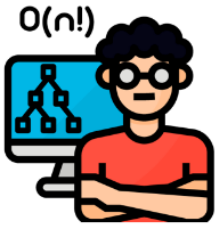
Instituto Politécnico Nacional

Escuela Superior de Cómputo

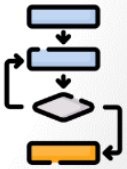


Análisis de algoritmos

Tema 03: Análisis temporal

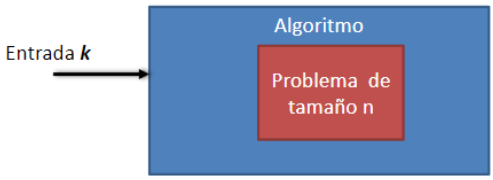
Prof. Edgardo Adrián Franco Martínez
http://eafranco.com
edfrancom@ipn.mx
@edfrancom edfrancom

https://creativecommons.org/licenses/by-nc-sa/4.0/deed.es



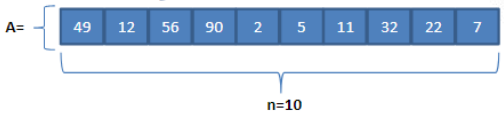
Caso de entrada

- Un **caso de entrada** para un algoritmo **es una instancia** del problema inicial.
- Un algoritmo para resolver un problema de tamaño n, puede recibir una o más entradas, las cuales pueden definir el número de operaciones que hará.
  - Se puede determinar que tipos de operaciones utiliza y cuantas veces las ejecuta para una entrada especifica ***k***.
    - Por ejemplo realizar una búsqueda lineal del 40 en un arreglo de 100,000 elementos (n=100,000).



Ejemplo 1 (Búsqueda lineal)

```
func BusquedaLineal(Valor,A,n)
{
        i=1;
        while(i<=n&&A[i]!=Valor)
        {
                i=i+1;
        }
        return i;
}
```



Tamaño de problema n=10

***Caso 1:** Valor=11, A={49,12,56,90,2,5,11,32,22,7}*

Si el Valor=11 ⟹ se entra al ciclo 6 veces ⟹ k=6

Si el ciclo se ejecutará k veces

→k sumas (una por cada iteración).

→k + 2 asignaciones (las del ciclo y las realizadas fuera del ciclo).

→k + 1 operaciones lógicas (la condición se debe probar k + 1 veces, la última es para saber que el ciclo no debe volver a ejecutarse).

→k + 1 comparaciones con el índice.

→k + 1 comparaciones con elementos de A:

→k + 1 accesos a elementos de A:

→6k + 6 operaciones en total.

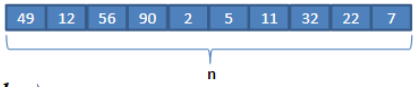
Da clic para ver paso a paso

Operación básica

- Del ejemplo notamos que el número de veces que se ejecutan algunas operaciones, para valores sucesivamente mayores que ***k*** y/o ***n;*** se presenta un modelo de crecimiento similar al que tiene el número total de operaciones que ejecuta.
- Para hacer una estimación de la cantidad de tiempo que tarda un algoritmo en ejecutarse, **no es necesario contar el número total de operaciones que realiza**.
- **Se puede elegir** alguna, a la que se identificará como **operación básica** **que observe un comportamiento parecido al del número total de operaciones realizadas** y que, por lo tanto, será proporcional al tiempo total de ejecución.

- En general, debe procurarse que la **operación básica**, en la cual se basa el análisis, de alguna forma **esté relacionada con el tipo de problema que se intenta resolver**, ignorando las asignaciones de valores iniciales y las operaciones sobre variables para control de ciclos (índices).
  - Pg. la operación básica en el Algoritmo (búsqueda lineal) es la **comparación entre los elementos del arreglo y el valor buscado**.

```
func BusquedaLineal(Valor,A,n)
{
        i=1;
        while(i<=n&&A[i]!=Valor)
        {
                i=i+1;
        }
        return i;
}
```



→Operación básica "Compración A[i]!=Valor"

Ejemplo 2 (Producto de 2 mayores)

- El algoritmo siguiente obtiene el producto de los dos valores más grandes contenidos en un arreglo *A* de *n* enteros

```
func Producto2Mayores(A,n)
        if(A[1] > A[2])
                mayor1 = A[1];
                mayor2 = A[2];
        else
                mayor1 = A[2];
                mayor2 = A[1];
        i = 3;

        while(i<=n)
                if(A[i] > mayor1)
                        mayor2 = mayor1;
                        mayor1 = A[i];
                else if (A[i] > mayor2)
                        mayor2 = A[i];
                i = i + 1;

        return = mayor1 * mayor2;
```

