Diseño de Algoritmos

Apuntes para clases

(Esto no pretende ser un libro)

José Canumán Chacón

Profesor Depto Ingeniería en Computación
Facultad de Ingeniería
Universidad de Magallanes
jcanuman @kataix

jose.canuman@umag.cl

Ver. 5.01

Índice

La palabra *algoritmo* proviene del nombre del matemático del siglo 9, Abū'Abd Allāh Muhammad bin Mūsā **al-Khwārizmī**, nacido en la antigua ciudad de Kwarizm , ahora llamada Khiva, en la provincia de Khorezm de Uzbekistan. Además escribió en Baghdad cerca del 820 DC, el famoso libro The Compendious Book on Calculation by Completion and Balancing (al-Kitāb al-Mukhtaṣar fī Ḥisāb **al-Jabr** wal-Muqābalah), que en su título contiene la palabra "álgebra", por eso también es citado como **padre del álgebra**.

Referencias

PARTE

I

Introduction to Algorithms by *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.*

Algorithms by *Robert Sedgewick and Kevin Wayne*

Data Structures and Algorithms by *Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, John Hopcroft, Ullman D. Jeffrey*

The Art of Computer Programming by *Donald Knuth*

Data Structures and Algorithm Analysis in C by *Mark Allen Weiss*

Introduction to Algorithms: A Creative Approach by *Udi Manber*

Handbook of Algorithms and Data Structures by *Gaston H. Gonnet, Ricardo Baeza-Yates*

Fundamentos de algoritmia by *Brassard, G.; Bratley, P.*

GeeksforGeeks
Handbook Baeza-Gonnet
Ejemplos MA Weiss

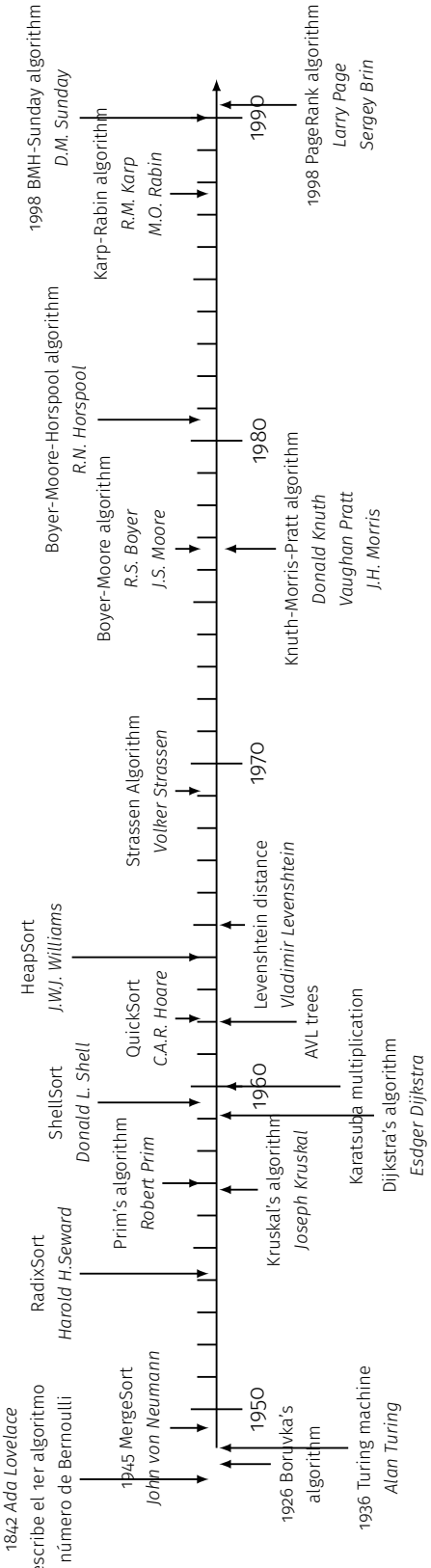
De que trata el curso:

Programación y resolución de problemas, con aplicaciones.
Algoritmo: método para resolver un problema y saber medir su eficiencia.
Uso de estructura de datos segun algoritmo.

Pero también:

Reutilizar código.
Entender un problema.
Saber aplicar un algoritmo a un problema.
Usar un método de diseño para aplicar un algoritmo.
Usar lenguaje técnico/algorítmico.
Aplicar cultura informática.
Diferenciar los conceptos de la implementación.

1	Metas
	Conocer algoritmos clásicos para resolver problemas no triviales. Hacer análisis de la eficiencia de los algoritmos y aplicar técnicas de diseño de los mismos.
2	Malla (Año 3|4/Semestre 1)
	ING (V): Diseño de algoritmos, Req: Estructuras de datos CIVIL 2003(VII): Diseño de algoritmos, Req: Estructuras de datos CIVIL 2020(VII): Diseño de algoritmos, Req: Estructuras de datos
3	Evaluación
	40 % Pruebas



PARTE

II

Introducción

Sección 1

Pero, ¿Qué es un algoritmo?

La idea intuitiva más general de un algoritmo es un procedimiento que consiste de un conjunto finito de instrucciones que, dada una entrada, nos permite obtener una salida si tal salida existe u obtener nada si no existe para esa entrada en particular. Todo esto a través de la ejecución sistemática de las instrucciones. Se requiere que un algoritmo se detenga en cada entrada, lo que implica que cada instrucción requiere una cantidad finita de tiempo, y cada entrada tiene una longitud finita.
Según **Dijkstra**(1971), un algoritmo se corresponde con una descripción de un patrón de comportamiento, expresado en términos de un conjunto finito de acciones.

Por ejemplo, considera el proceso de preparar café como un algoritmo:

- **Entrada:** Agua, café, cafetera.
- **Instrucciones:**
  1. Hervir agua.
  2. Colocar el café en la cafetera.
  3. Verter agua caliente en la cafetera.
  4. Esperar 5 minutos.
- **Salida:** Café listo para servir.

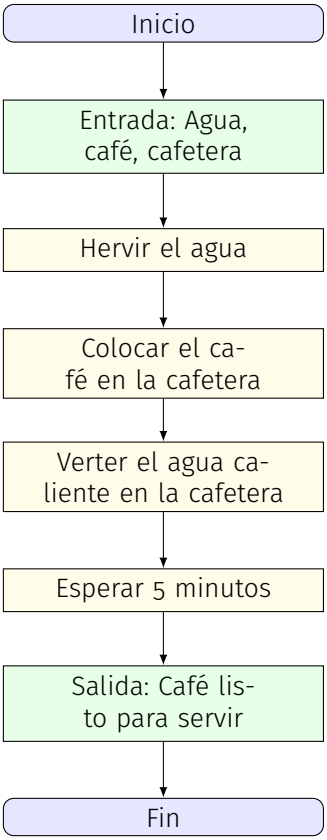


Figura 3. Descripción y diagrama de flujo para el proceso de preparar café.

También requerimos que la salida ante una entrada sea única, es decir, el algoritmo es determinista en el sentido de que ejecuta el mismo conjunto de instrucciones ante una entrada en particular.

Figura 1. Donald Ervin Knuth

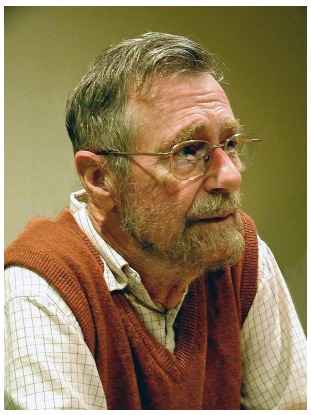


Figura 2. Edsger Wybe Dijkstra

Existe una definición formal dada por **Knuth**, donde indica que un algoritmo debería tener 5 propiedades:

Carácter finito: siempre debe terminar después de un número finito de pasos.

Precisión: cada paso debe ser preciso, no ambiguo.

Entrada: puede tener 0 o mas ingresos.

Salida: una o mas salidas, relacionada con la entrada.

Eficacia: todas las operaciones deben ser lo suficientemente básicas que terminan exactamente y de longitud finita.

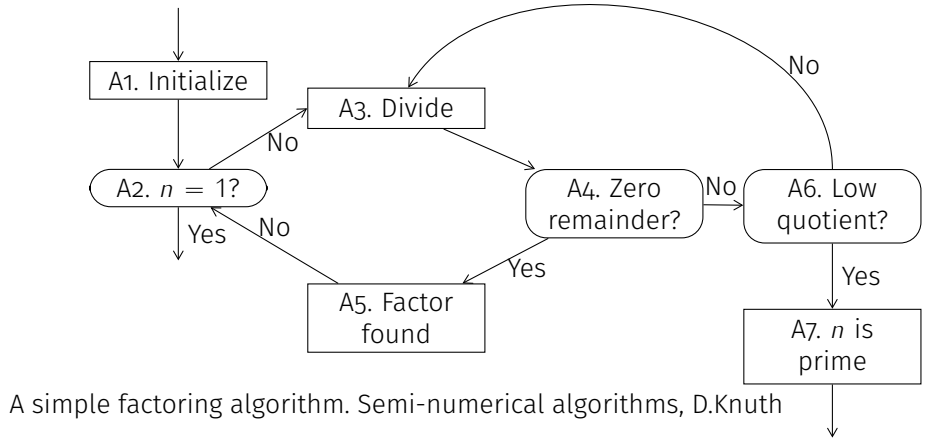


Figura 4. Un algoritmo simple de factorización. Algoritmos semi-numéricos, D. Knuth.

Subsección 1.1

Los algoritmos en la historia

Los algoritmos han sido una parte fundamental del desarrollo de las matemáticas y la informática a lo largo de la historia. A continuación, se presenta una breve referencia a su evolución a través de diferentes épocas:

1.1.1. Antigüedad

En la antigüedad, uno de los algoritmos más conocidos es el Algoritmo de Euclides, utilizado para calcular el máximo común divisor (MCD) de dos números. Este algoritmo, que data del siglo III a.C., es un ejemplo temprano de un procedimiento sistemático para resolver un problema matemático.

1.1.2. Edad Media

Durante la Edad Media, el matemático persa Al-Juarismi hizo contribuciones significativas al desarrollo de los algoritmos. Su obra, "El libro de la suma y el balanceo" (Al-Kitāb al-Mukhtaṣar fī ḥisāb al-jabr wa-l-muqābala), fue traducida al latín en el siglo XII por Robert de Chester. La traducción, titulada "Algoritmi de numero Indorum", fue fundamental para introducir el álgebra en Europa. Esta obra es una de las razones por las que el término "algoritmo" se deriva del nombre de Al-Juarismi.
Además, la obra introdujo métodos sistemáticos para resolver ecuaciones lineales y cuadráticas, y es considerada una de las bases del álgebra moderna.

1.1.3. Edad Moderna

En la Edad Moderna, el trabajo de Alan Turing en las máquinas computacionales sentó las bases de la informática teórica. Turing introdujo el concepto de la máquina de Turing, un modelo abstracto de computación que formaliza la noción de algoritmo y computabilidad. Su trabajo ha influido profundamente en el desarrollo de la teoría de la computación y en la creación de las computadoras modernas.
Estos hitos históricos muestran cómo los algoritmos han evolucionado desde simples procedimientos matemáticos hasta convertirse en el núcleo de la informática moderna.

