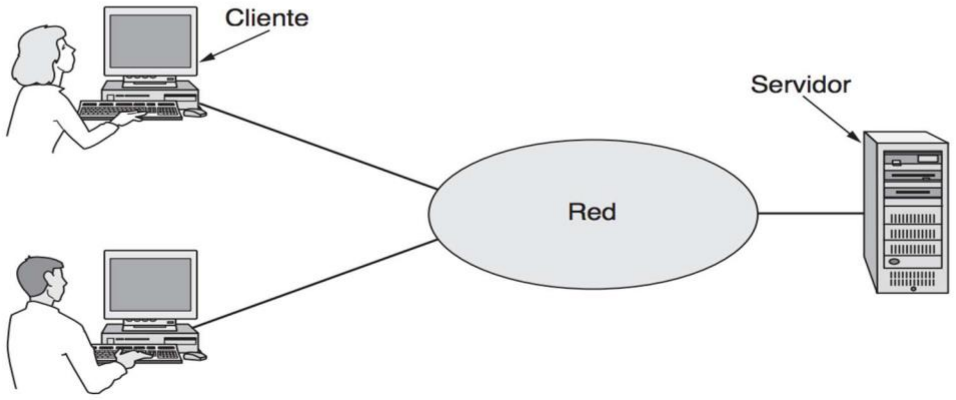


Figura 2: Esquema de una red de computadoras

A esta disposición se le conoce como modelo cliente -servidor. Es un modelo ampliamente utilizado y forma la base de muchas redes. La realización más popular es la de una aplicación web, en la cual el servidor genera páginas web basadas en su base de datos en respuesta a las solicitudes de los clientes que pueden actualizarla. El modelo cliente-servidor es aplicable cuando el cliente y el servidor se encuentran en el mismo edificio (y pertenecen a la misma empresa), pero también cuando están muy alejados. Por ejemplo, cuando una persona accede desde su hogar a una página en Internet se emplea el mismo modelo, en donde el servidor web remoto representa al servidor y la computadora personal del usuario representa al cliente. En la mayoría de las situaciones un servidor puede atender un gran número (cientos o miles) de clientes simultáneamente.

La evolución de las comunicaciones y los dispositivos personales, así como las necesidades emergentes de compartir información en tiempo real han posibilitado la expansión de Internet a todos los rincones del mundo. De esta forma cualquier persona puede acceder a sus archivos, compartir datos, comunicarse o buscar información en cualquier momento a través de su computadora, notebook, teléfonos celulares entre otros, tal como se muestra en la siguiente figura.

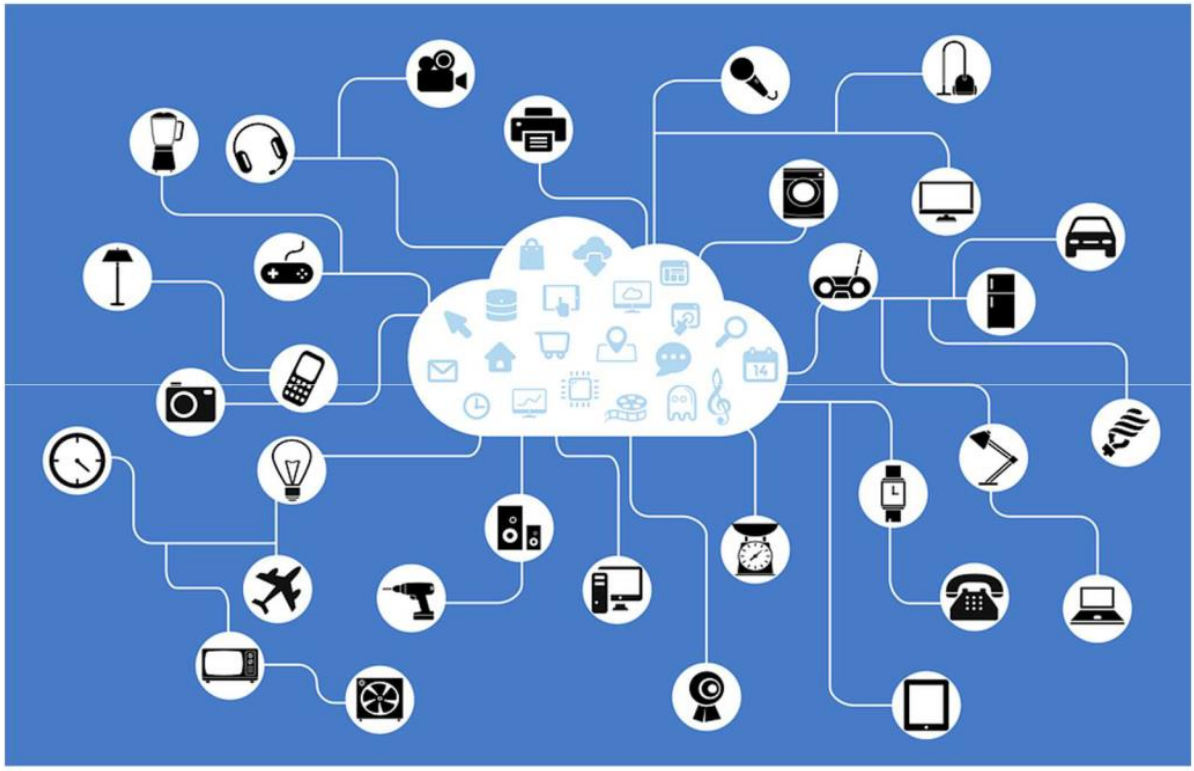


Figura 3: Integración de tecnología a través de Internet

Programación y construcción de Software

El único tipo de instrucciones que una computadora puede entender es el lenguaje de máquina, o lenguaje de bajo nivel, donde diferentes tipos de procesadores pueden tener distintos lenguajes de máquina. El lenguaje máquina está compuesto de ceros y unos lo que hace que programar en lenguaje máquina sea un proceso tedioso y sujeto a errores.

Una alternativa a utilizar lenguaje de máquina es el lenguaje Assembly, Assembler o ensamblador, que es también un lenguaje de bajo nivel y es más fácil de entender que ceros y unos. Sin embargo, el único lenguaje que una computadora puede entender directamente es el lenguaje máquina, ¿entonces cómo es posible que entienda lenguajes como Assembler? La respuesta es que el lenguaje Assembler es convertido o traducido a lenguaje de máquina mediante un programa llamado *ensamblador*. Es importante destacar que hay una correspondencia directa entre el lenguaje Assembler y el lenguaje máquina, lo que significa que para cada instrucción de lenguaje assembler existe una instrucción de lenguaje máquina, lo que hace la traducción un proceso directo.

Sin embargo, más allá de que el lenguaje Assembleres más sencillo que el lenguaje máquina, distintos tipos de procesadores tienen diferentes conjuntos de instrucciones lo que se traduce en distintos dialectos de Assembler de una computadora a otra.

La solución para hacer la tarea de programación más sencilla y posibilitar a los programas funcionar en distintos tipos de computadoras es utilizar lenguajes de alto nivel, que son más similares al lenguaje natural que utilizamos para comunicarnos diariamente y por motivos históricos estos lenguajes utilizan palabras del idioma inglés. Uno de los primeros lenguajes de programación de alto nivel fue FORTRAN (del inglés FORmula TRANslation, o traducción de fórmulas) que fue desarrollado en los comienzos de los años 50 para ayudar a resolver problemas matemáticos. Desde ese entonces, una gran cantidad de lenguajes de programación de alto nivel han sido creados para abordar distintos tipos de problemas y solucionar las necesidades de distintos tipos de usuarios. Algunos de ellos incluyen a COBOL, también desarrollado en los 50 para abordar aplicaciones empresariales y de negocios; BASIC en los 60 para programadores recién iniciados, Pascal en los 70 para problemas científicos, C, C++ y muchos otros.