- En este algoritmo se realizan las siguientes operaciones:
  a) Comparación con los elementos del arreglo
  b) Asignaciones a mayor1 y mayor2
  c) Asignación al índice *i*
  d) Asignación a la función
  e) Producto de los mayores
  f) Incremento al índice *i*
  g) Comparación entre el índice *i* y la longitud del arreglo *n*

- Las operaciones (c), (f) y (g) no se consideran por realizarse entre índices, las operaciones (d) y (e) se ejecutan una sola vez y no son proporcionales al número total de operaciones.

- Entonces, se tiene que las **operaciones que se pueden considerar para hacer el análisis** son: (a) Comparación con los elementos del arreglo y (b) las asignaciones a los elementos mayores.

- (c), (f), (g), (d) y (e) pueden omitirse para el análisis
- (a) y (b) operaciones básicas del algoritmo

```
func Producto2Mayores(A,n)                    →(d)
        if(A[1] > A[2])                       →(a)
                mayor1 = A[1];                →(b)
                mayor2 = A[2];                →(b)
        else
                mayor1 = A[2];                →(b)
                mayor2 = A[1];                →(b)
        i = 3;                                →(c)

        while(i<=n)                           →(g)
                if(A[i] > mayor1)             →(a)
                        mayor2 = mayor1;      →(b)
                        mayor1 = A[i];        →(b)
                else if (A[i] > mayor2)       →(a)
                        mayor2 = A[i];        →(b)
                i = i + 1;                    →(f)

        return = mayor1 * mayor2;             →(e)
```

Elección de la operación básica

- El análisis de un algoritmo se puede hacer considerando sólo aquella operación que cumpla los siguientes **criterios**:

  a) Debe estar **relacionada** con el tipo de **problema** que se resuelve.
  b) Debe **ejecutarse** un número de veces cuyo **modelo de crecimiento sea similar al del número total de operaciones** que efectúa el algoritmo.

- Si ninguna de las operaciones encontradas cumple con ambos criterios, es posible declinar por el primero. Si aun así no es posible encontrar una operación representativa, se debe hacer un análisis global, contando todas las operaciones.

- Elección de la operación básica para algunos problemas:
  - Búsqueda de un elemento en un conjunto
    - Comparación entre el valor y los elementos del conjunto
  - Multiplicar dos matrices
    - Producto de los elementos de las matrices
  - Recorrer un árbol
    - Visitar un nodo
  - Resolver un sistema de ecuaciones lineales
    - Suma y resta de las ecuaciones
  - Ordenar un conjunto de valores
    - Comparación entre valores

Análisis de algoritmos
03 Análisis temporal
Prof. Edgardo Adrián Franco Martínez

Da clic para ver paso a paso

Concepto de Instancia

- Un **problema computacional** consiste en una caracterización de un conjunto de datos de entrada, junto con la especificación de la salida deseada con base a cada entrada.

- Un problema computacional tiene una o más **instancias**, que son **valores particulares para los datos de entrada**, sobre los cuales se puede ejecutar el algoritmo para resolver el problema; i.e. un caso específico de un problema.

- **Ejemplo**: el problema computacional de *multiplicar dos números enteros* tiene, las siguientes instancias: multiplicar 345 por 4653, multiplicar 2637 por 10000, multiplicar -32341 por 12, etc.

Análisis Peor Caso, Mejor Caso y Caso Medio

- El comportamiento de un algoritmo puede variar notablemente para diferentes instancias.

- Suelen estudiarse tres casos para un mismo algoritmo: caso mejor, caso peor, caso medio.

- **Tipos de análisis:**
  - **Peor caso**: indica el mayor tiempo obtenido, teniendo en consideración todas las entradas posibles.
  - **Mejor caso**: indica el menor tiempo obtenido, teniendo en consideración todas las entradas posibles.
  - **Caso medio**: indica el tiempo medio obtenido, considerando todas las entradas posibles.

Retomando el ejemplo 01: Búsqueda lineal

49	12	56	90	2	5	11	32	22	7	99	02	35	1

- **Problema:** Encontrar la posición de un determinado número en un arreglo desordenado.
  - ¿Cuales serían el peor caso, mejor caso y caso promedio?