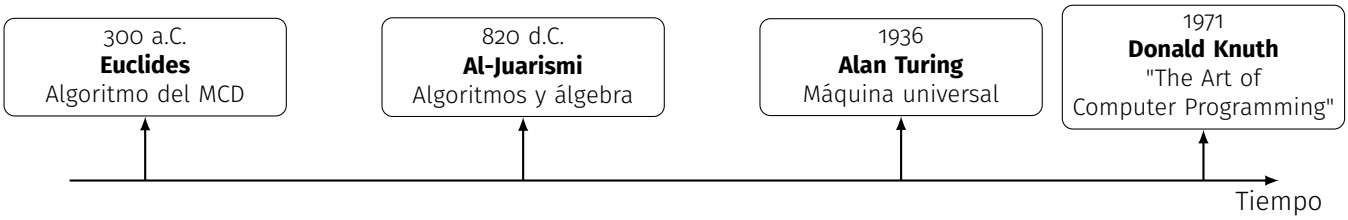


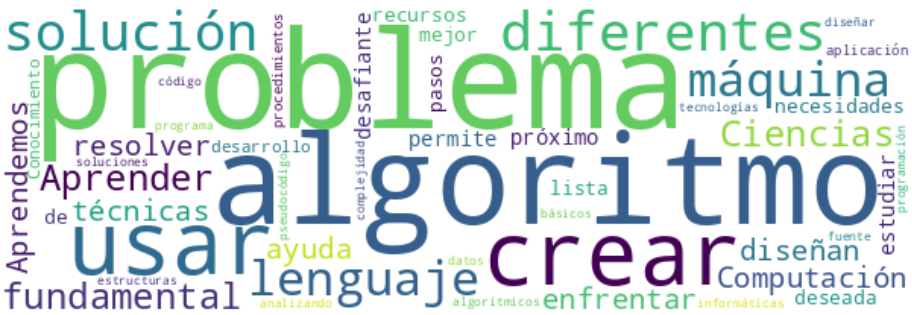
Figura 5. Línea del tiempo: evolución de los algoritmos.

¿Porqué estudiar Algoritmos?

- Los algoritmos son fundamentales en las Ciencias de la Computación.
- Nos ayudan a resolver problemas eficientemente.
- Facilitan el diseño de soluciones óptimas y reutilizables.
- Son la base de áreas avanzadas como Machine Learning, criptografía y optimización.
- Permiten usar mejor los recursos computacionales.

Además de las aplicaciones clásicas, los algoritmos son fundamentales en áreas como:

- **Machine Learning (ML):** Clasificación de datos, predicción y detección de patrones.
- **Optimización logística:** Planificación de rutas de entrega.
- **Ciberseguridad:** Encriptación y análisis de vulnerabilidades.
- **Procesamiento de señales:** Aplicaciones como compresión de audio e imágenes.



Subsección 1.2

Los límites de los algoritmos: P y NP

Complejidad Computacional: P, NP y NP-completo

En el ámbito de la teoría de la complejidad computacional, los problemas se clasifican en diferentes clases según la dificultad de resolverlos. Dos de las clases más importantes son P y NP.

Problemas P

Estos son problemas que pueden ser resueltos en tiempo polinómico por una máquina determinista. En otras palabras, existe un algoritmo que puede resolver cualquier instancia del problema en un tiempo que es una función polinómica del tamaño de la entrada. Un ejemplo clásico de un problema en P es la ordenación de una lista de números con el algoritmo QuickSort, que tiene complejidad $O(n \log n)$.

Problemas NP

Esta clase incluye problemas para los cuales, si se proporciona una solución, esta puede ser verificada en tiempo polinómico por una máquina determinista. Sin embargo, no se sabe si todos los problemas en NP pueden ser resueltos en tiempo polinómico. Un ejemplo de un problema en NP es el problema de la satisfacibilidad booleana (SAT).

Problemas NP-completos

Estos son los problemas más difíciles dentro de NP. Un problema es NP-completo si es al menos tan difícil como cualquier otro problema en NP, lo que significa que si se encuentra un algoritmo en tiempo polinómico para resolver un problema NP-completo, entonces todos los problemas en NP pueden ser resueltos en tiempo polinómico. El problema del viajante (TSP) es un ejemplo clásico de un problema NP-completo. En este problema, se busca encontrar el camino más corto que visita un conjunto de ciudades exactamente una vez y regresa a la ciudad de origen.

Importancia en la informática

La distinción entre P y NP es fundamental en la informática teórica porque aborda la cuestión de si todos los problemas cuya solución puede ser verificada rápidamente también pueden ser resueltos rápidamente. Esta es una de las preguntas abiertas más importantes en la teoría de la computación. En la práctica, muchos problemas reales son NP-completos, y encontrar soluciones eficientes para ellos tiene un impacto significativo en campos como la optimización, la logística y la inteligencia artificial.
La relación entre P y NP sigue siendo una de las preguntas más importantes en la informática teórica. Si alguien prueba que $P = NP$ o $P \neq NP$, cambiaría drásticamente cómo entendemos el mundo de los algoritmos.

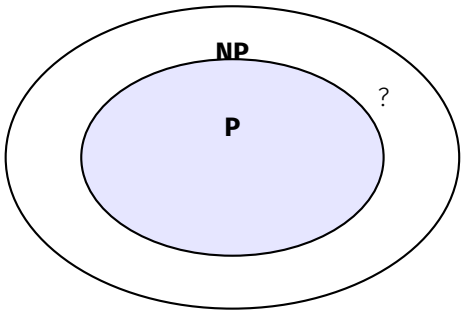


Figura 6. Relación entre las clases P y NP.

PARTE

III

Análisis

Sección 2

¿Cómo medir un algoritmo?

Generalmente un problema computacional puede resolverse mediante diversos algoritmos o programas. El análisis de algoritmos es una actividad muy importante, especialmente en entornos con recursos restringidos. Es necesario para:

- Comparar algoritmos distintos
- Predecir el comportamiento de un algoritmo en circunstancias extremas
- Ajustar los parámetros de un algoritmo para obtener los mejores resultados

El análisis puede ser empírico(experimental) o teórico.
Eficiencia: capacidad de resolver el problema propuesto empleando un bajo consumo de recursos computacionales.

Costo espacial, costo temporal

En ocasiones el tiempo y memoria son recursos competitivos y un buen algoritmo es aquel que resuelve el problema con un buen compromiso tiempo/memoria.

Sección 3

Orden asintótico

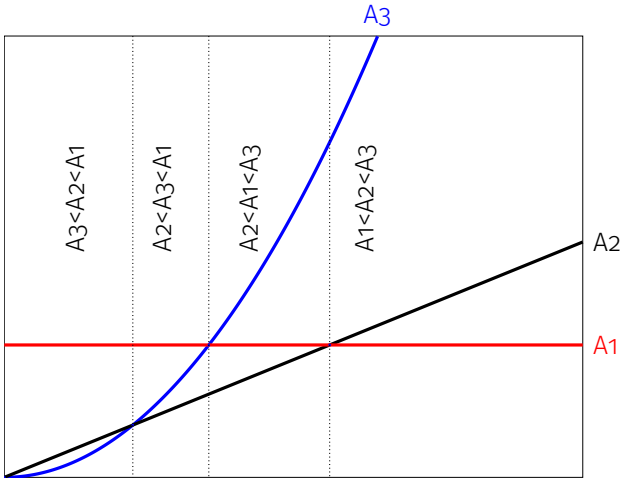
Supongamos que tenemos 3 algoritmos que calculan lo mismo.

```c
//A1.c
#include <stdio.h>

int main(){
  int n,m;

  scanf("&d",n);
  m=n*n;

  printf("%d",m);
}
```

```c
//A2.c
#include <stdio.h>

int main(){
  int i,n,m=0;

  scanf("&d",n);
  for(i=1; i <=n;i++)
     m=m+n;

  printf("%d",m);
}
```

```c
//A3.c
#include <stdio.h>

int main(){
  int i,j,n,m=0;

  scanf("&d",n);
  for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
       m=m+1;

  printf("%d",m);
}
```

Intuitivamente parece que el mejor programa es A1 y el peor A3. Pero, ¿cuál es el mejor?
En general tendremos un comportamiento relativo tal como:



Una buena caracterización del costo computacional debería permitir establecer la calidad de un programa con independencia tanto del hardware como del tamaño de la entrada.
Ventajas de la caracterización asintótica:

- Generalmente los programas sólo son útiles para resolver problemas de gran tamaño (si es pequeño podríamos resolverlos manualmente sin dificultad)
- Al considerar sólo tamaños grandes, se pueden hacer aproximaciones sencillas que simplifican considerablemente el análisis del costo.
- La bondad relativa de distintos programas ya no depende de los valores concretos de los tiempos de ejecución de las distintas operaciones elementales empleadas (siempre que éstos no dependan del tamaño), ni del tamaño concreto de las instancias del problema a resolver.

Para simplificar el análisis del costo, generalmente se utiliza el concepto de *paso*: Un PASO es la ejecución de un segmento de código cuyo tiempo de proceso no depende del tamaño del problema considerado, o bien está acotado por alguna constante.
Costo computacional de un programa: Número de PASOS en función del tamaño del problema. Construcciones a las que se asigna un PASO:

- Asignación, operaciones aritméticas o lógicas, comparación, acceso a un elemento de vector o matriz, etc.
- Cualquier secuencia finita de estas operaciones cuya longitud no dependa del tamaño.

Construcciones a las que no se le puede asignar un PASO, sino un número de PASOS en función del tamaño:

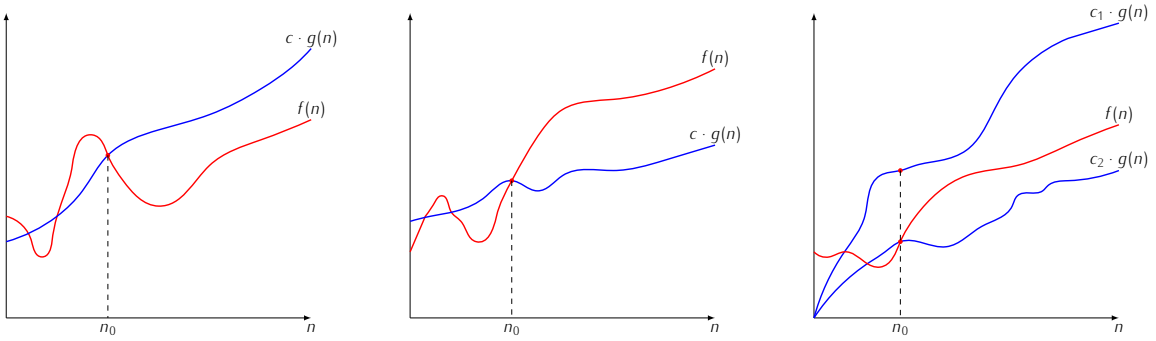
- Asignación de variables estructuradas (ej. vectores) cuyo número de elementos dependa del tamaño
- Ciclos cuyo número de iteraciones dependa del tamaño

La comparación de funciones de costo debe ser insensible a "constantes de implementación" tales como los tiempos concretos de ejecución de operaciones individuales (cuyo costo sea independiente del tamaño). La independencia de las constantes se consigue considerando sólo el comportamiento asintótico de la función de costo (es decir, para tamaños "grandes"). A menudo ocurre que, para una tamaño dada, hay

diversas instancias con costos diferentes, por lo que el costo no puede expresarse propiamente como función de la tamaño. En estas ocasiones conviene determinar cotas superiores e inferiores de la función costo; es decir, en el mejor caso y en el peor caso. Hay situaciones en las que incluso las cotas para mejores y peores casos son funciones complicadas. Generalmente bastará determinar funciones simples que acoten superior e inferiormente los costos de todas las instancias para tamaños grandes.