Los sistemas y su enfoque

¿Por qué hablamos de *sistemas*?

En la primera mitad del siglo XX, surgió la necesidad de diseñar métodos de investigación y estudio de los fenómenos complejos a causa de una acumulación de problemáticas en las que los métodos de investigación de las ciencias particulares se mostraban insuficientes. Por un lado, los *nuevos sistemas de producción* que incluían varias automatizaciones, el manejo de grandes cantidades de energía (termoeléctrica, nuclear…) que requería de especialistas de variadas ramas, el desarrollo y organización de transporte terrestre, marítimo y aéreo y otros fenómenos. Por otro, los *grandes desarrollos científicos* en la física (relatividad, estructura atómica, mecánica cuántica), biología (genética, evolución, estudio de poblaciones), química (teoría del enlace de Lewis, tabla periódica, estructura cristalina), matemática (álgebra de Boole, desarrollo del cálculo, problemas de Hilbert). Estas grandes revoluciones en el hacer y el pensar hicieron necesario el desarrollo de un enfoque complejo para la investigación de fenómenos complejos. Así nació el *enfoque sistémico*, sustentado por la Teoría General de los Sistemas (TGS) formulada por Ludwig von Bertalanffy a mediados del siglo XX.

Bertalanffy se dedicó especialmente a los organismos como sistemas biológicos, pero luego generalizó su estudio a todo tipo de sistemas. De tal manera que hoy se utiliza el término sistema en todas las áreas del conocimiento humano.

¿Qué es un Sistema?

Llamamos **sistema** a todo conjunto de elementos relacionados entre sí –puede ser por una finalidad en común-, que tienen un cierto orden u organización y que cumplen una función.

Los sistemas tienen **composición** (los elementos que lo forman), una **estructura** interna dada por el conjunto de relaciones entre sus componentes. Y también tienen un **entorno o ambiente** que es el conjunto de cosas que no pertenecen al sistema pero que actúan sobre él o sobre las que él actúa intercambiando **materia, energía** e **información** (MEI).

Figura 4: Elementos de un sistema

Los sistemas están inmersos en un **entorno o ambiente,** que es el conjunto de elementos que está fuera del sistema, es decir que no pertenecen al sistema pero que actúan sobre él o sobre las que el sistema actúa intercambiando **materia, energía** e **información** (MEI).

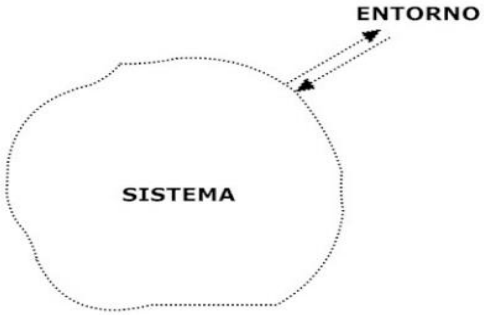


Figura 5: Relación del sistema con su entorno

Características de los sistemas

La característica principal de los sistemas es que poseen una **propiedad emergente** que no poseen sus componentes particulares. Por ejemplo, *la vida* es la propiedad emergente de un sistema compuesto por huesos, órganos, etc.; *marchar* es la propiedad emergente del sistema automóvil compuesto por chapas, motor, luces, etc. Este hecho se suele enunciar con la siguiente afirmación

EL TODO ES MÁS QUE LA SUMA DE LAS PARTES

Otras características de los sistemas son:

a. **Límite o frontera**: Son demarcaciones que permiten establecer qué elementos pertenecen o no al sistema. Los límites pueden ser:
   - *Concretos*: los que tienen existencia material (ríos que separan países, paredes que definen aulas, etc.)
   - *Simbólicos*: los que no tienen existencia material y vienen dados por acuerdos, reglas o normas (un alumno pertenece a un curso porque lo establece la escuela, más allá de que pueda hallarse en otro salón o fuera de la misma)
b. **Depósitos o almacenamientos:** son lugares donde se almacena materia, energía o información (MEI). Los depósitos pueden ser:
   - *Permanentes*: aquellos en que están diseñados para que su contenido no se altere (CD-ROM, libros, carteles fijos, etc.)
   - *Transitorios*: aquellos diseñados para que su contenido sufra modificaciones (pizarrón, cartuchera, tanques de agua, etc.)
c. **Canales**: Son lugares o conductos por donde circula materia, energía o información (MEI). Los canales pueden comunicar dos sistemas entre sí o partes de un mismo sistema (las calles pueden ser canales de materia, los cables pueden ser canales de energía si llevan corriente o de información si son telefónicos o de redes, etc.)
d. **Subsistemas**: los sistemas complejos (muchos componentes y relaciones entre ellos) pueden dividirse para su estudio en subsistemas. Esto permite diferentes niveles de estudio de los mismos. Se llama *nivel cero* al análisis del sistema en su totalidad y su intercambio con el entorno. A partir de allí se define el nivel 1, nivel 2, etc.

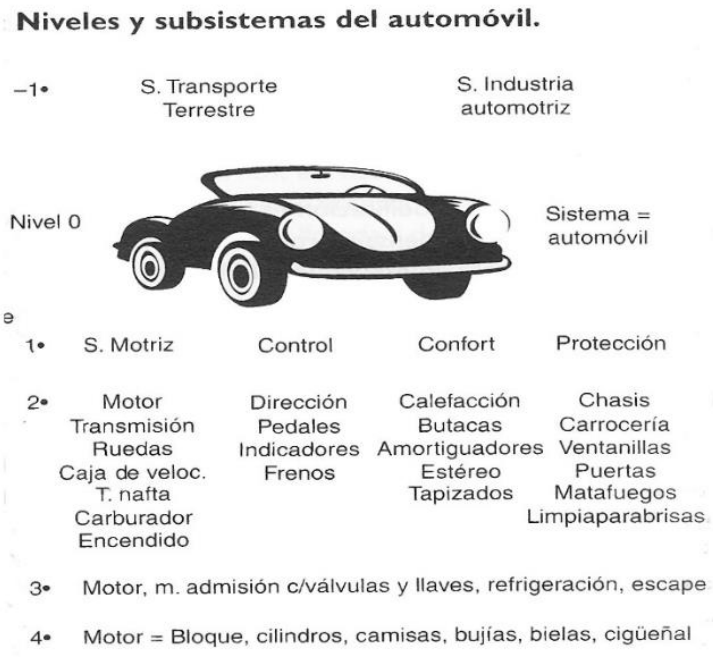


Figura 6: Sistemas y Subsistemas

Intercambio entre sistemas

Los sistemas intercambian entre sí materia, energía e información (MEI). Para que se dé este intercambio es necesario que MEI atraviese los límites del sistema hacia (o desde) el entorno. Si el sistema intercambia con el medio se dice que es *abierto*, de lo contrario se considera *cerrado*.

En *sistemas cerrados* cualquier estado final está determinado por sus condiciones iniciales, ya que no hay modo de que el entorno actúe sobre él. Si un sistema cerrado tampoco intercambia energía se dice que es aislado. En realidad, el único sistema que se considera absolutamente aislado es el universo. De igual modo, muchos sistemas mecánicos e informáticos pueden considerarse razonablemente cerrados.

Los *sistemas abiertos*, en cambio, pueden, crecer, cambiar, adaptarse al ambiente, incluso algunos reproducirse. Si un sistema posee la organización necesaria para controlar su propio desarrollo, asegurando la continuidad de su composición y estructura (*homeostasis*) y la de los flujos y transformaciones con que funciona (*homeorresis*) –mientras las perturbaciones producidas desde su entorno no superen cierto grado–, entonces el sistema es *autopoyético*. Los seres vivos, los ecosistemas y organizaciones sociales pueden considerarse sistemas abiertos.

Estos flujos de MEI se pueden representar en diagramas como el siguiente

Figura 7: Entradas y Salidas

Para clarificar, las líneas de los diferentes flujos pueden representarse por diferentes colores o trazos.