**Conclusión: Si se conociera la distribución de los datos, podemos sacar provecho de esto, para un mejor análisis y diseño del algoritmo. Por otra parte, sino conocemos la distribución, entonces lo mejor es considerar el peor de los casos.*

Análisis Temporal (mejor, peor y caso medio)

- Con los conceptos anteriores es posible llevar a cabo el análisis temporal de un algoritmo i.e. calcular la **función complejidad temporal** $ft(n)$.

- Considérese el Algoritmo del ejemplo 3 (Búsqueda lineal), para hacer el análisis de su comportamiento:
  - **Operación básica:** Comparaciones con elementos del arreglo
  - **Caso muestra:** A = [2, 7, 4, 1, 3] y n = 5:
    - Si **Valor = 2**, se hace una comparación, $f_t(5) = 1$
    - Si **Valor = 4**, se hacen tres comparaciones, $f_t(5) = 3$
    - Si **Valor = 8**, se hacen cinco comparaciones, y $f_t(5) = 5$

```
func BusquedaLineal(Valor,A,n)
{
        i=1;
        while(i<=n&&A[i]!=Valor)
        {
                i=i+1;
        }
        return i;
}
```

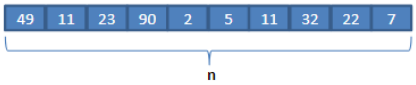
2	7	4	1	3

n=5

- Del análisis anterior es posible descubrir que la **función complejidad temporal** no es tal, **en realidad es una relación**, ya que **para un mismo tamaño de problema se obtienen distintos valores de la función complejidad.**

- Para la mayoría de los algoritmos el número de operaciones depende, no sólo del tamaño del problema, sino también de **la instancia específica que se presente (caso de entrada)**.

```
func BusquedaLineal(Valor,A,n)
{
        i=1;
        while(i<=n&&A[i]!=Valor)
        {
                i=i+1;
        }
        return i;
}
```



→**Caso a: 1 comparación** (Valor = 49)
→**Caso b: 2 comparaciones** (Valor = 11)
→**Caso c: 3 comparaciones** (Valor = 23)
→...
→**Caso x: n comparaciones.** (Valor = 8)

Sea:

- $I(n)=\{I_1, I_2, I_3, \ldots, I_k\}$ el conjunto de instancias del problema de tamaño n.

- $O(n)=\{O_1, O_2, O_3, \ldots, O_k\}$ el conjunto formado por el número de operaciones que un algoritmo realiza para resolver cada instancia.

- Entonces, O_j es el número de operaciones ejecutadas para resolver la instancia I_j, para $1 \leq j \leq k$.

- Se distinguen **tres casos en el valor de la función complejidad temporal**

$$\text{Peor caso } f_t(n) = \text{máx}(\{O_1, O_2, O_3, \ldots, O_k\})$$
$$\text{Mejor caso } f_t(n) = \min(\{O_1, O_2, O_3, \ldots, O_k\})$$
$$\text{Caso medio } f_t(n) = \sum_{i=1}^{k} O_i P(i)$$

$P(i)$ es la probabilidad de que ocurra la instancia I_i

- **El mejor caso** se presenta cuando para **un caso de entrada I_1** a un problema de tamaño ***n;*** el algoritmo ejecuta el **mínimo número posible de operaciones**.

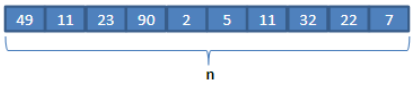
- **El peor caso** se presenta cuando para **un caso de entrada I_2** a un problema de tamaño ***n;*** el algoritmo ejecuta el **máximo número de operaciones**.

- **El caso medio** se consideran todos los casos posibles para calcular el promedio de las operaciones que se hacen tomando en cuenta la probabilidad de que ocurra cada instancia I_3.

Retomando el Ejemplo 1 (Búsqueda lineal)

- **Algoritmo:** Búsqueda lineal
  - **Problema:** Búsqueda lineal de un valor en un arreglo de tamaño ***n:***
  - **Tamaño del Problema**: ***n*** = número de elementos en el arreglo.
  - **Operación básica:** Comparación del valor con los elementos del arreglo *A[i]!=Valor*.