Notación asintótica

Abstracción de las constantes asociadas al concepto de "PASO", de la noción de tamaños "grandes" y de la de "cotas asintóticas":

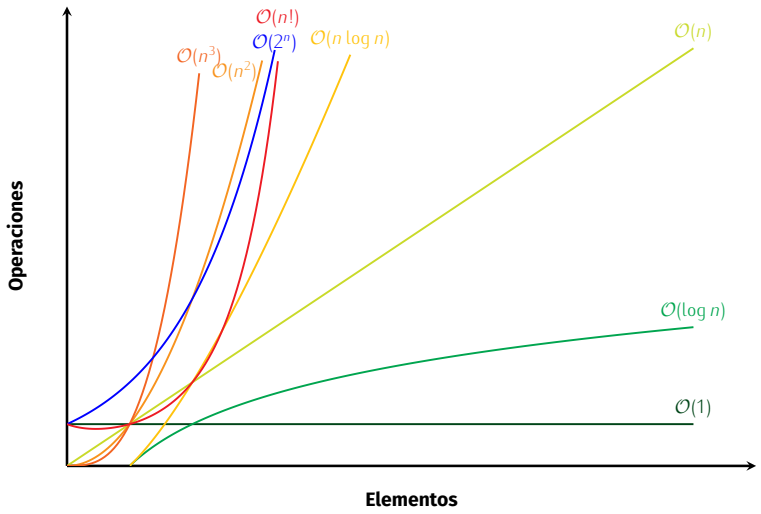


$$\mathcal{O}(g) = \{f : \mathbb{N} \to \mathbb{R}_0^+ | \exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N}, \forall n \geq n_0, \ f(n) \leq cg(n)\}$$
$$\Omega(g) = \{f : \mathbb{N} \to \mathbb{R}_0^+ | \exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N}, \forall n \geq n_0, \ f(n) \geq cg(n)\}$$
$$\Theta(g) = \{f : \mathbb{N} \to \mathbb{R}_0^+ | \exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N}, \forall n \geq n_0, \ c_1 g(n) \geq f(n) \geq c_2 g(n)\}$$

Se dice que $f(n) = \mathcal{O}(g(n))$.
Se dice que $f(n) = \Omega(g(n))$.
Se dice que $f(n) = \Theta(g(n))$.

Subsección 3.1
Simplificación

$$\mathcal{O}(c) = \mathcal{O}(1), c \in \mathbb{R}^+$$
$$\mathcal{O}(cf) = \mathcal{O}(f), c \in \mathbb{R}^+$$
$$\mathcal{O}(f + g) = \mathcal{O}(max\{f, g\})$$
$$\mathcal{O}\left(\sum_{i=0}^{k} c_i n^i\right) = \mathcal{O}(n^k)$$



Notación asintótica: jerarquía de costos computacionales

Sección 4

Análisis: Estimación de la ejecución

El tiempo de un algoritmo puede medirse en número de operaciones(comparaciones en otros casos). En forma empírica puede ser contando o midiendo el tiempo de las operaciones.

```
1 count = count+1;
2 sum = sum + count;
```

line	cost	times
1	c_1	1
2	c_2	1

$T(n) = c_1 + c_2$

```
1 if n<0
2    count = 0;
3  else
4    count = 1;
```

line	cost	times
1	c_1	1
2	c_2	1
4	c_3	1

$T(n) = c_1 + max(c_2, c_3)$

```
1 i=1;
2 sum = 0;
3 while(i <= n){
4   i = i+1;
5   sum = sum + i;
6 }
```

line	cost	times
1	c_1	1
2	c_2	1
3	c_3	$n+1$
4	c_4	n
5	c_5	n

$T(n) = c_1 + c_2 + (n+1) \cdot c_3 + n \cdot c_4 + n \cdot c_5$

```
1  i=1;
2  sum = 0;
3  while(i <= n){
4    j=1;
5    while (j <= n){
6    sum = sum + i;
7    j=j+1;
8    }
9    i = i+1;
10 }
```

line	cost	times
1	c_1	1
2	c_2	1
3	c_3	$n+1$
4	c_4	n
5	c_5	$n \cdot (n+1)$
6	c_6	$n \cdot n$
7	c_7	$n \cdot n$
9	c_8	n

$T(n) = c_1 + c_2 + (n+1) \cdot c_3 + n \cdot c_4 + (n+1) \cdot c_5 + n^2 \cdot c_6 + n^2 \cdot c_7 + n \cdot c_8$

Subsección 4.1

Peor caso, mejor caso y promedio

A menudo el costo NO es (sólo) función de la tamaño. En los ejemplos vistos hasta ahora, todas las "instancias" de una tamaño dada tenían el mismo costo computacional. Pero esto no es siempre así.

```
1  void
2  InsertionSort( int A[ ], int N ){
3    int j, P;
4    int Tmp;
5
6    for( P = 1; P < N; P++ ){
7      Tmp = A[ P ];
8      for( j = P; j > 0 && A[ j - 1 ] > Tmp; j-- )
9        A[ j ] = A[ j - 1 ];
10     A[ j ] = Tmp;
11   }
12 }
```

line	cost	times
6	c_1	n
7	c_2	$n-1$
8	c_3	$\sum_{j=1}^{n} j$
9	c_4	$\sum_{j=1}^{n} (j-1)$
10	c_5	$n-1$

$$T(n) = c_1 \cdot n + c_2 \cdot (n-1) + c_3 \cdot \sum_{j=1}^{n} j + c_4 \cdot \sum_{j=1}^{n} (j-1) + c_5 \cdot (n-1)$$

$$T(n) = c_1 \cdot n + c_2 \cdot n - c_2 + c_3 \cdot \sum_{j=1}^{n} j + c_4 \cdot \sum_{j=1}^{n} j - c_4 \cdot \sum_{j=1}^{n} 1 + c_5 \cdot n - c_5$$

Supongamos que las asignaciones tienen el mismo costo, esto es: $c_2 = c_4 = c_5$,

$$T(n) = c_1 \cdot n + c_2 \cdot (n-1) + c_3 \cdot \sum_{j=1}^{n} j + c_2 \cdot \sum_{j=1}^{n} (j-1) + c_2 \cdot (n-1)$$

$$T(n) = c_1 \cdot n + c_2 \cdot \left(2 \cdot (n-1) + \sum_{j=1}^{n} (j-1)\right) + c_3 \cdot \sum_{j=1}^{n} j$$

Es claro que este tiempo no es independiente de las entradas dadas, Luego tenemos:

mejor caso: corresponde a cuando se realiza un número mínimo de operaciones. En este caso ocurre cuando el arreglo A ya está ordenado, y en ese caso el ciclo

interno no se ejecuta. Entonces:

$$T(n) = c_1 \cdot n + 2 \cdot c_2 \cdot (n-1)$$
$$T(n) = (c_1 + 2c_2)n - 2c_2$$
$$T(n) = a \cdot n + b \in \mathcal{O}(n)$$

peor caso: corresponde cuando se ejecuta el número máximo de operaciones. En este caso, cuando el ciclo interno se ejecuta todas las veces posibles. En este caso se tiene:

$$T(n) = c_1 \cdot n + c_2 \cdot \left(2 \cdot (n-1) + \sum_{j=1}^{n}(j-1)\right) + c_3 \cdot \sum_{j=1}^{n} j$$
$$T(n) = \frac{c_2 + c_3}{2}n^2 + \left(c_1 + \frac{3c_2 + c_3}{2}\right)n - 2c_2$$
$$T(n) = a \cdot n^2 + b \cdot n + c \in \mathcal{O}(n^2)$$

casos típicos o promedios: corresponde cuando se da una instancia aleatoria generada uniformemente entre todas las posibles. Para este caso podemos pensar que cada una de las iteraciones del ciclo interno se ejecuta la mitad de las operaciones posibles, es decir, que el valor actual queda aproximadamente en el medio del arreglo a su izquierda. En este caso se tiene:

$$T(n) = c_1 \cdot n + c_2 \cdot \left(2 \cdot (n-1) + \sum_{j=1}^{n}\frac{j-1}{2}\right) + c_3 \cdot \sum_{j=1}^{n}\frac{j}{2}$$
$$T(n) = \frac{c_2 + c_3}{4}n^2 + \left(c_1 + \frac{7c_2 + c_3}{4}\right)n - 2c_2$$
$$T(n) = a \cdot n^2 + b \cdot n + c \in \mathcal{O}(n^2)$$

```
1  for( i = 1; i <= n; i++ )
2     for( j = 1; j <= i; j++ )
3        for( k = 1; k <= j; k++ )
4              x=x+1;
```

line	cost	times
1	c_1	$n+1$
2	c_2	$\sum_{j=1}^{n} j+1$
3	c_3	$\sum_{j=1}^{n}\sum_{k=1}^{j} k+1$
4	c_4	$\sum_{j=1}^{n}\sum_{k=1}^{j} k$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot \sum_{j=1}^{n}(j+1) + c_3 \cdot \sum_{j=1}^{n}\sum_{k=1}^{j}(k+1) + c_4 \cdot \sum_{j=1}^{n}\sum_{k=1}^{j} k$$
$$T(n) = a \cdot n^3 + b \cdot n^2 + c \cdot n + d \in \mathcal{O}(n^3)$$

Subsección 4.2

Mínimo

$$T(n) = n - 1 \in O(n)$$

Algoritmo 1 Búsqueda del elemento mínimo

```
1: function Minimo(V)                                   ▷ V[0..n − 1]
2:     m ← V[0]
3:     for i ← 1 to n − 1 do
4:         if V[i] < m then
5:             m ← V[i]
6:     return m
```

4.2.1. Búsqueda secuencial

Algoritmo 2 Búsqueda secuencial #1

```
1: procedure bSec1(V, x)                                ▷ V[0..n − 1]
2:     r ← n
3:     i ← 0
4:     while i ≤ n − 1 ∧ r = n do
5:         if x = V[i] then
6:             r ← i
7:         i ← i + 1
8:     return r
```

Algoritmo 3 Búsqueda secuencial #2

```
1: procedure bSec2(V, x)                                ▷ V[0..n − 1]
2:     r ← 0
3:     while r ≤ n − 1 ∧ x ≠ V[r] do
4:         r ← r + 1
5:     return r
```

$t_{mín}(n) = 1 \in O(1)$
$t_{máx}(n) = n \in O(n)$
$\overline{t}(n) \in O(n)$

```c
//bseq2.c
#include <stdio.h>

int bseq2(int v[], int x, int n){
   int r=0;

   while((r <=n-1)&&(x != v[r])){
       r = r+1;
   }
   return r;
}

int main(){
    int a[] = {3,6,23,45,7,8,9,2,66};
    int x = 88;

    printf("%d\n",bseq2(a,x,9));
}
```

```
Compilación:
gcc bseq2.c -o bseq2
Ejecución:
$./bseq2
9
$
```

