

# Fundamentos y Estructuras de Programación

## Nociones de Complejidad Computacional

Carlos Alberto Ramirez Restrepo

Programa de Ingeniería de Sistemas  
Pontificia Universidad Javeriana  
Cali, Colombia  
`carlosalbertoramirez@javerianacali.edu.co`

# Plan

- 1 Generalidades
- 2 Cálculo de complejidad por conteo
- 3 Cálculo de complejidad por inspección

# Plan

- 1 Generalidades
- 2 Cálculo de complejidad por conteo
- 3 Cálculo de complejidad por inspección

# Generalidades

- ✓ Como se vio anteriormente, de forma general, un problema se denomina **no decidable** cuando no es posible encontrar un algoritmo que lo resuelva.

# Generalidades

- ✓ Como se vio anteriormente, de forma general, un problema se denomina **no decidable** cuando no es posible encontrar un algoritmo que lo resuelva.
- ✓ Los problemas para los cuales existe un algoritmo que los resuelva se denominan **decidibles**.

# Generalidades

- ✓ Como se vio anteriormente, de forma general, un problema se denomina **no decidable** cuando no es posible encontrar un algoritmo que lo resuelva.
- ✓ Los problemas para los cuales existe un algoritmo que los resuelva se denominan **decidibles**.
- ✓ Por otro lado, los problemas computacionales pueden ser también clasificados de acuerdo a su complejidad en problemas **tratables** y problemas **intratables**.

# Generalidades

- ✓ Como se vio anteriormente, de forma general, un problema se denomina **no decidable** cuando no es posible encontrar un algoritmo que lo resuelva.
- ✓ Los problemas para los cuales existe un algoritmo que los resuelva se denominan **decidibles**.
- ✓ Por otro lado, los problemas computacionales pueden ser también clasificados de acuerdo a su complejidad en problemas **tratables** y problemas **intratables**.
- ✓ La **complejidad** de un algoritmo mide el grado u orden de crecimiento que tiene el tiempo de ejecución del algoritmo dado el tamaño de la entrada que tenga.

# Generalidades

- ✓ Un problema es denominado **intratable** si, con entradas grandes, no puede ser resuelto por ningún computador, no importa lo rápido que sea, cuanta memoria tenga o cuanto tiempo se le de para que lo resuelva.



# Generalidades

- ✓ Un problema es denominado **intratable** si, con entradas grandes, no puede ser resuelto por ningún computador, no importa lo rápido que sea, cuanta memoria tenga o cuanto tiempo se le de para que lo resuelva.
- ✓ Lo anterior sucede debido a que los algoritmos que existen para solucionar estos problemas tienen una complejidad muy grande.

# Generalidades

Existen dos maneras de hallar la complejidad de un algoritmo:

1. por **conteo**, ó

# Generalidades

Existen dos maneras de hallar la complejidad de un algoritmo:

1. por **conteo**, ó
2. por **inspección** o **tanteo**.

# Plan

- 1 Generalidades
- 2 Cálculo de complejidad por conteo
- 3 Cálculo de complejidad por inspección

# Cálculo de complejidad

## Cálculo de complejidad por conteo

- ✓ Para encontrar la complejidad de un algoritmo por conteo se debe tomar cada línea de código y determinar cuántas veces se ejecuta.

# Cálculo de complejidad

## Cálculo de complejidad por conteo

- ✓ Para encontrar la complejidad de un algoritmo por conteo se debe tomar cada línea de código y determinar cuántas veces se ejecuta.
- ✓ Luego, se debe sumar las cantidades encontradas y la complejidad será del orden del resultado dado.

# Cálculo de complejidad

## Cálculo de complejidad por conteo

- ✓ Para encontrar la complejidad de un algoritmo por conteo se debe tomar cada línea de código y determinar cuántas veces se ejecuta.
- ✓ Luego, se debe sumar las cantidades encontradas y la complejidad será del orden del resultado dado.
- ✓ Esta complejidad es una aproximación de cuánto se demorará todo el algoritmo en ejecutarse.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo

Considere el siguiente algoritmo:

```
void imprime100(){  
    int i = 1;  
    while(i <= 100){  
        printf("%d",i);  
        i++;  
    }  
}
```



# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo

Para calcular la complejidad de este algoritmo por conteo se puede utilizar una tabla donde se numeran las líneas de código y se determine el número de veces que se ejecuta cada una:

Número de línea	Línea de código	Número de ejecuciones
1	void imprime100(){	
2	int i = 1;	1
3	while(i <= 100){	101
4	printf("%d",i);	100
5	i++;	100
6	}	
7	}	

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo

Para calcular la complejidad de este algoritmo por conteo se puede utilizar una tabla donde se numeran las líneas de código y se determine el número de veces que se ejecuta cada una:

Número de línea	Línea de código	Número de ejecuciones
1	void imprime100(){	
2	int i = 1;	1
3	while(i <= 100){	101
4	printf("%d",i);	100
5	i++;	100
6	}	
7	}	

La suma de las cantidades encontradas es:

$$\text{Número total ejecuciones} = 1 + 101 + 100 + 100 = 302$$

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo

- ✓ Dado que 302 es una constante, luego se tiene que la complejidad del algoritmo anterior es  $O(1)$ .