→**Mejor caso 1 comparación** (Ejemplo Buscar: Valor = 49)
→**Peor caso n comparaciones**. (Ejemplo Buscar: Valor = 8)




```
func BusquedaLineal(Valor,A,n)
{
        i=1;
        while(i<=n&&A[i]!=Valor)
        {
                i=i+1;
        }
        return i;
}
```

- **Análisis Temporal**
  - **Mejor caso:** ocurre cuando el valor es el primer elemento del arreglo.

$$f_t(n) = 1 \quad \textit{(Una comparación A[i]!=Valor)}$$

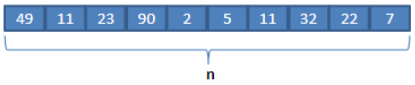
  - **Peor caso:** sucede cuando el valor es el ultimo en el arrelo o no se encuentra en el arreglo.

$$f_t(n) = n \quad \textit{(n comparaciones A[i]!=Valor)}$$

  - **Caso medio:**

$$f_t(n) = 1P(1) + 2P(2) + 3P(3) + 4P(4) + \ldots + nP(n) + nP(n+1)$$

    - Donde $P(i)$ es la probabilidad de que el valor se encuentre en la localidad i; $(1 \leq i \leq n)$ y $P(n+1)$ es la probabilidad de que no esté en el arreglo y el número que lo acompaña es la cantidad de operaciones básicas para cada uno de ellos.



- Si se supone que todos los casos son igualmente probables:

$$P(i) = \frac{1}{n+1}$$

Observación

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

- **Caso médio:**

Número de operaciones básicas por caso

$$f_t(n) = 1P(1) + 2P(2) + 3P(3) + 4P(4) + \ldots + nP(n) + (n)P(n+1) \quad .$$

$$f_t(n) = \frac{1}{n+1}\left(\sum_{i=1}^{n} i + n\right)$$

$$f_t(n) = \frac{1}{n+1}\left(\frac{n(n+1)}{2} + n\right) = \frac{n}{2} + \frac{n}{n+1} = \frac{n^2+n+2n}{2(n+1)} = \frac{n^2+3n}{2(n+1)} = \frac{n(3+n)}{2(n+1)}$$

$$f_t(n) = \frac{n(3+n)}{2(n+1)} \quad \textit{Comparaciones si la probabilidad de todos los casos es la misma}$$

Retomando el Ejemplo 2 (Productos Mayores)

- **Algoritmo:** Producto mayores.
  - **Problema:** Dado un arreglo de valores, encontrar el producto de los dos números mayores.
  - **Tamaño del Problema:** *n* =número de elementos en el arreglo.
  - **Operación básica:** Comparación con los elementos del arreglo y las asignaciones a los elementos mayores.

```
func Producto2Mayores(A,n)
        if(A[1] > A[2])
                mayor1 = A[1];
                mayor2 = A[2];
        else
                mayor1 = A[2];
                mayor2 = A[1];
        i = 3;

        while(i<=n)
                if(A[i] > mayor1)
                        mayor2 = mayor1;
                        mayor1 = A[i];
                else if (A[i] > mayor2)
                        mayor2 = A[i];
                i = i + 1;

        return = mayor1 * mayor2;
```

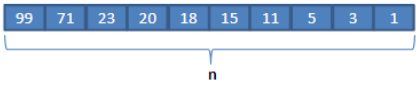

```
func Producto2Mayores(A,n)
        if(A[1] > A[2])                 →Primer comparación
                mayor1 = A[1];          →Asignacion
                mayor2 = A[2];          →Asignacion
        else
                mayor1 = A[2];          →Asignacion
                mayor2 = A[1];          →Asignacion
        i = 3;

        while(i<=n)                     →n-2 Comparaciones
                if(A[i] > mayor1)       →Asignacion
                        mayor2 = mayor1;→Asignacion
                        mayor1 = A[i];  →*Si A[i]≤mayor2
                else if (A[i] > mayor2) →Asignacion
                        mayor2 = A[i];
                i = i + 1;

        return = mayor1 * mayor2;
```

- **Análisis Temporal**
  - **Mejor caso:** ocurre cuando el arreglo está ordenado descendentemente *(se realiza la primer comparación y dos asignaciones, posteriormente solo se compara n-2 veces el if y n-2 veces el else if).*

Comparaciones	Asignaciones
1+(n-2)+(n-2)	2



$$f_t(n)=1+(n-2)+(n-2)+2=3+2(n-2)=2n-1$$

```
func Producto2Mayores(A,n)
        if(A[1] > A[2])
                mayor1 = A[1];
                mayor2 = A[2];
        else
                mayor1 = A[2];
                mayor2 = A[1];
        i = 3;

        while(i<=n)
                if(A[i] > mayor1)
                        mayor2 = mayor1;
                        mayor1 = A[i];
                else if (A[i] > mayor2)
                        mayor2 = A[i];
                i = i + 1;

        return = mayor1 * mayor2;
```