Claramente el mejor caso se produce cuando $r = 0$. Sólo se realiza una comparación. $t_{mín}(n) = 1 \in O(1)$. El peor caso se produce cuando $r = n - 1$ o $r = n$, en ambos casos se realizan n comparaciones, luego: $t_{máx}(n) = n \in O(n)$. En este caso se debe calcular el caso promedio, para ello se debe usar probabilidades.
Sea α la probabilidad de que el elemento esté, y $\beta = 1 - \alpha$ la probabilidad que no esté, luego:

$$\alpha = P(r \in [0, n-1]), \beta = P(r = n)$$

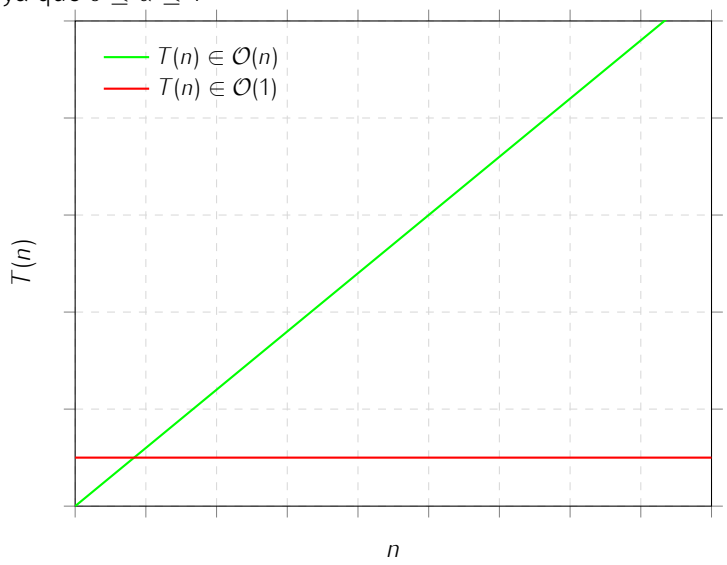
Hipótesis de equiprobabilidad:

$$P(r = 0) = P(r = 1) = \ldots = P(r = n-1) = \frac{\alpha}{n}$$

El número de comparaciones promedio es:

$$\begin{aligned}\bar{t}(n) &= \beta n + \sum_{i=0}^{n-1} \frac{\alpha}{n} i \\ &= (1-\alpha)n + \frac{\alpha}{n} \sum_{i=0}^{n-1} i \\ &= (1-\alpha)n + \frac{\alpha}{n} \left(\frac{n(n-1)}{2} \right) \\ &= \frac{(2-\alpha)n - \alpha}{2} \in O(n)\end{aligned}$$

ya que $0 \leq \alpha \leq 1$



Sección 5

Ordenación por comparación

Subsección 5.1

Intercambio

Bubble Sort
Método sencillo que consiste en revisar cada elemento del conjunto con el siguiente, intercambiándolos de posición si están en orden inverso. Es necesario revisar varias veces toda la lista hasta que ya no se necesiten intercambios. Este método también se llama por intercambio directo.

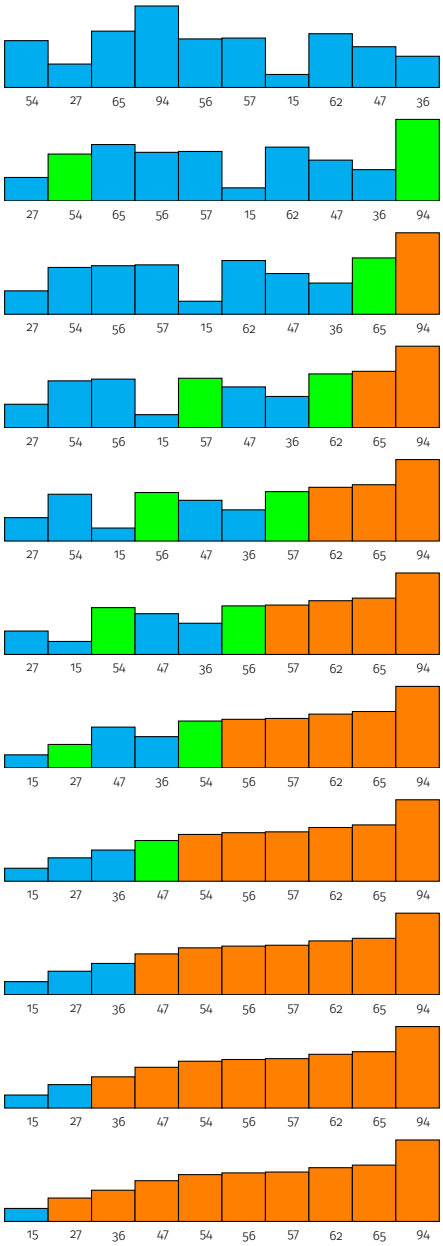
```c
void bubbleSort(int v[], int n) {
  int i, j;

  for (i = 0; i < n - 1; i++) {
    for (j = 0; j < n - i - 1; j++) {
       if (v[j] > v[j + 1])
          swap(&v[j], &v[j + 1]);
    }
  }
}
```

Algoritmo 4 Ordenación por intercambio

1: **procedure** SwapSort(V) ▷ $V[0..n-1]$
2: **for** $i \leftarrow 1$ to $n-1$ **do**
3: **for** $j \leftarrow n-1$ to i **do**
4: **if** $V[j] < V[j-1]$ **then**
5: $V[j] \leftrightarrow V[j-1]$ ▷ $swap(V[j], V[j-1])$

$T(n) \in O(n^2)$

```
 0: 54 27 65 94 56 57 15 62 47 36
 1: 27 54 65 56 57 15 62 47 36 94
 2: 27 54 56 57 15 62 47 36 65 94
 3: 27 54 56 15 57 47 36 62 65 94
 4: 27 54 15 56 47 36 57 62 65 94
 5: 27 15 54 47 36 56 57 62 65 94
 6: 15 27 47 36 54 56 57 62 65 94
 7: 15 27 36 47 54 56 57 62 65 94
 8: 15 27 36 47 54 56 57 62 65 94
 9: 15 27 36 47 54 56 57 62 65 94
    15 27 36 47 54 56 57 62 65 94
```



Subsección 5.2
Selección

Selection Sort
Este método consiste en seleccionar el menor elemento del conjunto y a continuación intercambiarlo con el elemento que ocupa la primera posición del vector. Hay que repetir esta operación con los $n-1$ elementos restantes, luego los $n-2$ elementos restantes y así sucesivamente hasta que solo quede un elemento.

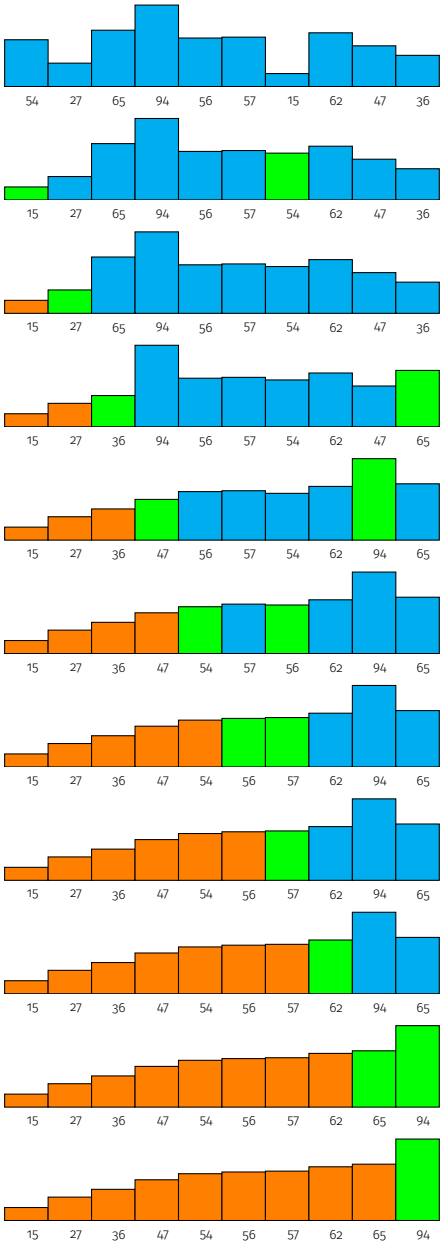
```
1  void selectionSort(int v[], int n) {
2    int i, j, min_idx;
3
4    for (i = 0; i < n - 1; i++) {
5    min_idx = i;
6    for (j = i + 1; j < n; j++)
7      if (v[j] < v[min_idx])
8      min_idx = j;
9
10   swap(&v[min_idx], &v[i]);
11   }
12 }
```

Algoritmo 5 Ordenación por Selección

```
1:  procedure SelectionSort(V)                    ▷ V[0..n − 1]
2:      for i ← 1 to n − 1 do
3:          k ← i − 1
4:          m ← V[i − 1]
5:          for j ← i to n − 1 do
6:              if V[j] < m then
7:                  k ← j
8:                  m ← V[j]
9:          V[k] ← V[i − 1]
10:         V[i − 1] ← m
```

$T(n) \in O(n^2)$

```
0: 54 27 65 94 56 57 15 62 47 36
1: 15 27 65 94 56 57 54 62 47 36
2: 15 27 65 94 56 57 54 62 47 36
3: 15 27 36 94 56 57 54 62 47 65
4: 15 27 36 47 56 57 54 62 94 65
5: 15 27 36 47 54 57 56 62 94 65
6: 15 27 36 47 54 56 57 62 94 65
7: 15 27 36 47 54 56 57 62 94 65
8: 15 27 36 47 54 56 57 62 94 65
9: 15 27 36 47 54 56 57 62 65 94
   15 27 36 47 54 56 57 62 65 94
```



$$t_{mín}(n) = n - 1 \in O(n)$$
$$t_{máx}(n) = \frac{n(n-1)}{2} \in O(n^2)$$
$$\bar{t}(n) \in O(n^2)$$

Subsección 5.3
Inserción

Insertion Sort
Método utilizado por los jugadores de cartas(naipes). En cada paso, a partir de $i = 2$, el i-ésimo elemento de la secuencia se procesa y se transfiere a la secuencia destino(que ya esta ordenada), insertándolo en el lugar correspondiente.