## Cálculo de complejidad

### Cálculo de complejidad por conteo - Ejemplo

- ✓ Dado que 302 es una constante, luego se tiene que la complejidad del algoritmo anterior es  $O(1)$ .
- ✓  $O(1)$  es la notación utilizada para expresar que el tiempo de computo de un algoritmo es una función constante con respecto al tamaño de la entrada.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 2

Considere el siguiente algoritmo:

```
void imprimeN(int n){  
    int i = 1;  
    while(i <= n){  
        printf("%d",i);  
        i++;  
    }  
}
```

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo

Se enumeran las líneas y se procede a contabilizar:

Número de línea	Línea de código	Número de ejecuciones
1	void imprimeN(int n){	
2	int i = 1;	1
3	while(i <= n){	$n + 1$
4	printf("%d",i);	$n$
5	i++;	$n$
6	}	
7	}	

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo

Se enumeran las líneas y se procede a contabilizar:

Número de línea	Línea de código	Número de ejecuciones
1	void imprimeN(int n){	
2	int i = 1;	1
3	while(i <= n){	$n + 1$
4	printf("%d",i);	$n$
5	i++;	$n$
6	}	
7	}	

La suma de las cantidades encontradas es:

$$\text{Número total ejecuciones} = 1 + (n + 1) + n + n = 3n + 2$$

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo

- ✓ Dado que  $3n + 2$  es una función lineal, luego se tiene que la complejidad del algoritmo anterior es  $O(n)$ .



# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo

- ✓ Dado que  $3n + 2$  es una función lineal, luego se tiene que la complejidad del algoritmo anterior es  $O(n)$ .
- ✓  $O(n)$  es la notación utilizada para expresar que el tiempo de computo del algoritmo es una función lineal del tamaño de la entrada.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 3

Considere el siguiente algoritmo:

```
void imprime_mitad(int n){  
    int i = n;  
    while(i > 0){  
        printf("%d",i);  
        i = i / 2;  
    }  
}
```

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 3

- ✓ A primera vista, el algoritmo anterior es similar a los algoritmos de los ejemplos anteriores.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 3

- ✓ A primera vista, el algoritmo anterior es similar a los algoritmos de los ejemplos anteriores.
- ✓ No obstante, en cada iteración el valor de  $i$  es reducido a la mitad.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 3

- ✓ A primera vista, el algoritmo anterior es similar a los algoritmos de los ejemplos anteriores.
- ✓ No obstante, en cada iteración el valor de  $i$  es reducido a la mitad.
- ✓ Entonces en lugar de imprimirse los números de 1 a  $n$ , en realidad se imprimirán los números  $n, n/2, n/4, n/8, \dots, n/2^k$

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 3

- ✓ De lo anterior, es requerido encontrar la iteración en la cual el algoritmo finaliza, osea el punto donde  $n = 2^k$ .

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 3

- ✓ De lo anterior, es requerido encontrar la iteración en la cual el algoritmo finaliza, osea el punto donde  $n = 2^k$ .
- ✓ Para hallar el valor de  $k$ , se utilizan las propiedades de los logaritmos:

$$n = 2^k$$

$$\log_2 n = \log_2 2^k$$

$$\log_2 n = k$$

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 3

- ✓ De lo anterior, es requerido encontrar la iteración en la cual el algoritmo finaliza, osea el punto donde  $n = 2^k$ .
- ✓ Para hallar el valor de  $k$ , se utilizan las propiedades de los logaritmos:

$$n = 2^k$$

$$\log_2 n = \log_2 2^k$$

$$\log_2 n = k$$

- ✓ En consecuencia, el número de iteraciones que se realizan es  $\log_2 n$ .



# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 3

De esta manera se tiene lo siguiente:

Número de línea	Línea de código	Número de ejecuciones
1	void imprime_mitad(int n){	
2	int i = n;	1
3	while(i > 0){	$\log_2 n + 2$
4	printf("%d",i);	$\log_2 n + 1$
5	i = i/2;	$\log_2 n + 1$
6	}	
7	}	

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 3

De esta manera se tiene lo siguiente:

Número de línea	Línea de código	Número de ejecuciones
1	void imprime_mitad(int n){	
2	int i = n;	1
3	while(i > 0){	$\log_2 n + 2$
4	printf("%d",i);	$\log_2 n + 1$
5	i = i/2;	$\log_2 n + 1$
6	}	
7	}	

La suma de las cantidades encontradas es:

$$\begin{aligned}
 \text{Número total ejecuciones} &= 1 + (\log_2 n + 2) + (\log_2 n + 1) + (\log_2 n + 1) \\
 &= 3\log_2 n + 5
 \end{aligned}$$

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 3

- ✓ Dado que  $3\log_2 n + 5$  es una función logarítmica, luego se tiene que la complejidad del algoritmo anterior es  $O(\log_2 n)$ .