Este es el *nivel cero* de representación de un sistema, con las entradas y salidas de MEI que atraviesan sus límites. Este tipo de representaciones se denomina *diagrama de entrada y salida* (E/S o U/O) o *diagrama de caja negra*, ya que no interesa mostrar qué sucede dentro del sistema.

Sistemas tecnológicos

Los **sistemas tecnológicos**, son aquellos diseñados por los seres humanos para que cumplan con una finalidad específica. Por eso se dice que son *sistemas teleológicos artificiales*. La orientación para al fin que se busca suele definir la propiedad emergente del sistema tecnológico. En el ejemplo del automóvil, la propiedad emergente de marchar también se busca como finalidad o propósito del sistema.

Es conveniente aclarar que *los sistemas son recortes de la realidad* que alguien se propone estudiar o considerar; a ese recorte se le llama ***Abstracción***. En algunos sistemas tecnológicos como un automóvil es sencillo identificar este recorte. Sin embargo, en la red de generación y distribución de energía eléctrica del país no resulta tan sencillo.

Algunos sistemas tecnológicos se caracterizan por procesar materia: son los **sistemas de procesamiento de materia** (SM). Estos están diseñados para producir, procesar, generar, transformar o distribuir materiales. Las industrias, las huertas, las licuadoras, etc. pueden considerarse SM.

Otros se caracterizan por procesar energía, los **sistemas de procesamiento de energía** (SE). Estos están diseñados para generar, transformar, distribuir energía. Los ventiladores, automóviles, represas hidroeléctricas, explosivos, etc. pueden considerarse SE.

Los que se caracterizan por procesar información se llaman **sistemas de información** (SI). Están diseñados con el fin de generar, trasformar y distribuir información entre otras tareas. Los sistemas que controlan los automóviles, las redes sociales, los sistemas de punto de venta, el comercio electrónico, por mencionar algunos, son ejemplos de SI.

Desde la aparición del software, los SI han incorporado el software para hacer más eficiente su funcionamiento a un grado tal que se los denomina **Sistemas Informáticos,** acoplando la palabra "automático" a la palabra "información".

¿Cómo se construye el Software?

El software, como cualquier otro producto, se construye aplicando un proceso que conduzca a un resultado de calidad, que satisfaga las necesidades de quienes lo utilizan. Un proceso de desarrollo de software es una secuencia estructurada de actividades que conduce a la obtención de un producto de software. En definitiva, un proceso define quién está haciendo qué, cuándo y cómo alcanzar un determinado objetivo. En este caso el objetivo es construir un producto de software nuevo o mejorar uno existente.

Figura 9: Proceso de Construcción del Software

Pueden identificarse cuatro actividades fundamentales que son comunes a todos los procesos de software:

1. **Especificación del software**: donde clientes y profesionales definen el software que se construirá, sus características y las restricciones para su uso.

2. **Desarrollo del software**, donde se diseña y programa el software.
3. **Validación del software**, donde se controla que el software satisfaga lo que el cliente quiere.
4. **Evolución del software**, donde se incorporan mejoras y nuevas características que permitirán a ese producto adaptarse a las necesidades cambiantes del cliente y el mercado.

Si consideramos las características del software que se explicaron anteriormente, determinamos como conclusión que el software no se obtiene por medio de un proceso de manifactura en serie o como líneas de producción, sino que para obtenerlo usamos un **proyecto**, que se lo puede definir como un esfuerzo planificado, temporal y único, realizado para crear productos o servicios únicos que agreguen valor. Estos proyectos utilizan **procesos** que definen que tareas deben realizar las personas que trabajan en el proyecto, para obtener los resultados deseados, utilizando como apoyo herramientas que facilitarán su trabajo. En este caso el resultado deseado en el **Producto de Software.**

Figura 10: Relación entre Proceso, Proyecto y Producto en el desarrollo de Software

Diseño de Algoritmos

El hombre, en el día a día, se enfrenta constantemente a diferentes problemas que debe solucionar y para lograr solucionarlos hace uso de herramientas que le facilitan la tarea. Así, podemos pensar el uso de una calculadora para poder sumar el precio de los productos en un local y así cobrarle al cliente.

Al igual que la calculadora, la computadora también sirve para resolver problemas, pero la diferencia está en la capacidad de procesamiento de las computadoras, que hace que se puedan resolver problemas de gran complejidad, que, si los quisiéramos resolver manualmente, nos llevaría mucho tiempo o ni siquiera podríamos llegar a resolverlos.

Un programador es antes que nada una persona que resuelve problemas; el programador procede a resolver un problema, a partir de la definición de un algoritmo y de la traducción de dicho algoritmo a un programa que ejecutará la computadora.

En la oración anterior se nombran algunos conceptos que debemos profundizar:

Algoritmo: un algoritmo es un método para resolver un problema, que consiste en la realización de un conjunto de pasos lógicamente ordenados tal que, partiendo de ciertos datos de entrada, permite obtener ciertos resultados que conforman la solución del problema. Así, como en la vida real, cuando tenemos que resolver un problema, o lograr un objetivo, por ejemplo: “Tengo que atarme los cordones”, para alcanzar la solución de ese problema, realizamos un conjunto de pasos, de manera ordenada y secuencial. Es decir, podríamos definir un algoritmo para atarnos los cordones de la siguiente forma:

1. Ponerme las zapatillas.
2. Agarrar los cordones con ambas manos.
3. Hacer el primer nudo.
4. Hacer un bucle con cada uno de los cordones.
5. Cruzar los dos bucles y ajustar.
6. Corroborar que al caminar los cordones no se sueltan y la zapatilla se encuentra correctamente atada.

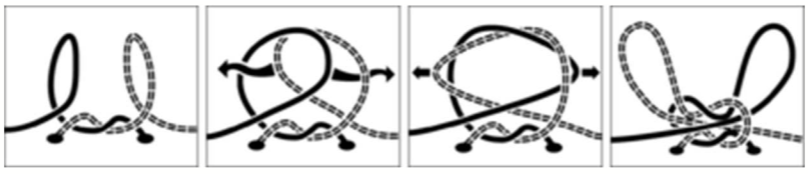


Figura 11: Algoritmo gráfico para atarse los cordones

El concepto de algoritmo es fundamental en el proceso de programación de una computadora, pero si nos detenemos a observar a nuestro alrededor, así como el ejemplo anterior podemos descubrir muchos otros: nos están dando un algoritmo cuando nos indican la forma de llegar a una dirección dada, seguimos algoritmos cuando conducimos un automóvil o cualquier tipo de vehículo. Todos los procesos de cálculo matemático que normalmente realiza una persona en sus tareas cotidianas, como sumar, restar, multiplicar o dividir, están basados en algoritmos que fueron aprendidos en la escuela primaria. Como se ve, la ejecución de algoritmos forma parte de la vida moderna.

Por otro lado, la complejidad de los distintos problemas que podamos abordar puede variar desde muy sencilla a muy compleja, dependiendo de la situación y la cantidad de elementos que intervienen. En casos de mayor complejidad suele ser una buena solución dividir al problema en diferentes subproblemas que puedan ser resueltos de manera independiente. De esta forma la solución final al problema inicial será determinada por las distintas soluciones de los problemas más pequeños cuya

resolución es más sencilla.

Programa: luego de haber definido el algoritmo necesario, se debe traducir dicho algoritmo en un conjunto de instrucciones, entendibles por la computadora, que le indican a la misma lo que debe hacer; este conjunto de instrucciones conforma lo que se denomina, un programa.

Para escribir un programa se utilizan lenguajes de programación, que son lenguajes que pueden ser entendidos y procesados por la computadora. Un lenguaje de programación es tan sólo un medio para expresar un algoritmo y una computadora es sólo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente.