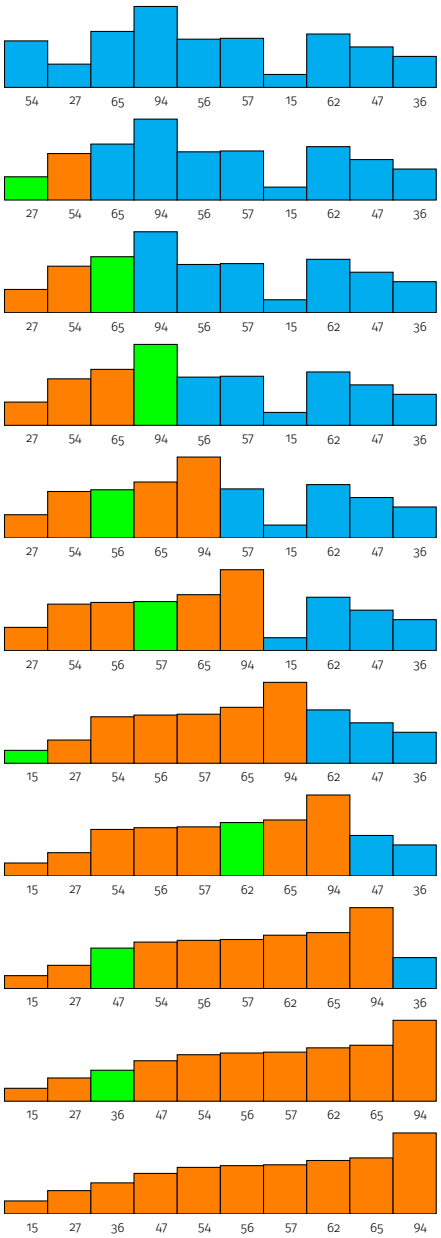
```
1  void insertionSort(int v[], int n) {
2   int i, key, j;
3
4   for (i = 1; i < n; i++) {
5     key = v[i];
6     j = i - 1;
7
8     while (j >= 0 && v[j] > key) {
9         v[j + 1] = v[j];
10        j = j - 1;
11    }
12    v[j + 1] = key;
13  }
14 }
```

Algoritmo 6 Ordenación por Inserción

```
1: procedure InsertionSort(V)                    ▷ V[0..n − 1]
2:     for i ← 1 to n − 1 do
3:         x ← V[i]
4:         j ← i − 1
5:         while j ≥ 1 ∧ x < V[j] do
6:             V[j + 1] ← V[j]
7:             j ← j − 1
8:         V[j + 1] ← x
```

$T(n) \in \mathcal{O}(n^2), \in \mathcal{O}(n)$

```
0: 54 27 65 94 56 57 15 62 47 36
1: 27 54 65 94 56 57 15 62 47 36
2: 27 54 65 94 56 57 15 62 47 36
3: 27 54 65 94 56 57 15 62 47 36
4: 27 54 56 65 94 57 15 62 47 36
5: 27 54 56 57 65 94 15 62 47 36
6: 15 27 54 56 57 65 94 62 47 36
7: 15 27 54 56 57 62 65 94 47 36
8: 15 27 47 54 56 57 62 65 94 36
9: 15 27 36 47 54 56 57 62 65 94
   15 27 36 47 54 56 57 62 65 94
```



Sección 6

Solución de ecuaciones de recurrencias

Subsección 6.1

Recurrencias homogéneas

Son de la forma:

$$a_0T(n) + a_1T(n-1) + a_2T(n-2) + \ldots + a_kT(n-k) = 0$$

que para resolver se deben buscar las soluciones que sean combinaciones de funciones exponenciales.
Para ello se realiza el cambio de variable $x^k = T(n)$ y se obtiene la *ecuación característica*:

$$a_0x^k + a_1x^{k-1} + a_2x^{k-2} + \ldots + a_k = 0$$

Se denota por $r_1, r_2, \ldots, r_k$ a las raíces(reales o complejas) de la ecuación, podemos tener dos casos:

Caso #1: Raíces distinas

Solución de la recurrencia:

$$T(n) = c_1r_1^n + c_2r_2^n + \ldots + c_kr_k^n = \sum_{i=1}^{k} c_ir_i^n$$

c_i se determinan a partir de las condiciones iniciales.

Ejemplo 1

Dada la siguiente recurrencia, determine la solución:

$$T(n) = T(n-1) + T(n-2), n \geq 2, T(0) = 0, T(1) = 1.$$

Haciendo $x^2 = T(n)$, se obtiene la ecuación característica $x^2 = x + 1$, cuyas raíces son:

$$r_1 = \frac{1+\sqrt{5}}{2}, r_2 = \frac{1-\sqrt{5}}{2}$$

Luego:

$$T(n) = c_1\left(\frac{1+\sqrt{5}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Constantes c_1 y c_2 usando las condiciones iniciales:

$$T(0) = c_1\left(\frac{1+\sqrt{5}}{2}\right)^0 + c_2\left(\frac{1-\sqrt{5}}{2}\right)^0 = c_1 + c_2 = 0$$

$$T(1) = c_1\left(\frac{1+\sqrt{5}}{2}\right)^1 + c_2\left(\frac{1-\sqrt{5}}{2}\right)^1 = 1$$

De aquí:

$$c_1 = -c_2 = \frac{1}{\sqrt{5}}$$

Sustituyendo:

$$T(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Sea

$$\phi = \left(\frac{1+\sqrt{5}}{2}\right)$$

Entonces:

$$T(n) = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n)$$

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

Caso #2: Raíces iguales

Supongamos que se repite r_1 (multiplicidad $m > 1$). La *ecuación característica* es de la forma:

$$(x - r_1)^m(x - r_2)\ldots(x - r_{k-m+1}) = 0$$

Solución de la recurrencia:

$$T(n) = \sum_{i=1}^{m} c_i n^{i-1} r_1^n + \sum_{i=m+1}^{k} c_i r_{i-m+1}^n$$

c_i se determinan a partir de las condiciones iniciales.

Ejemplo 2

Encontrar la solución de la siguiente ecuación de recurrencia:

$$T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3), n \geq 2, T(k) = k,\ para\ k = 0, 1, 2$$

Ecuación característica:

$$x^3 - 5x^2 + 8x - 4 = 0\ o\ (x-2)^2(x-1) = 0$$

Entonces:

$$T(n) = c_1 2^n + c_2 n 2^n + c_3 1.$$

$c_1 = 2, c_2 = -1/2\ y\ c_3 = -2$

Finalmente:

$$T(n) = 2^{n+1} - n2^{n-1} - 2.$$

$T(n) \in O(n2^n)$

Subsección 6.2

Ecuaciones no homogéneas

Considerar la ecuación de la forma:

$$a_0 T(n) + a_1 T(n-1) + \ldots + a_k T(n-k) = b^n p(n)$$

a_i y b son números reales $p(n)$ polinomio en n de grado d

Ecuación característica:

$$(a_0 x^k + a_1 x^{k-1} + \ldots + a_k)(x-b)^{d+1} = 0$$

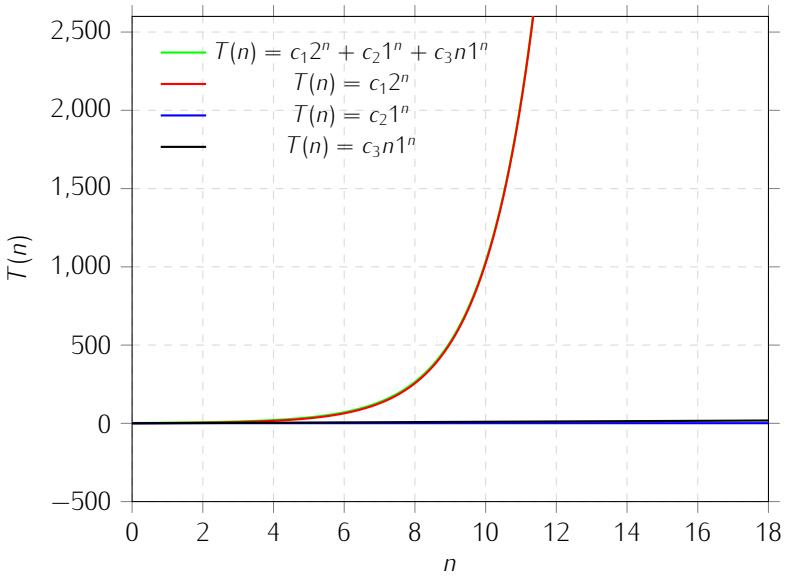
Ejemplo 3

$$T(n) = 2T(n-1) + n\ o\ T(n) - 2T(n-1) = n$$

De ahí se ve claramente que $b = 1$, $p(n) = n^1$ y $d = 1$
Ec. característica $(x-2)(x-1)^2 = 0$., por tanto

$$T(n) = c_1 2^n + c_2 1^n + c_3 n 1^n$$

$T(n) \in O(2^n)$



Subsección 6.3
Cambio de variables

Se aplica cuando n es potencia de un real a, osea $n = a^k$.

Ejemplo 4

$$T(n) = 4T\left(\frac{n}{2}\right) + n,\ n > 3$$
$$T(1) = 1\ y\ T(2) = 6$$

Si $n = 2^k$ podemos escribir:

$$T(2^k) = 4T(2^{k-1}) + 2^k$$

$\frac{2^k}{2} = 2^{-1}2^k = 2^{k-1}$

Cambio de variable nuevamente, $t_k = T(2^k)$:

$$t_k = 4t_{k-1} + 2^k$$

Esta es una ec. no homogénea (en k), cuya solución es:

$$t_k = c_1(2^k)^2 + c_2 2^k$$

Deshaciendo los cambios:

$$T(n) = c_1 n^2 + c_2 n$$

Calculando c_1 y c_2 de las condiciones iniciales, $c_1 = 2$ y $c_2 = -1$.

$$T(n) = 2n^2 - n$$

$T(n) \in O(n^2)$

Subsección 6.4
Iteraciones

También se dice desenrrollando.

Ejemplo 5 Resuelva la siguiente ecuación de recurrencia. Suponga que $p > q^r$.

$$T(n) = pT\left(\frac{n}{q}\right) + kn^r, T(1) = 1$$

Solución: Desenrollando la ecuación $j-1$ veces:

$$T(n) = kn^r\left(1 + \frac{p}{q^r} + \left(\frac{p}{q^r}\right)^2 + \cdots + \left(\frac{p}{q^r}\right)^{j-1}\right) + p^j T\left(\frac{n}{q^j}\right)$$

que es lo mismo que:

$$T(n) = kn^r\left(\left(\frac{p}{q^r}\right)^0 + \left(\frac{p}{q^r}\right)^1 + \left(\frac{p}{q^r}\right)^2 + \cdots + \left(\frac{p}{q^r}\right)^{j-1}\right) + p^j T\left(\frac{n}{q^j}\right)$$

que es:

$$T(n) = kn^r \sum_{i=0}^{j-1}\left(\frac{p}{q^r}\right)^i + p^j T\left(\frac{n}{q^j}\right)$$

Usando:

$$\sum_{j=1}^{n} c^j = \frac{c^{n+1}-1}{c-1}, c \neq 1$$

Como $p > q^r$ (en este caso):

$$T(n) = kn^r \frac{\left(\frac{p}{q^r}\right)^j - 1}{\frac{p}{q^r} - 1} + p^j T\left(\frac{n}{q^j}\right)$$

Suponiendo que $n = q^j \Rightarrow \log_q n = j$:

$$T(n) = kn^r \frac{\left(\frac{p}{q^r}\right)^{\log_q n} - 1}{\frac{p}{q^r} - 1} + p^{\log_q n} T(1)$$

Dado que:

$$\left(\frac{p}{q^r}\right)^{\log_q n} = \frac{(q^{\log_q p})^{\log_q n}}{q^{r\log_q n}} = \frac{(q^{\log_q n})^{\log_q p}}{q^{\log_q n^r}} = \frac{n^{\log_q p}}{n^r}$$

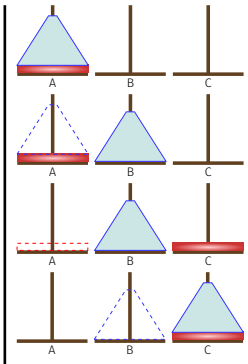
y que $T(1) = 1$ y considerando $K' = \frac{k}{\frac{p}{q^r} - 1}$:

$$T(n) = K'n^r\left(\frac{n^{log_q p}}{n^r} - 1\right) + n^{\log_q p} = (K'+1)n^{\log_q p} - K'n^r$$