$f_t(2)=2(2)-1=3$

$f_t(3)=2(3)-1=5$

$f_t(5)=2(5)-1=9$

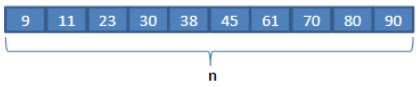
$f_t(10)=2(10)-1=19$

$f_t(20)=2(20)-1=39$

- **Peor caso:** el arreglo está ordenado de manera ascendente *(se realiza la primer comparación y dos asignaciones, posteriormente solo se compara n-2 veces el if y siempre se cumplirá por lo que hará 2(n-2) asignaciones.*

$$f_t(n)=1+(n-2)+2(n-2)+2=3+3(n-2)=3n-3$$

Comparaciones	Asignaciones
1+(n-2)	2+2(n-2)




```
func Producto2Mayores(A,n)
        if(A[1] > A[2])
                mayor1 = A[1];
                mayor2 = A[2];
        else
                mayor1 = A[2];
                mayor2 = A[1];
        i = 3;

        while(i<=n)
                if(A[i] > mayor1)
                        mayor2 = mayor1;
                        mayor1 = A[i];
                else if (A[i] > mayor2)
                        mayor2 = A[i];
                i = i + 1;

        return = mayor1 * mayor2;
```

$f_t(2)=3(2)-3=3$

$f_t(3)=3(3)-3=6$

$f_t(5)=3(5)-3=12$

$f_t(10)=3(10)-3=27$

$f_t(20)=3(20)-3=57$

- **Caso medio:** en este problema se tienen $\binom{|U|}{n} n!$ casos, donde *U* es el conjunto del que se extraen los elementos del arreglo.
  - $\binom{|U|}{n}$ Determina el número de posibles conjuntos de n elementos del conjunto U.
  - $n!$ Determina el número de maneras de acomodar los n elementos
  - Para hacer el cálculo se deben de contar las operaciones que se harían en cada caso. (Laborioso y complicado)

- El algoritmo hace siempre **una comparación al inicio y dos asignaciones** y en el interior del ciclo puede ser que se realice **una comparación con dos asignaciones, dos comparaciones con una asignación o dos comparaciones y ninguna asignación**; obsérvese que **para cada *A[i]*** puede ser cierta una de tres aseveraciones:
  - ***A[i] > mayor1***: Se hace una comparación y dos asignaciones
  - ***A[i]≤mayor1 && A[i]>mayor2:*** Se hacen dos comparaciones y una asignación
  - ***A[i]≤mayor1 && A[i]≤mayor2*** : Se hacen dos comparaciones

- **Caso medio** Si cada caso tiene la misma probabilidad de ocurrencia en promedio se harán:
  - ***Caso: A[i] > mayor1***
  - $\frac{1}{3}\left(1+2+(n-2)+2(n-2)\right)=\frac{1}{3}(3+3(n-2))=\frac{1}{3}(3n-3)$
  - ***Caso: A[i]≤mayor1 && A[i]>mayor2***
  - $\frac{1}{3}\left(1+2+2(n-2)+(n-2)\right)=\frac{1}{3}(3+3(n-2))=\frac{1}{3}(3n-3)$
  - ***Caso: A[i]≤mayor1 && A[i]≤mayor2***
  - $\frac{1}{3}\left(1+2+(n-2)+(n-2)\right)=\frac{1}{3}(3+2(n-2))=\frac{1}{3}(2n-1)$

$f_t(2)=3$

$f_t(3)=5.6$

$f_t(5)=11$

$f_t(10)=24.3$

$f_t(20)=51$

- **Caso medio:**

$$f_t(n)=\frac{1}{3}(2(3n-3)+(2n-1))=\frac{1}{3}(8n-7)$$

$$\boldsymbol{f_t(n)=\frac{8n-7}{3}\ operaciones\ básicas}$$

- Es importante mencionar que no todos los algoritmos presentan casos que hacen variar la complejidad temporal para un tamaño de problema especifico, y resulta interesante tener una herramienta de análisis para detectar cuándo se particionará el análisis en casos; por el momento la única ayuda con la que se cuenta es la intuición y preguntarse: **¿Se puede resolver el problema de manera trivial para alguna instancia específica?**. Si la respuesta es afirmativa el algoritmo tendrá casos, por lo que el problema de detección se reduce a contestar esta "simple" pregunta.