Algoritmos

Concepto

Es un método para resolver un problema, que consiste en la realización de un conjunto de pasos lógicamente ordenados, tal que, partiendo de cierto s datos de entrada, permite obtener ciertos resultados que conforman la solución del problema.

Características de los algoritmos

Las características fundamentales que debe cumplir todo algoritmo son:

• Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.

• Un algoritmo debe estar específicamente definido. Es decir, si se ejecuta un mismo algoritmo dos veces, con los mismos datos de entada, se debe obtener el mismo resultado cada vez.

• Un algoritmo debe ser finito. Si se sigue un algo ritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos. Debe tener un inicio y un final.

• Un algoritmo debe ser correcto: el resultado del algoritmo debe ser el resultado esperado.

• Un algoritmo es independiente tanto del lenguaje de programación en el que se expresa como de la computadora que lo ejecuta.

Como vimos anteriormente, el programador debe constantemente resolver problemas de manera algorítmica, lo que significa plantear el problema de forma tal que queden indicados los pasos necesarios para obtener los resultados pedidos, a partir de los datos conocidos. Lo anterior implica que un algoritmo básicamente consta de tres elementos: Datos de Entrada, Procesos y la Información de Salida.

Figura 12: Estructura de un programa, datos de entrada y salida

Cuando dicho algoritmo se transforma en un programa de computadora:

- Las entradas se darán por medio de un dispositivo de entrada (como los vistos en el bloque anterior), como pueden ser el teclado, disco duro, teléfono, etc. Este proceso se lo conoce como entrada de datos, operación de lectura o acción de leer.
- Las salidas de datos se presentan en dispositivos periféricos de salida, que pueden ser pantalla, impresora, discos, etc. Este proceso se lo conoce como salida de datos, operación de escritura o acción de escribir.

Dado un problema, para plantear un algoritmo que permita resolverlo, es conveniente entender correctamente la situación problemática y su contexto, tratando de deducir del mismo los elementos ya indicados (entradas, procesos y salida). En este sentido entonces, para crear un algoritmo:

1. Comenzar identificando los resultados esperados, porque así quedan claros los objetivos a cumplir.

2. Luego, individualizar los datos con que se cuenta y determinar si con estos datos es suficiente para llegar a los resultados esperados. Es decir, definir los datos de entrada con los que se va a trabajar para lograr el resultado.

3. Finalmente, si los datos son completos y los objetivos claros, se intentan plantear los procesos necesarios para pasar de los datos de entrada a los datos de salida.

Para comprender esto, veamos un ejemplo:

Problema:

Obtención del área de un rectángulo:

Altura: 5 cm

Base: 10 cm

1. Resultado esperado: área del rectángulo. Salida: área
Fórmula del área: base x altura.
2. Los datos con los que se dispone, es decir las entradas de datos son:
    Dato de Entrada 1: altura: 5 cm
    Dato de Entrada 2: base: 10 cm
3. El proceso para obtener el área del rectángulo:
    área = base * altura área = 50

Herramientas para la representación gráfica de los algoritmos

Como se especificó anteriormente, un algoritmo es independiente del lenguaje de programación que se utilice. Es por esto, que existen distintas técnicas de representación de un algoritmo que permiten está diferenciación con el lenguaje de programación elegido. De esta forma el algoritmo puede ser representado en cualquier lenguaje. Existen diversas herramientas para representar gráficamente un algoritmo. En este material presentaremos dos:

1. Diagrama de flujo.

2. Lenguaje de especificación de algoritmos: pseudo código.

Diagramas de Flujo

Un diagrama de flujo hace uso de símbolos estándar que, unidos por flechas, indican la secuencia en que se deben ejecutar.

Estos símbolos son, por ejemplo:

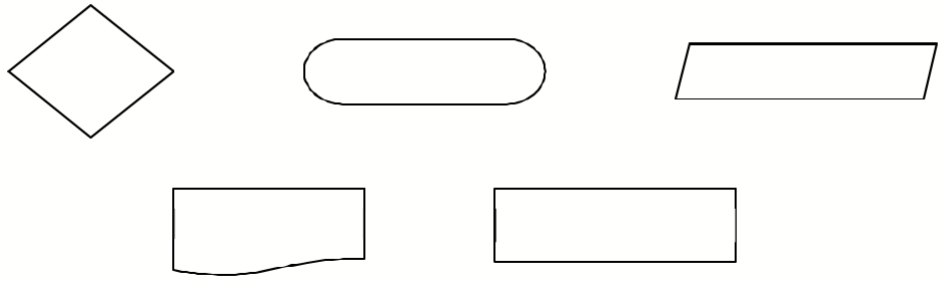


Figura 13: Símbolos utilizados en Diagramas de Flujo

Más adelante, a medida que profundizamos en los temas, retomaremos el uso de esta herramienta y mostraremos algunos ejemplos de su uso.

Pseudocódigo

Conocido como lenguaje de especificación de algoritmos, el pseudocódigo tiene una estructura muy similar al lenguaje natural y sirve para poder expresar algoritmos y programas de forma independiente del lenguaje de programación. Además, es muy utilizado para comunicar y representar ideas que puedan ser entendidas por programadores que conozcan distintos lenguajes. El pseudocódigo luego se traduce a un lenguaje de programación específico ya que la computadora no puede ejecutar el pseudocódigo. Su uso tiene ventajas porque permite al programador una mejor concentración de la lógica y estructuras de control y no preocuparse de las reglas de un lenguaje de programación específico.

Un ejemplo básico de pseudocódigo, considerando el ejemplo utilizado anteriormente, es el siguiente:

```
INICIO FUNCION CALCULAR_AREA
  DEFINIR BASE: 5
  DEFINIR ALTURA: 10
  DEFINIR AREA: BASE * ALTURA
  DEVOLVER AREA
FIN
```

Lenguajes de Programación

Concepto

Los lenguajes de programación son lenguajes que pue den ser entendidos y procesados por la computadora.

Tipos de Lenguajes de Programación

Los principales tipos de lenguajes utilizados en la actualidad son tres:

- Lenguajes máquina
- Lenguaje de bajo nivel (ensamblador)
- Lenguajes de alto nivel

La elección del lenguaje de programación a utilizar depende mucho del objetivo del software. Por ejemplo, para desarrollar aplicaciones que deben responder en tiempo real como, por ejemplo, el control de la velocidad crucero en un sistema de navegación de un auto[1], debemos tener mayor control del hardware disponible, por lo que privilegiaremos lenguajes de más bajo nivel que nos permitan hacer un uso más eficiente de los recursos. En cambio, para aplicaciones de escritorio como sistemas de gestión de productos, calendarios, correo electrónico, entre otras privilegiaremos la elección de lenguajes de más alto nivel que nos permitan ser más eficientes en cuanto a la codificación ya que, en términos generales, es necesario escribir menos líneas de código en los lenguajes de alto nivel, que para su equivalente en bajo nivel.

El lenguaje máquina

Los lenguajes máquina son aquellos que están escritos en lenguajes cuyas instrucciones son cadenas binarias (cadenas o series de caracteres -dígitos- O y 1) que especifican una operación, y las posiciones (dirección) de memoria implicadas en la operación s e denominan instrucciones de máquina o código máquina. El código máquina es el conocido código binario.

En los primeros tiempos del desarrollo de los ordenadores era necesario programarlos directamente de esta forma, sin embargo, eran máquinas extraordinariamente limitadas, con muy pocas instrucciones por lo que aún era posible; en la actualidad esto es completamente irrealizable por lo que es necesario utilizar lenguajes más fácilmente comprensibles para los humanos que deben ser traducidos a código máquina para su ejecución.

Ejemplo de una instrucción:
1110 0010 0010 0001 0000 0000 0010 0000

El lenguaje de bajo nivel

Los lenguajes de bajo nivel son más fáciles de utilizar que los lenguajes máquina, pero, al igual, que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el ensamblador o assembler. Las instrucciones en lenguaje ensamblador son instrucciones conocidas como mnemotécnicas (mnemonics). Por ejemplo, nemotécnicos típicos de operaciones aritméticas son: en inglés, ADD, SUB, DIV, etc.; en español, SUM, para sumar, RES, para restar, DIV, para dividir etc.