Luego: $T(n) \in O(n^{\log_q p})$

Pendiente: 1. $p = q^r$, 2. $p < q^r$

Ejemplo 6 Torres de Hanoi



Algoritmo 7 Torres de Hanoi

```
1: procedure HANOI(disc, source, destination, auxiliary)
2:     if disc = 1 then
3:         mover disc from source to destination
4:     else
5:         HANOI(disc − 1, source, auxiliary, destination)
6:         mover disc from source to destination
7:         HANOI(disc − 1, auxiliary, destination, source)
```

$$T(n) = T(n-1) + 1 + T(n-1)$$

$$T(n) = 2T(n-1) + 1$$

Usando solución para recurrencias no homogéneas:

$$T(n) - 2T(n-1) = 1$$

Claramente se aprecia que $b = 1$, $p(n) = 1$, luego $d = 0$.
Ecuación característica:

$$(x-2)(x-1) = 0$$

Cuya solución es:

$$T(n) = c_1 2^n + c_2 1^n$$

Como para $n = 1$ disco se realiza un movimiento, $T(1) = 1$, para $n = 2$:

$$T(2) = 2T(1) + 1 = 3$$

Buscando las constantes, tenemos:

$$T(1) = c_1 2^1 + c_2 = 1$$

$$T(2) = c_1 2^2 + c_2 = 3$$

Cuya solución es, $c_1 = 1$ y $c_2 = -1$.
Finalmente:

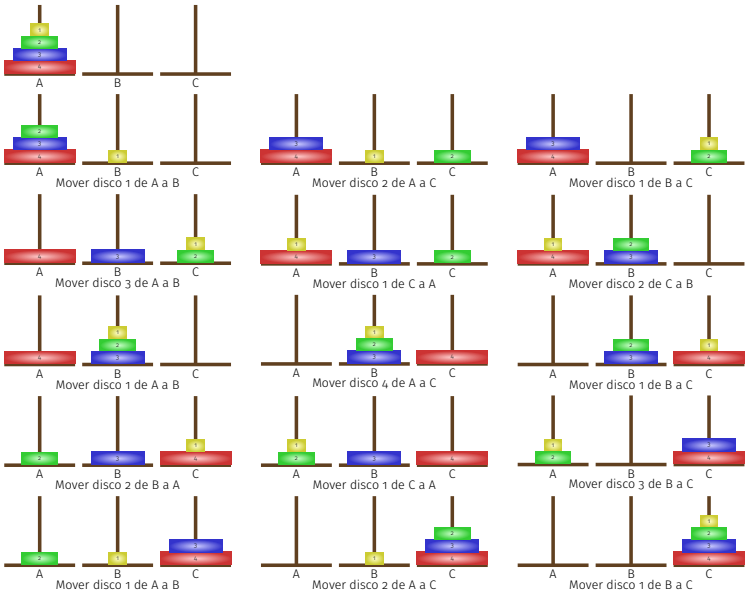
$$T(n) = 2^n - 1$$

$T(n)$ expresa la cantidad de movimientos dependiendo de la cantidad de discos.
Ej: $T(4) = 2^4 - 1 = 15$, 15 movimientos para 4 discos.

Secuencia Hanoi:

n	1	2	3	4	5
$T(n)$	1	3	7	15	31

```
void hanoi(int a,char from,char to,char aux){
  if(a==1)
    printf("\t\tMover disco 1 de %c a %c\n",from,to);
  else{
    hanoi(a-1,from,aux,to);
    printf("\t\tMover disco %d de %c a %c\n",a,from,to);
    hanoi(a-1,aux,to,from);
  }
}
Se llama:
hanoi(4,'A','C','B');
```



$$T(1) = 1$$
$$T(n) = 8(2 \cdot T(n/2) + 1),$$

```
int Test (int s, int t, int n) {
  int i, ;
  if (n==1) {
    if(found_between(s,t))
      return 1;
    else
      return 0;
  }
  for (i = 0; i <= 7; i++) {
    if(Test(s,t-1,n%2) && Test(s-1,t,n%2))
      return 1;
    else
      return 0;
  }
}
```

Análisis:

Ejemplo 7

$$\begin{aligned}
T(n) &= 16 \cdot T(n/2) + 8 \\
&= 16(16T(n/n^2) + 8) + 8 \\
&= 16^2 T(n/2^2) + 16 \cdot 8 + 8 \\
&= 16^3 T(n/2^3) + 16^2 \cdot 8 + 16 \cdot 8 + 8 \\
&= 16^3 T(n/2^3) + 16^2 \cdot 8 + 16^1 \cdot 8 + 16^0 \cdot 8 \\
&= \cdots \\
&= 16^k T(n/2^k) + \cdots + 16^2 \cdot 8 + 16 \cdot 8 + 16^0 \cdot 8 \\
&= 16^k T(n/2^k) + 8 \sum_{i=0}^{k-1} 16^i \\
&= 16^k T(n/2^k) + 8\left(\frac{16^k}{15} - \frac{1}{15}\right)
\end{aligned}$$

Como $2^k = n, 16^k = 2^{4k} = (2^k)^4$

$$\begin{aligned}
&= (2^k)^4 T(n/2^k) + 8\left(\frac{(2^k)^4}{15} - \frac{1}{15}\right) \\
&= n^4 + \frac{8n^4}{15} + \frac{8}{15} \\
&= \frac{23}{15}n^4 + \frac{8}{15}.
\end{aligned}$$

Ejemplo 8

Resolver, donde n es potencia de 2.

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 3T\left(\frac{n}{2}\right) + n & \text{si } n > 1 \end{cases}$$

Calculamos algunos valores de la secuencia:

n	1	2	4	8	16	32
$T(n)$	1	5	19	65	211	665

Ejemplo:

$$\begin{aligned}
T(16) &= 3T(8) + 16 \\
&= 3(65) + 16 \\
&= 211
\end{aligned}$$

De esta manera podemos ver que:

n	T(n)
2^0	1
2^1	$3^1 \cdot +2^1$
2^2	$3^2 \cdot 1 + 3 \cdot 2 + 2^2$
2^3	$3^3 \cdot 1 + 3^2 \cdot 2 + 3 \cdot 2^2 + 2^3$
2^4	$3^4 \cdot 1 + 3^3 \cdot 2 + 3^2 \cdot 2^2 + 3 \cdot 2^3 + 2^4$
2^5	$3^5 \cdot 1 + 3^4 \cdot 2 + 3^4 \cdot 2^2 + 3^2 \cdot 2^3 + 3 \cdot 2^4 + 2^5$
$\dots$	$\dots$

Para k-veces:

$$
\begin{aligned}
T(2^k) &= 3^k \cdot 2^0 + 3^{k-1} \cdot 2^1 + 3^{k-2} \cdot 2^2 + \cdots + 3^1 \cdot 2^{k-1} + 3^0 \cdot 2^k \\
&= \sum_{i=0}^{k} 3^{k-i} \cdot 2^i \\
&= 3^k \sum_{i=0}^{k} \left(\frac{2}{3}\right)^i \\
&= 3^k \left[\frac{1 - \left(\frac{2}{3}\right)^{k+1}}{1 - \frac{2}{3}}\right] \\
&= 3^k \left[\frac{1 - \left(\frac{2}{3}\right)^{k+1}}{\frac{1}{3}}\right] \\
&= 3^{k+1}\left[1 - \left(\frac{2}{3}\right)^{k+1}\right] \\
&= 3^{k+1} - \frac{3^{k+1}2^{k+1}}{3^{k+1}} \\
&= 3^{k+1} - 2^{k+1}
\end{aligned}
$$

Como $n = 2^k$, entonces $k = \lg n$. Luego:

$$
\begin{aligned}
T(n) &= T(2^{\lg n}) \\
&= 3^{1+\lg n} - 2^{1+\lg n}
\end{aligned}
$$

Y $3^{\lg n} = n^{\lg 3}$. Entonces:

$$
\begin{aligned}
T(n) &= 3^1 3^{\lg n} - 2^1 2^{\lg n} \\
&= 3n^{\lg 3} - 2n
\end{aligned}
$$

Por lo tanto, $T(n) \in \mathcal{O}(n^{\lg 3})$
Nota: $\lg 3 = 1.585$

SECCIÓN 7

Cuestiones y problemas

7.1 Encuentre las soluciones asintóticas de las siguientes recurrencias:

a) $T(n) = T(n-1) + 5n^2 - 3n$
b) $a_n = 5a_{n-1} + 6a_{n-2} = 0, n \geq 2, a_0 = 1, a_1 = 3$
c) $2a_{n+2} - 11a_{n+1} + 5a_n = 0, n \geq 0, a_0 = 2, a_1 = -8$
d) $3a_{n+1} = 2a_n + a_{n-1} = 0, n \geq 1, a_0 = 7, a_1 = 3$
e) $a_{n+2} + a_n = 0, n \geq 0, a_0 = 0, a_1 = 3$
f) $a_{n+2} + 4a_n = 0, n \geq 0, a_0 = a_1 = 1$
g) $a_n - 6a_{n-1} + 9a_{n-2} = 0, n \geq 2, a_0 = 5, a_1 = 12$
h) $a_n + 2a_{n-1} + 2a_{n-2} = 0, n \geq 2, a_0 = 1, a_1 = 3$
i) $a_{n+1} - a_n = 2n + 3, n \geq 0, a_0 = 1$
j) $a_{n+1} - a_n = 3n^2 - n, n \geq 0, a_0 = 3$
k) $a_{n+1} - a_n = 5, n \geq 0, a_0 = 1$
l) $a_n + na_{n-1} = n! n \geq 1, a_0 = 1$
m) $a_{n+1} - 2a_n = 2^n, n \geq 0, a_0 = 1$

7.2 Si a_n, $n \geq 0$, es una solución de la relación de recurrencia $a_{n+1} - da_n = 0$ y $a_3 = 153/49$, $a_5 = 1377/2401$, ¿cuánto vale d?

7.3 Si $a_0 = 0$, $a_1 = 1$, $a_2 = 4$ y $a_3 = 37$ satisfacen la relación de recurrencia $a_{n+2} + ba_{n+1} + ca_n = 0$, donde $n \geq 0$ y b, c son constantes, encuentre a_n.

7.4 Resolver $a_{n+2} - 6a_{n+1} + 9a_n = 3(2^n) + 7(3^n), n \geq 0, a_0 = 1, a_1 = 4$

7.5 Resuelva la recurrencia, determine el orden y compruebe por Teorema Maestro:

$$T(n) = \begin{cases} 1 & \text{si } n = 0 \\ 3T(\frac{n}{4}) + \Theta(\sqrt{n}) & \text{si } n > 0 \end{cases}$$

7.6 Encuentre la complejidad de la siguiente función:

```
int func1 (int n){
  int i=1;
  while(i<n){
    int j=n;
    while(j>0)
      j = j/2;
    i=2*i;
  }
}
```

7.7 Considere el siguiente algoritmo para ordenar un arreglo de enteros de tamaño n:

- Dividir el arreglo en $\frac{n}{k}$ pedazos de tamaño k
- Ordenar cada pedazo usando ordenamiento por inserción
- Mezclar los pedazos

Analice este algoritmo obteniendo el orden temporal del mismo. En base a su resultado ¿Qué valor de k usaría Ud.? Justifique.