[1] **Control de velocidad crucero**: El control de velocidad, también conocido como regulador de velocidad o auto crucero, es un sistema que controla de forma automática el factor de movimiento de un vehículo de motor. El conductor configura la velocidad y el sistema controlará la válvula de aceleración del vehículo para mantener la velocidad de forma continua.

Lenguajes de alto nivel

Los lenguajes de alto nivel son los más utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Otra razón es que un programa escrito en lenguaje de alto nivel es independiente de la máquina; esto es, las instrucciones del programa de la computadora no dependen del diseño del hardware o de una computadora en particular. En consecuencia, los programas escritos en lenguaje de alto nivel son portables o transportables, lo que significa la posibilidad de poder ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras; al contrario que los programas en lenguaje máquina o ensamblador, que sólo se pueden ejecutar en un determinado tipo de computadora. Esto es posible porque los lenguajes de alto nivel son traducidos a lenguaje máquina por un tipo de programa especial denominado "compilador". Un compilador toma como entrada un algoritmo escrito en un lenguaje de alto nivel y lo convierte a instrucciones inteligibles por el ordenador; los compiladores deben estar adaptados a cada tipo de ordenador pues deben generar código máquina específico para el mismo.

Ejemplos de Lenguajes de Alto Nivel:

C, C++, Java, Python, Visual BASIC, C#, JavaScript

¿Qué es un Programa?

Es un algoritmo escrito en algún lenguaje de programación de computadoras.

Pasos para la construcción de un programa

Definición Del Problema

En este paso se determina la información inicial para la elaboración del programa. Es donde se determina qué es lo que debe resolverse con el computador, el cual requiere una definición clara y precisa.

Es importante que se conozca lo que se desea que realice la computadora; mientras la definición del problema no se conozca del todo, no tiene mucho caso continuar con la siguiente etapa.

Análisis Del Problema

Una vez que se ha comprendido lo que se desea de la computadora, es necesario definir:

- Los datos de entrada.
- Los datos de salida
- Los métodos y fórmulas que se necesitan para procesar los datos.

Una recomendación muy práctica es la de colocarse en el lugar de la computadora y analizar qué es lo que se necesita que se ordene y en qué secuencia para producir los resultados esperados.

Diseño Del Algoritmo

Se puede utilizar algunas de las herramientas de representación de algoritmos mencionadas anteriormente. Este proceso consiste en definir la secuencia de pasos que se deben llevar a cabo para conseguir la salida identificada en el paso anterior.

Codificación

La codificación es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocódigo), en una serie de instrucciones detalladas, en un código reconocible por la computadora. La serie de instrucciones detalladas se conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

Prueba Y Depuración

Se denomina prueba de escritorio a la comprobación que se hace de un algoritmo para saber si está bien realizado. Esta prueba consiste en tomar datos específicos como entrada y seguir la secuencia indicada en el algoritmo hasta obtener un resultado, el análisis de estos resultados indicará si el algoritmo está correcto o si por el contrario hay necesidad de corregirlo o hacerle ajustes.

Algoritmos Fundamentales

Algoritmos de Ordenación

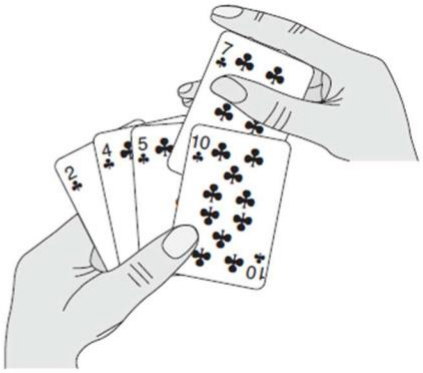
Los algoritmos de ordenación sirven para dar un orden determinado a los elementos de una lista. Este procedimiento de ordenación, mediante el cual se disponen los elementos del array en un orden especificado, tal como orden alfabético u orden numérico, es una tarea muy usual en la mayoría de los programas.

Un diccionario es un ejemplo de una lista ordenada alfabéticamente, y una agenda telefónica o lista de cuentas de un banco es un ejemplo de una lista ordenada numéricamente.

El orden de clasificación u ordenación puede ser ascendente (de menor a mayor) o descendente (de mayor a menor), por lo tanto, debe existir una función o característica de los elementos que determine su precedencia. Los ordenamientos eficientes son importantes para optimizar el uso de otros algoritmos (como los de búsqueda y fusión) que requieren listas ordenadas para una ejecución rápida. También es útil para poner datos en forma canónica y para generar resultados legibles por humanos.

Existen numerosos algoritmos de ordenación de listas: **inserción, burbuja, selección, rápido (quick sort), fusión (merge), montículo (heap), shell, etc.** Las diferencias entre estos algoritmos se basan en su eficiencia y en su **orden de complejidad** (es una medida de la dificultad computacional de resolver un problema). A continuación, describiremos algunos de ellos.

Ordenamiento por inserción



Este algoritmo es el más sencillo de comprender ya que es una representación natural de cómo aplicaríamos el orden a un conjunto de elementos. Supongamos que tenemos un mazo de cartas desordenadas, este algoritmo propone ir tomando las cartas de a una y luego ir colocándolas en la posición correcta con respecto a las anteriores ya ordenadas.

En términos generales, inicialmente se tiene un solo elemento, que por defecto es un conjunto ordenado. Después, cuando hay k elementos ordenados de menor a mayor, se toma el elemento k+1 y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha) o cuando ya no se encuentran elementos (todos los elementos fueron desplazados y este es el más pequeño). En este punto se inserta el elemento k+1 debiendo desplazarse los demás elementos.

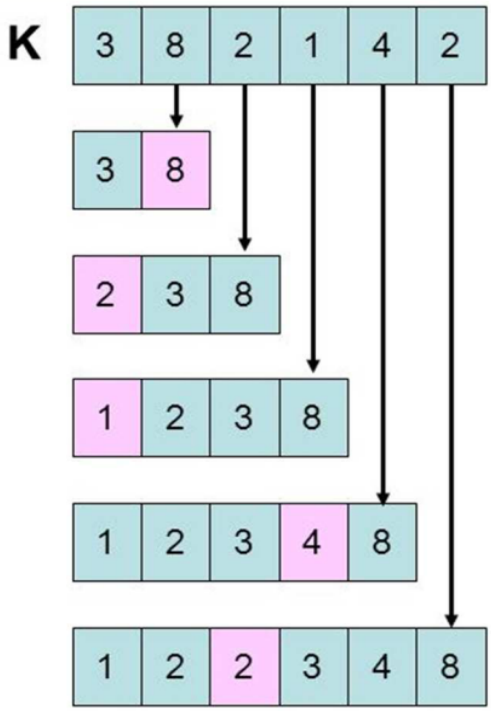


Figura 27: Ejemplo gráfico del algoritmo de ordenamiento por inserción

El pseudocódigo para este algoritmo es el siguiente:

```
INICIO inserción (A: lista de elementos)
   PARA (ENTERO i = 1; i < longitud(A); i++):
      ENTERO valor = A[i]
      ENTERO j = i-1

      MIENTRAS (j >= 0 && A[j] > valor)
      HACER:
         A[j+1] = A[j]
         j--
      FIN_MIENTRAS

      A[j+1] = valor
   FIN_PARA
FIN
```

Algoritmo de la burbuja

La ordenación por burbuja es uno de los métodos más fáciles de ordenación, ya que el algoritmo de ordenación utilizado es muy simple.

Este algoritmo consiste en comparar cada elemento de la lista con el siguiente (por parejas), si no están en el orden correcto, se intercambian entre sí sus valores. El valor más pequeño flota hasta el principio de la lista como si fuera una burbuja en un vaso de gaseosa.