7.8 Indicar cual es el objetivo del siguiente código. Además encuentre del orden temporal.

```
int prog1 ( int a[], int x, int i, int j )
{
  if ( i > j )
     return -1;
  if ( i == j ){
     if ( a[i] == x )
        return i;
     else
        return -1;
  }
  else{
      int n1 = ( 2 * i + j ) / 3;
      int n2 = ( i + 2 * j ) / 3;

      if ( x <= a[n1] )
         j = n1;
      else if ( x >= a[n2] )
         i = n2;
      else{
         i = n1 + 1;
         j = n2 - 1;
      }
      return prog1 ( a, x, i, j );
  }
}
```

7.9 Hallar una ecuación de recurrencia lineal homogénea y sus condiciones iniciales cuyo término general sea: $a_n = 3^{n+2} + n3^{n-2}$.

7.10 Determine $sum[1 \ldots k]$

Contar

```
1: function COUNT(n)                          ▷ n = k^2 para algún k entero
2:     k ← √n
3:     for j ← 1 to k do
4:         sum[j] ← 0
5:         for i ← 1 to j^2 do
6:             sum[j] ← sum[j] + i
       return sum[1 ... k]
```

7.11 ¿Cuál es el valor retornado por la siguiente función? Exprese su respuesta en función de n. Usando notación O, indique el peor caso en la ejecución.

```
int pesky(int n){
    int i,j,k;
    int r = 0;

    for(i=1; i <= n-1; i++)
       for(j=1; j <= i; j++)
          for(k=j; k <= i+j; k++)
             r = r+1;
    return(r);
}
```

7.12 Escribir el código en C para sumar dos matrices de nxm y dejar el resultado en una tercera matríz. Indique el orden de su código. Repita el problema para multiplicación de matrices.

7.13 Diseñe un algoritmo para calcular la moda de un conjunto de valores, de tamaño n. Su algoritmo debe ser $O(n)$.

7.14 Indique la relación de inclusión entre los diferentes órdenes:

$O(n)$, $O(\log n)$, $O(\sqrt{n})$, $O(n^3)$, $O(n^n)$, $O(1)$, $O(n \log n)$, $O(2^n)$, $O(n^2)$.

Ejemplo $O(n) \subset O(n \log n)$

7.15 Dada la ecuación de recurrencia,

$$f(n) = \begin{cases} d & \text{si } n = 1 \\ af(n/c) + bn^x & \text{si } n \geq 2 \end{cases}$$

determinar la solución, considerando el caso en que $a = c^x$ y el caso $a \neq c^x$

7.16 Considere el siguiente algoritmo: suponga que la operación crucial es examinar un elemento. El algoritmo examina los n elementos de un conjunto y, de alguna manera, eso le permite descartar $2/5$ de los elementos para entonces realizar una llamada recursiva sobre los restantes $3/5$ elementos. Escriba una ecuación de recurrencia que describa este comportamiento.

7.17 Escriba la ecuación de recurrencia.

```
1  int sort(int A[],int n, int i, intj){ // n potencia de 3
2     int k;
3
4     if(i<j){
5        k = ((j-i)+1)/3;
6        sort(A,n,i,i+k-1);
7        sort(A,n,i+k,i+2k-1);
8        sort(A,n,i+2k,j);
9        Merge(A,i,i+k,i+2k,j);
10       // Merge intercala
11       // A[i..(i+k-1)],A[(i+k)..(i+2k-1)] y
12       // A[i+2k .. j] en A[i..j] con un costo de 5n/3-2
13    }
14 }
```

PARTE

IV

Repaso Matemáticas para Ciencias de la Computación

Subsección 7.1

Potencias

$$a^0 = 1, a \neq 0 \qquad a^p + a^p = 2a^p \neq a^{2p} \qquad \sqrt[n]{a} = a^{1/n}$$
$$a^p a^q = a^{p+q} \qquad a^{-p} = \frac{1}{a^p} \qquad \sqrt[n]{a^m} = a^{m/n}$$
$$\frac{a^p}{a^q} = a^{p-q} \qquad (ab)^p = a^p b^p \qquad \sqrt[n]{\frac{a}{b}} = \frac{\sqrt[n]{a}}{\sqrt[n]{b}}$$
$$(a^p)^q = a^{pq} \qquad 2^n + 2^n = 2^{n+1}$$

Subsección 7.2

Logaritmos

Sea b un número real positivo mayor a 1, x un número real y suponga que para algún número real positivo y tenemos $y = b^x$. Entonces, x se llama el *logaritmo de y en base b*, y se escribe:

$$x = \log_b y$$

Aquí b es la base del logaritmo.

7.2.1. Identidades y propiedades

$$\log_a 1 = 0, \log_a a = 1 \qquad e = \lim(1 + \frac{1}{n})^n = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots = 2.7182818\ldots$$
$$\log_b xy = \log_b x + \log_b y \qquad \log_a x = \log_b x \log_a b \ o \ \log_b x = \frac{\log_a x}{\log_a b}$$
$$\log_b \frac{x}{y} = \log_b x - \log_b y \qquad x^{\log_b y} = y^{\log_b x}, x, y > 0$$
$$\log_b c^y = y \log_b c, \text{si } c > 0$$

Subsección 7.3

Piso($floor$) y techo($ceiling$)

$\lfloor x \rfloor$ denota el piso de x, y se define como el entero más grande cercano o igual a x. $\lceil x \rceil$ denota el techo de x, y se define como el entero más cercano o igual a x. Ejemplo:

$$\lfloor \sqrt{2} \rfloor = 1, \lceil \sqrt{2} \rceil = 2, \lfloor \sqrt{-2.5} \rfloor = -3, \lceil \sqrt{-2.5} \rceil = -2$$

$$\lfloor \pi \rfloor = 3, \lceil \pi \rceil = 4, \lfloor e \rfloor = 2, \lceil e \rceil = 3$$

7.3.1. Identidades y propiedades

$$a-1 < \lfloor a \rfloor \leq a \leq \lceil a \rceil < a+1$$
$$\lceil x/2 \rceil + \lfloor x/2 \rfloor = x$$
$$\lfloor -x \rfloor = -\lceil x \rceil$$
$$\lceil -x \rceil = \lfloor -x \rfloor$$

$$\lceil \sqrt{\lceil x \rceil} \rceil = \lceil \sqrt{x} \rceil$$
$$\lfloor n/2 \rfloor = \begin{cases} n/2 & \text{si } n \text{ es par} \\ (n-1)/2 & \text{si } n \text{ es impar} \end{cases}$$

Subsección 7.4

Factorial y coeficiente Binomial

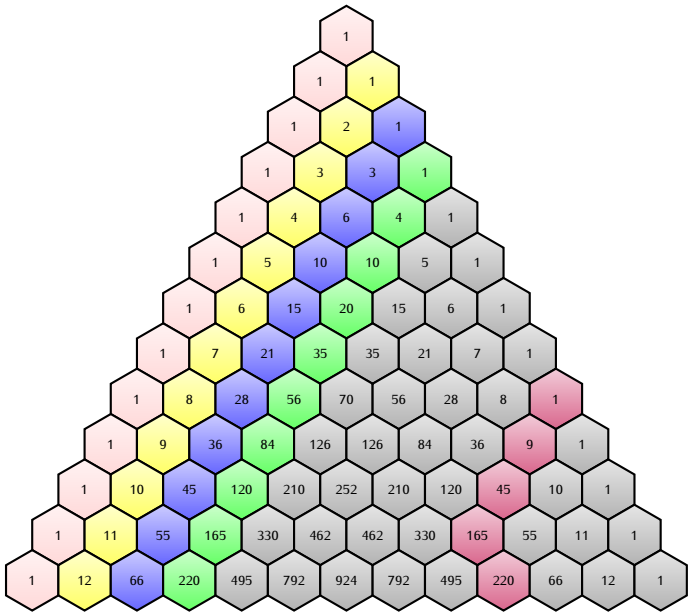
$0! = 1, n! = n(n-1)!, \text{si} n \geq 1$

$30! = 265252859812191058636308480000000.$

$n! \approx \sqrt{2\pi n}(\frac{n}{e})^n, e = 2.7182818\ldots$

$30! \approx 264517095922964306151924784891709$

$$n! = \sqrt{2\pi n}(\frac{n}{e})^n(1 + \frac{1}{12n} + \frac{1}{288n^2} - \frac{139}{5140n^3} + O(\frac{1}{n^4}))$$
$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$



7.4.1. Identidades y propiedades

$$\binom{n}{k} = \binom{n}{n-k}$$
$$\binom{n}{n} = \binom{n}{0} = 1$$
$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$
$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{n-k}$$
$$(1+x)^n = \sum_{j=0}^{n} \binom{n}{j} x^j$$
$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$$
$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n} = 2^n$$
$$1 + 2 + \ldots + n = \binom{1}{1} + \binom{2}{1} + \cdots + \binom{n}{1} = \binom{n+1}{2}$$
$$\sum_{jpar} \binom{n}{j} = \sum_{jimpar} \binom{n}{j}$$
$$\sum_{k \le n} \binom{r+k}{k} = \binom{r+n+1}{n}$$
$$\sum_{m \le k \le n} \binom{k}{m} = \binom{n+1}{m+1}$$

Subsección 7.5

Sumatorias

$$\sum_{j=1}^{n} a_j = \sum_{1 \le j \le n} a_j = a_1 + a_2 + a_3 + \cdots + a_n$$

7.5.1. Identidades y propiedades

$$\sum_{j=1}^{n} a_{n-j+1} = a_{n-1+1} + a_{n-2+1} + \cdots + a_{n-n+1} = \sum_{j=1}^{n} a_j$$

$$\begin{aligned}
\sum_{j=1}^{n} a_{n-j} &= \sum_{1 \le j \le n} a_{n-j} \\
&= \sum_{1 \le n-j \le n} a_{n-(n-j)} \\
&= \sum_{1-n \le n-j-n \le n-n} a_{n-(n-j)} \\
&= \sum_{1-n \le -j \le 0} a_j \\
&= \sum_{0 \le j \le n-1} a_j \\
&= \sum_{j=0}^{n-1} a_j
\end{aligned}$$