A continuación, se muestra un ejemplo gráfico de este algoritmo, considerando la siguiente lista inicial:

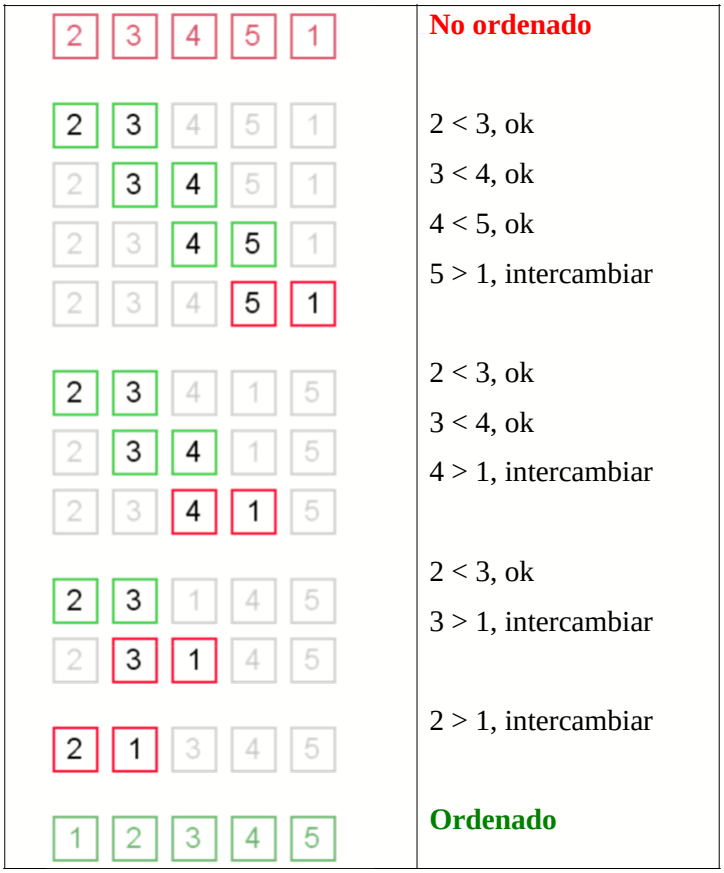


Figura 28: Pasos para ordenar una lista con el método de la burbuja

El pseudocódigo para este algoritmo es el siguiente:

```
INICIO burbuja (A: lista de elementos)
  n = longitud(A)
  HACER:
    intercambiado = falso

    PARA (ENTERO i = 1; i < n; i++)
      // si este par no está ordenado
      SI (A[i-1] > A[i]) ENTONCES:
        // los intercambiamos y recordamos que algo ha cambiado
        ENTERO Aux = A[i-1]
        A[i-1] = A[i]
        A[i] = Aux

        intercambiado = verdadero
      FIN_SI
    FIN_PARA
  MIENTRAS (intercambiado == verdadero)
FIN
```

Es importante notar que la recorrida completa de la lista (determinada por la sentencia PARA del pseudocódigo) será ejecutada hasta que intercambiado deje de ser verdadero, es decir que seguiremos

recorriendo la lista e intercambiando elementos desordenados hasta que no encontremos ninguno más fuera de orden.

Ordenamiento por selección

El algoritmo de ordenamiento por selección es similar al método de la burbuja y funciona de la siguiente manera: inicialmente se recorre toda la lista buscando el menor de todos los elementos, una vez terminada la recorrida el menor elemento se coloca al inicio de la lista recorrida. En la siguiente iteración se recorre nuevamente la lista, pero comenzando en el segundo elemento (ya que al haber insertado el menor encontrado al inicio ya lo consideramos ordenado). El procedimiento continúa hasta que el último elemento recorrido es el menor de su subconjunto.

Una desventaja de este algoritmo con respecto a los anteriores mencionados es que no mejora su rendimiento cuando los datos ya están ordenados o parcialmente ordenados debido a que necesariamente recorre la lista en busca del menor de los datos aun cuando el primero de ellos ya es el menor a encontrar.

El pseudocódigo de este algoritmo es muy similar al de la burbuja:

```
INICIO selección (A: lista de elementos)
  n = longitud(A)

  PARA (ENTERO i = 0; i < n - 1; i++)
    ENTERO mínimo = i

    PARA (ENTERO j = i+1; j < n; j++)
      // si este par no está ordenado
      SI (A[j] < A[mínimo]) ENTONCES:
        //encontramos un nuevo mínimo
        mínimo = j
      FIN_SI
    FIN_PARA

    // intercambiamos el actual con el mínimo
    encontrado ENTERO Aux = A[mínimo]
    A[mínimo] =
    A[i] A[i] = Aux
  FIN_PARA
FIN
```

Algoritmo quick-sort

Esta es la técnica de ordenamiento más rápida conocida, desarrollada por C. Antony R. Hoare en 1960. El algoritmo original es recursivo, pero se utilizan versiones iterativas para mejorar su rendimiento (los algoritmos recursivos son en general más lentos que los iterativos, y consumen más recursos). Tiene la propiedad de trabajar mejor para elementos de entrada desordenados completamente que para elementos semiordenados. Esta situación es precisamente la opuesta al ordenamiento de burbuja o al de selección antes mencionados.

Este tipo de algoritmos se basa en la técnica "divide y vencerás", lo que supone que es más rápido y fácil ordenar dos arreglos o listas de datos pequeños, que un arreglo o lista más grande.

El algoritmo trabaja de la siguiente forma:

- Elegir un elemento de la lista de elementos a ordenar, al que llamaremos **pivote**.
- Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el **pivote** ocupa exactamente el lugar que le corresponderá en la lista ordenada.
- La lista queda separada en dos **sublistas**, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
- Repetir este proceso de forma recursiva para cada **sublista** mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido, algunas alternativas son:

- Tomar un elemento cualquiera como pivote, tiene la ventaja de no requerir ningún cálculo adicional, lo cual lo hace bastante rápido.
- Otra opción puede ser recorrer la lista para saber de antemano qué elemento ocupará la posición central de la lista, para elegirlo como pivote. No obstante, el cálculo adicional rebaja bastante la eficiencia del algoritmo en el caso promedio.
- La opción a medio camino es tomar tres elementos de la lista - por ejemplo, el primero, el segundo, y el último - y compararlos, eligiendo el valor del medio como pivote.

Algoritmos de Búsqueda

La búsqueda de un elemento dado en una lista es una aplicación muy usual en el desarrollo de programas. Dos algoritmos típicos que realizan esta tarea son la búsqueda **secuencial** o en serie y la búsqueda **binaria** o dicotómica. La búsqueda secuencial es el método utilizado para listas no ordenadas, mientras que la búsqueda binaria se utiliza en listas que ya están ordenados.

Búsqueda secuencial

Este algoritmo busca el elemento dado, recorriendo secuencialmente la lista desde un elemento al siguiente, comenzando en la primera posición de la lista y se detiene cuando encuentra el elemento buscado o bien se alcanza el final de la lista sin haberlo encontrado.

Por consiguiente, el algoritmo debe comprobar primero el elemento almacenado en la primera posición de la lista, a continuación, el segundo elemento y así sucesivamente, hasta que se encuentra el elemento buscado o se termina el recorrido de la lista. Esta tarea repetitiva se realiza con bucles, en nuestro caso con el bucle Para (en inglés, for).