$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

$$\sum_{j=1}^{n} j^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{j=1}^{n} j^3 = \frac{n^2(n+1)^2}{4}$$

$$\sum_{k=1}^{n} k(k+1) = \frac{n(n+1)(n+2)}{3}$$

$$\sum_{k=1}^{n} \frac{1}{k(k+1)} = \frac{n}{n+1}$$

$$\sum_{j=1}^{n} c^j = \frac{c^{n+1}-1}{c-1}, c \neq 1$$

$$\sum_{k=1}^{n} k(k+1)(k+2) = \frac{n(n+1)(n+2)(n+3)}{4}$$

$$\sum_{k=1}^{n} \frac{1}{k(k+1)(k+2)} = \frac{n(n+3)}{4(n+1)(n+2)}$$

$$\sum_{k=1}^{n} (2k-1) = n^2$$

$$\sum_{j=0}^{\infty} c^j = \frac{1}{1-c}, |c| < 1$$

$$\sum_{j=0}^{n} jc^j = \sum_{j=1}^{n} jc^j = \frac{nc^{n+2} - nc^{n+1} - c^{n+1} + c}{(c-1)^2}, c \neq 1$$

$$\sum_{j=0}^{\infty} jc^j = \frac{c}{(1-c)^2}, |c| < 1$$

$$\frac{\log(n+1)}{\log e} \leq \sum_{j=1}^{n} \frac{1}{j} \leq \frac{\log n}{\log e} + 1$$

$$H_n = \sum_{j=1}^{n} \frac{1}{j}$$

Subsección 7.6

Series

Sea $a_1, a_2, \ldots, a_n$, donde $a_i = a_1 + (i-1)k$:

$$a_1 + a_2 + \ldots + a_n = \frac{n(a_1 - a_n)}{2}$$

Sea $a_1, a_2, \ldots, a_n$, donde $a_i = ar^{i-1}$:

$$a + ar + ar^2 + \ldots + ar^{n-1} = \frac{a(1-r^n)}{1-r} = \frac{a_1 - ra_n}{1-r}$$

$$a + (a+d)r + (a+2d)r^2 + \ldots + (a+(n-1)d)r^{n-1} = \frac{a(1-r^n)}{1-r} + \frac{rd(1 - nr^{n-1} + (n-1)r^n)}{(1-r)^2}$$

$$1+2+3+\ldots+n=\frac{n(n+1)}{2}$$
$$1^2+2^2+3^2+\ldots+n^2=\frac{n(n+1)(2n+1)}{6}$$
$$1^3+2^3+3^3+\ldots+n^3=\frac{n^2(n+1)^2}{4}=(1+2+3+\ldots+n)^2$$
$$2^0+2^1+2^2+\ldots+2^n=2^{n+1}-1$$
$$a^0+a^1+a^2+\ldots+a^n=\frac{a^{n+1}-1}{a-1}\text{, y si } 0<a<1 \text{ entonces} \sum_{i=0}^{n} a^i \leq \frac{1}{1-a}$$
$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\frac{1}{5}-\ldots=\ln 2$$
$$1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\ldots=\frac{\pi}{4}$$
$$\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\ldots=\frac{\pi}{6}$$
$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\ldots+\frac{1}{N}\approx lnN\text{, el error tiende a } \gamma$$
$$H_n=\ln n+\gamma+\frac{1}{2n}-\frac{1}{12n^2}+\frac{1}{120n^4}+\ldots=O(\lg(n)),\ \gamma\approx 0.577215664901$$

7.6.1. Matrices

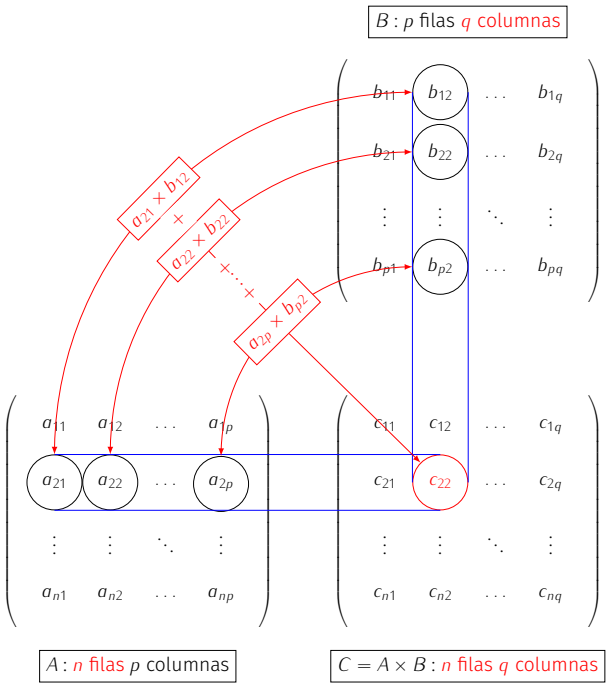
Determinante:Regla de Sarrus,

$$D=\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \begin{matrix} a_{11} & a_{12} \\ a_{21} & b_{22} \\ a_{31} & b_{32} \end{matrix}$$

$$=a_{11}a_{22}a_{33}+a_{21}a_{32}a_{13}+a_{31}a_{12}a_{23}-a_{13}a_{22}a_{31}-a_{23}a_{32}a_{11}-a_{33}a_{12}a_{21}$$

Dadas las matrices de la misma dimensión $A=(a_{ij})$ y $B=(b_{ij})$, entonces:

$$A+B=(a_{ij}+b_{ij})$$
$$A-B=(a_{ij}-b_{ij})$$
$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{bmatrix}$$
$$c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{in}+b_{nj}=\sum_{k=1}^{n} a_{ik}b_{kj}$$



Transpuesta:

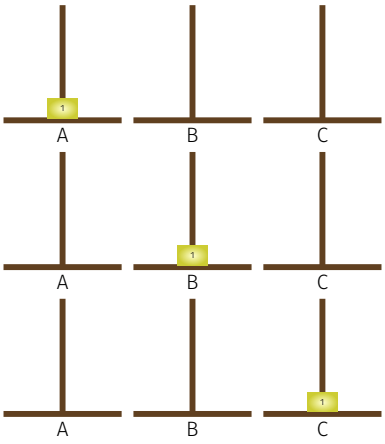
$$(A^T)_{ij} = A_{ji}, 1 \leq i \leq n, 1 \leq j \leq m$$

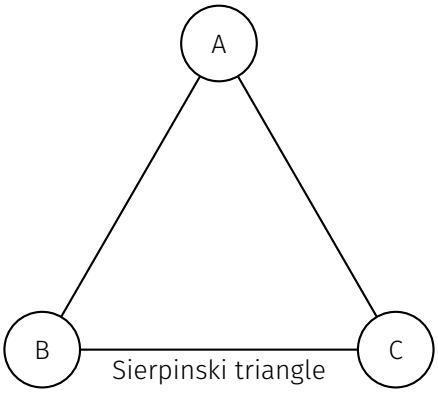
$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}^T = \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix}$$

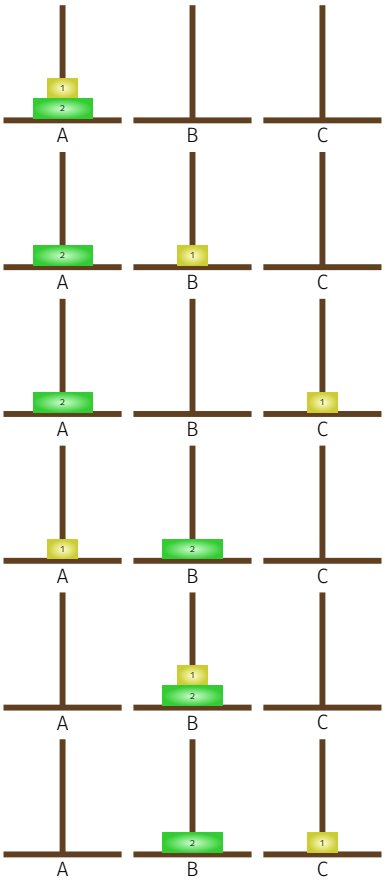
αA	νN
βB	$\xi \Xi$
$\gamma \Gamma$	oO
$\delta \Delta$	$\pi \Pi$
$\epsilon \varepsilon E$	$\rho \varrho P$
ζZ	$\sigma \Sigma$
ηH	τT
$\theta \vartheta \Theta$	$\upsilon \Upsilon$
ιI	$\phi \varphi \Phi$
κK	χX
$\lambda \Lambda$	$\psi \Psi$
μM	$\omega \Omega$

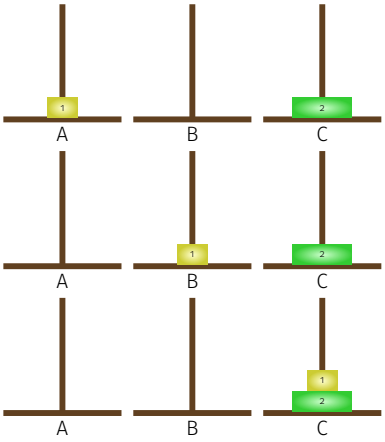
Subsección 7.7

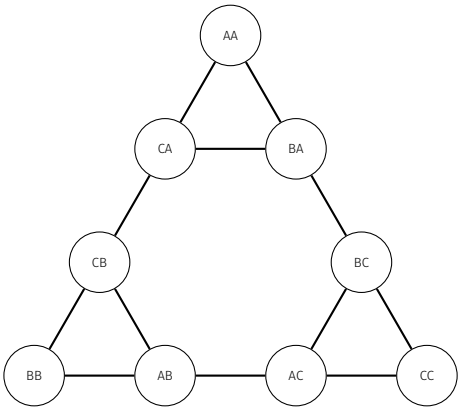
Hanoi

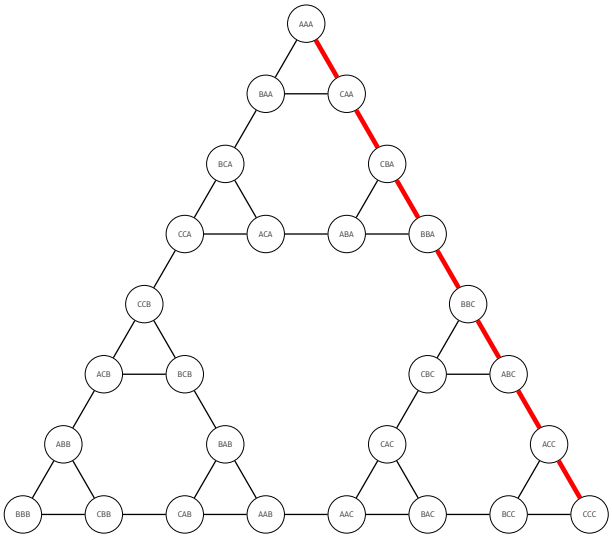


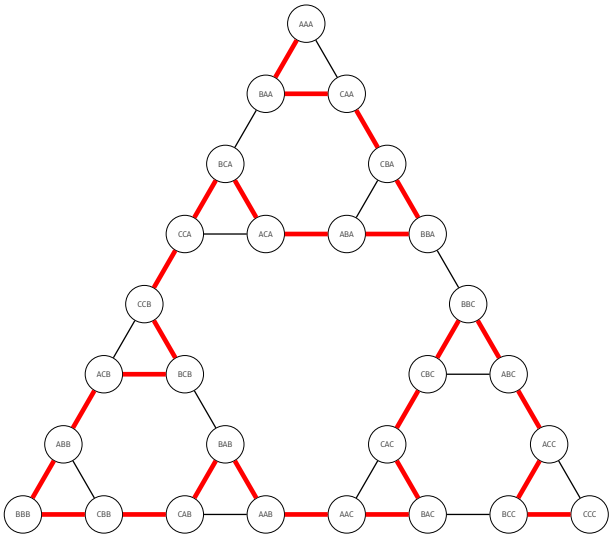



1
2
A
B
C


1
2
A
B
C


AA
CA
BA
CB
BC
BB
AB
AC
CC


AAA
BAA
CAA
BCA
CBA
CCA
ACA
ABA
BBA
CCB
BBC
ACB
BCB
CBC
ABC
ABB
BAB
CAC
ACC
BBB
CBB
CAB
AAB
AAC
BAC
BCC
CCC


AAA
BAA
CAA
BCA
CBA
CCA
ACA
ABA
BBA
CCB
BBC
ACB
BCB
CBC
ABC
ABB
BAB
CAC
ACC
BBB
CBB
CAB
AAB
AAC
BAC
BCC
CCC

Referencias

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 2009.

[2] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8):707–710, 1966.