Consideremos que tenemos una lista de alumnos de un curso de Programación y queremos saber si el alumno ‘Pedro Lopez’, se encuentra cursando el mismo, entonces debemos recorrer toda la lista y buscar el nombre ‘Pedro Lopez’, e indicar si se encontró o no el alumno buscado.

```
INICIO busquedaSecuencial (L: lista de alumnos, a: alumno buscado)
  ENTERO n = longitud(L)
  BOOLEAN seEncontró= falso;
  // recorremos la lista, revisando cada elemento de la misma, para ver
  // sí es el alumno a.

  PARA (ENTERO i = 1; i < n - 1; i++)
  // comparamos el alumno de la posición actual con el alumno buscado: a
     SI (L[i] == a) ENTONCES:
       // encontramos el alumno
       buscado seEncontró = verdadero;

// si nunca se cumple L[i] == a, entonces la variable que indica si se
// encontró o no el alumno: seEncontró, quedará valiendo falso.
  FIN_PARA
FIN
```

Búsqueda Binaria

Este algoritmo se utiliza cuando disponemos de una lista ordenada, lo que nos permite facilitar la búsqueda, ya que podemos ir disminuyendo el espacio de búsqueda a segmentos menores a la lista original y completa.

La idea es no buscar en aquellos segmentos de la lista donde sabemos que el valor seguro que no puede estar, considerando que la lista esta ordenada.

Pensemos en el ejemplo anterior de la lista de alumnos del curso de Programación, si tenemos la lista ordenada alfabéticamente por Apellido, podemos comenzar la búsqueda considerando la lista completa y evaluar un valor central de la misma, es probable que ese valor central no sea el buscado, pero podemos ver si ese valor central es mayor o menor al alumno buscado. Si nuestro alumno buscado se llama: 'Lopez Pedro', y el alumno que corresponde a la posición central de la lista es: 'Martinez Sofia', entonces sabemos que como la lista esta ordenada alfabéticamente por apellido, 'Lopez Pedro', efectivamente tiene que estar en el segmento de la lista que es la primera mitad de la misma, es decir que hemos reducido el espacio de búsqueda a la mitad, lo cual hace que encontremos el valor más rápido que si lo buscaríamos en toda la lista, recorriendo todos los elementos. Si proseguimos con este procedimiento y continuamos buscando el valor central de cada segmento obtenido, podemos ir reduciendo cada vez más el espacio de búsqueda hasta llegar al elemento buscado, si es que existe en la lista.

Para explicar el algoritmo de búsqueda binaria, consideremos que queremos ver si se encuentra en una lista el número 19, a partir de una lista que contiene 12 números ordenados de menor a mayor, como se muestra a continuación:

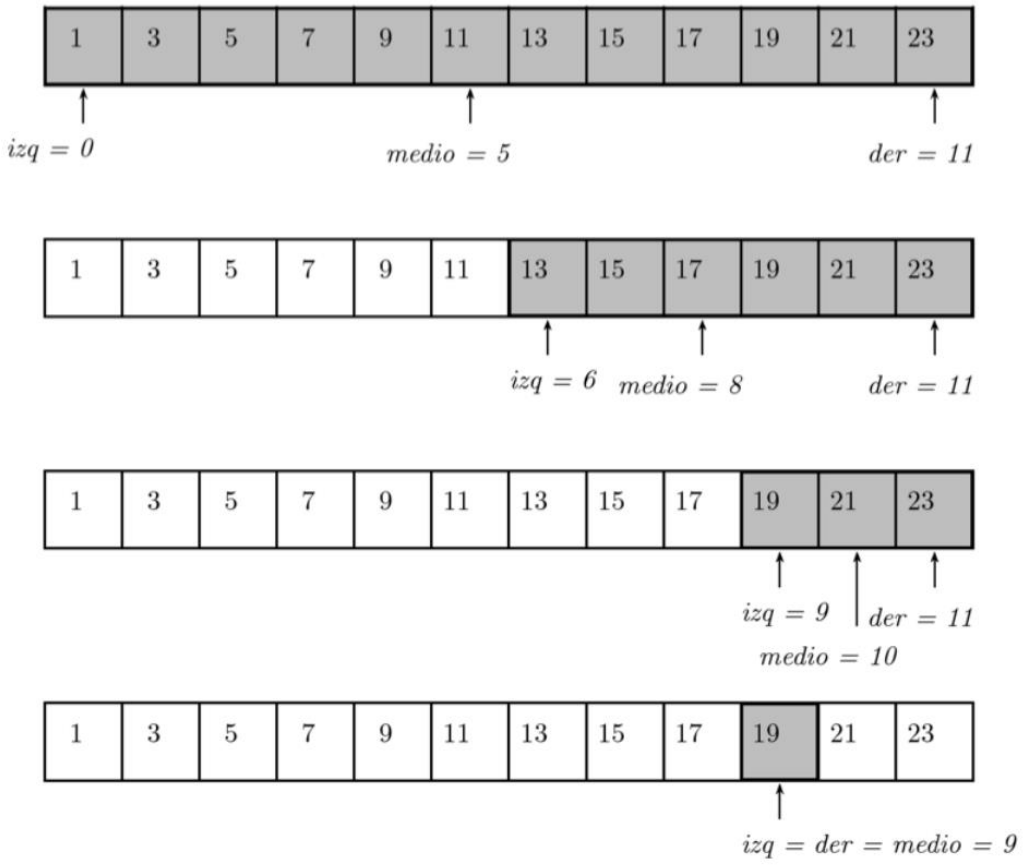


Figura 29: Búsqueda binaria sobre una lista de enteros

Para poder realizar la búsqueda binaria debemos:

- Primero, conocer cuál es el valor del índice izquierdo, derecho y del medio, de la siguiente forma:
  - `Índice` izq = 0; //sabemos que las listas comienzan con índices enumerados desde 0.
  - Índice der = Longitud de la lista inicial -1; // en este caso la longitud es 12, o sea, que el índice derecho valdrá 11.
  - Índice medio = (izq + der) / 2 // en este caso, (0 + 11) /2, considerando solo la parte entera de la división valdrá 5.
- A partir de la definición de estos índices, el siguiente paso es preguntar si en la posición del medio se encuentra el elemento buscado, es decir si Lista(medio)==19
- Si Lista(medio)==19, devuelve verdadero, entonces la búsqueda finaliza rápidamente.
- Si Lista(medio)==19, devuelve falso, entonces debemos preguntar si el valor de la lista en la posición medio es mayor o menor al valor buscado, p ara así saber si el segmento que nos interesa es del medio hacia la izquierda o del medio hacia la derecha. En este caso: Lista(medio) es menor a 19. Entonces el segmento que nos interesa de la lista es del medio (sin incluir, porque ya evaluamos y el medio no es igual a 19) hacia la derecha.
- El siguiente paso es volver a realizar el procedimiento descripto, pero considerando sólo el segmento que comienza en el medio hacia la derecha: son los mismos pasos, pero para una nueva lista que es un segmento de la lista original. Entonces:
  - Índice izq = 6;
  - Índice der = 11
  - Índice medio = (izq + der) / 2 // en este caso, (6+ 11) /2, considerando solo la parte entera de la división valdrá 8.

Como se ve este algoritmo de búsqueda binaria, a diferencia del algoritmo de búsqueda secuencial, no recorre toda la lista, sino que acorta la lista en segmentos más pequeños sucesivamente, esto es muy ventajoso en el caso de tener listas con gran cantidad de elementos, es decir con millones de valores, en estos casos realizar una búsqueda secuencial lleva mucho tiempo y si la lista ya se encuentra ordenada es mucho más eficiente realizar una búsqueda binaria que secuencial.

Algoritmos de Recorrido

Para visualizar o consultar los datos almacenados en un árbol se necesita *recorrer* el árbol o *visitar* los nodos del mismo. Al contrario de las listas, los árboles binarios no tienen realmente un primer valor, un segundo valor, tercer valor, etc. Se puede afirmar que la raíz viene el primero, pero ¿quién viene a continuación? Existen diferentes métodos de recorrido de árbol ya que la mayoría de las aplicaciones binarias son bastante sensibles al orden en el que se visitan los nodos, de forma que será preciso elegir cuidadosamente el tipo de recorrido.

Un recorrido de un árbol binario requiere que cada nodo del árbol sea procesado (visitado) una vez y solo una en una secuencia predeterminada. Existen dos enfoques generales para la secuencia de recorrido, **profundidad** y **anchura**.

En el **recorrido en profundidad**, el proceso exige un camino desde el nodo raíz a través de un hijo, al descendiente más lejano del primer hijo antes de proseguir a un segundo hijo. En otras palabras, en el recorrido en profundidad, todos los descendientes de un hijo se procesan antes del siguiente hijo. Para saber cómo regresarnos, vamos guardando los nodos v visitados en una estructura de **pila**. Es por esto que se acostumbra programar esta búsqueda de forma recursiva, con lo que el manejo de la pila lo realiza el lenguaje de programación utilizado.

Haciendo un recorrido en profundidad recorreríamos los nodos en el siguiente orden:

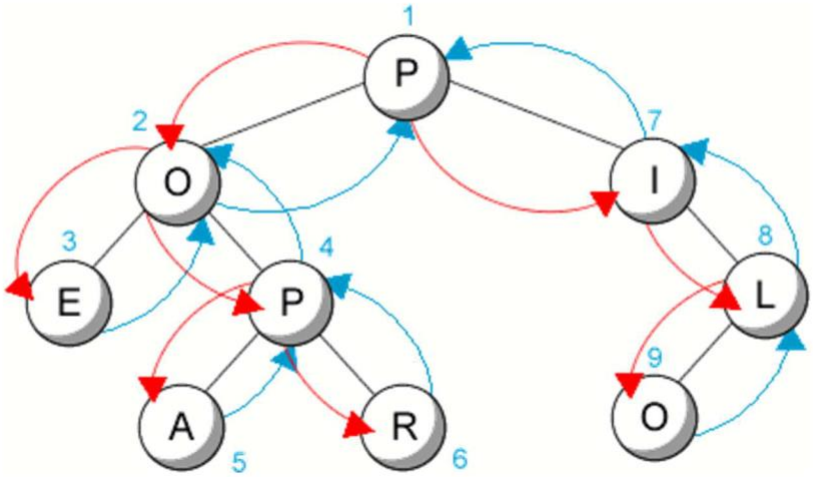


Figura 30: Recorrido en profundidad

En el **recorrido en anchura**, el proceso se realiza horizontalmente desde la raíz a todos sus hijos, a continuación, a los hijos de sus hijos y así sucesivamente hasta que todos los nodos han sido procesados. En otras palabras, en el recorrido en anchura, cada nivel se procesa totalmente antes de que comience el siguiente nivel. Para poder saber qué vértices visitar, utilizamos una **cola**.

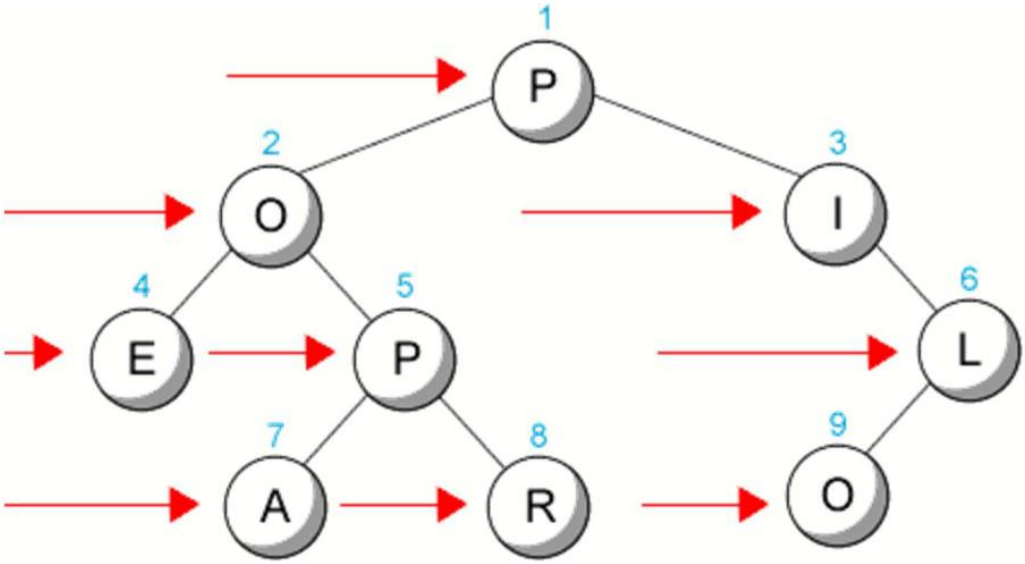


Figura 31: Recorrido en anchura

Fuentes de Información

- **Sommerville, Ian** - "I NGENIERÍA DE SOFTWARE" 9na Edición (Editorial Addison-Wesley Año 2011).
- **Pressman Roger** - "Ingeniería de Software" 7ma. Edición - (Editorial Mc Graw Hill Año 2010).
- **Jacobson, Booch y Rumbaugh** - "E L PROCESO UNIFICADO DE DESARROLLO" (Editorial Addison-Wesley - Año 2000 1ª edición).
- **Guerequeta Rosa, Vallecillo Antonio** - TÉCNICAS DE DISEÑO DE ALGORITMOS (Servicio de Publicaciones de la Universidad de Málaga, Año 1998)
- **Hernandez, Pier Paolo Guillen - ALGORITMOS** http://pier.guillen.com.mx/algorithms/09-busqueda/09.1-introduccion.htm
- **Curso de Estructuras de Datos y Algoritmos / Algoritmos recursivos** https://es.wikiversity.org/wiki/Curso_de_Estructuras_de_Datos_y_Algoritmos_/_Algoritmos_recursivo s
- **McConnell Steve** - CODE COMPLETE - Segunda edición (Editorial Microsoft Press, Año 2004)
- **Joyanes Aguilar Luis** PROGRAMACIÓN EN C++: ALGORITMOS, ESTRUCTURAS DE DAT OS Y OBJETOS - Segunda edición (Editorial McGraw-Hill, Año 2006)
- **Joyanes Aguilar Luis, Rodriguez Baena Luis, Fernandez Azuela Matilde** - Fundamentos de Programación
- **Frittelli Valerio** - Algoritmos y Estructuras de Datos - Segunda edición (Editorial Científica Universitaria, Año 2004)
- **Streib James T., Soma Takako** - GUIDE TO JAVA - A CONCISE INTRODUCTION TO PROGRAMMING (Editorial Springer, Año 2014)
- **Guttag John V.** - INTRODUCTION TO COMPUTATION AND PROGRAMMING USING PYTHON (Editorial MIT Press, Año 2013)
- **Ley de Moore** https://es.wikipedia.org/wiki/Ley_de_Moore
- **Tanenbaum Andrew S., Wetherall David J.** - REDES DE COMPUTADORAS - Quinta edición (Editorial Pearson, Año 2012)
- **Tanenbaum Andrew S.**- SISTEMAS OPERATIVOS MODERNOS - Tercera edición (Editorial Pearson, Año 2009)
- **Cola (Informática)**https://es.wikipedia.org/wiki/Cola_(inform%C3%A1tica)
- **Bubble Sort** https://en.wikipedia.org/wiki/Bubble_sort
- **Ordenamiento por Inserción** https://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n
- **Sorting algorithms/Insertion sort** http://rosettacode.org/wiki/Sorting_algorithms/Insertion_sort
- **Algoritmos de Ordenación** https://elbauldelprogramador.com/algoritmos-de-ordenacion/
- **Ordenamiento por Selección** https://es.wikipedia.org/wiki/Ordenamiento_por_selecci%C3%B3n
- **Ordenamiento rápido (Quicksort)** http://www.mis-algoritmos.com/ordenamiento-rapido-quicksort
- **Quicksort** https://es.wikipedia.org/wiki/Quicksort
- **Variables y Constantes** http://aurea.es/assets/2-tiposdatoslenguajec.pdf