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 3

- ✓ Dado que  $3\log_2 n + 5$  es una función logarítmica, luego se tiene que la complejidad del algoritmo anterior es  $O(\log_2 n)$ .
- ✓  $O(\log_2 n)$  es la notación utilizada para expresar que el tiempo de computo del algoritmo es una función logarítmica del tamaño de la entrada.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 4

Considere el siguiente algoritmo:

```
void imprimeNxN(int n){  
    for(int i=1; i<=n; i++){  
        for(int j=1; j<=n; j++){  
            printf("%d", (i-1)*n + j);  
        }  
    }  
}
```

Qué hace este algoritmo?

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 4

Se enumeran las líneas y se procede a contabilizar:

# Línea	Línea de código	Número de ejecuciones
1	void imprimeNxN(int n){	
2	for(int i=1; i<=n; i++){	$n + 1$
3	for(int j=1; j<=n; j++){	$n * (n + 1)$
4	printf("%d", (i-1)*n + j);	$n * n$
5	}	
6	}	
7	}	

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 4

Se enumeran las líneas y se procede a contabilizar:

# Línea	Línea de código	Número de ejecuciones
1	void imprimeNxN(int n){	
2	for(int i=1; i<=n; i++){	$n + 1$
3	for(int j=1; j<=n; j++){	$n * (n + 1)$
4	printf("%d", (i-1)*n + j);	$n * n$
5	}	
6	}	
7	}	

La suma de las cantidades encontradas es:

$$\text{Número total ejecuciones} = (n + 1) + n * (n + 1) + n * n = 2n^2 + 2n + 1$$

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 4

- ✓ Dado que  $2n^2 + 2n + 1$  es una función cuadrática, luego se tiene que la complejidad del algoritmo anterior es  $O(n^2)$ .



# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 4

- ✓ Dado que  $2n^2 + 2n + 1$  es una función cuadrática, luego se tiene que la complejidad del algoritmo anterior es  $O(n^2)$ .
- ✓  $O(n^2)$  es la notación utilizada para expresar que el tiempo de computo del algoritmo es una función cuadrática del tamaño de la entrada.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

Considere el siguiente algoritmo:

```
int sumaVector(int *v, int n){  
    int i = 0;  
    int sum = 0;  
    while(i < n){  
        if(v[i] % 2 != 0)  
            sum = sum + v[i];  
        i++;  
    }  
    return sum;  
}
```

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

Considere el siguiente algoritmo:

```
int sumaVector(int *v, int n){  
    int i = 0;  
    int sum = 0;  
    while(i < n){  
        if(v[i] % 2 != 0)  
            sum = sum + v[i];  
        i++;  
    }  
    return sum;  
}
```

Qué cálculo realiza este algoritmo?

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

- ✓ Cuando se analizan algoritmos con condicionales (como el algoritmo anterior) hay que tener en cuenta que el conteo se hace considerando si las guardas de los condicionales se cumplen o no.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

- ✓ Cuando se analizan algoritmos con condicionales (como el algoritmo anterior) hay que tener en cuenta que el conteo se hace considerando si las guardas de los condicionales se cumplen o no.
- ✓ La complejidad en estos algoritmos se halla:

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

- ✓ Cuando se analizan algoritmos con condicionales (como el algoritmo anterior) hay que tener en cuenta que el conteo se hace considerando si las guardas de los condicionales se cumplen o no.
- ✓ La complejidad en estos algoritmos se halla:
  - en el **peor** de los casos (cuando se asume que las guardas de los condicionales siempre se cumplen),

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

- ✓ Cuando se analizan algoritmos con condicionales (como el algoritmo anterior) hay que tener en cuenta que el conteo se hace considerando si las guardas de los condicionales se cumplen o no.
- ✓ La complejidad en estos algoritmos se halla:
  - en el **peor** de los casos (cuando se asume que las guardas de los condicionales siempre se cumplen),
  - el caso **promedio** (cuando se asume que las guardas algunas veces se cumplen y otras veces no)

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

- ✓ Cuando se analizan algoritmos con condicionales (como el algoritmo anterior) hay que tener en cuenta que el conteo se hace considerando si las guardas de los condicionales se cumplen o no.
- ✓ La complejidad en estos algoritmos se halla:
  - en el **peor** de los casos (cuando se asume que las guardas de los condicionales siempre se cumplen),
  - el caso **promedio** (cuando se asume que las guardas algunas veces se cumplen y otras veces no)y
  - el **mejor** de los casos (cuando se asume que las guardas no se cumplen).



# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

Se enumeran las líneas y se procede a contabilizar:

# Línea	Línea de código	Número de ejecuciones
1	int sumaVector(int *v, int n){	
2	int i = 0;	1
3	int sum = 0;	1
4	while(i < n){	$n + 1$
5	if(v[i] % 2 != 0)	$n$
6	sum = sum + v[i];	?
7	i++;	$n$
8	}	
9	return sum;	1
10	}	

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

- ✓ La cantidad de veces que se ejecuta la línea 6 es indefinida debido a que depende de si la guarda del condicional es verdadera o falsa.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

- ✓ La cantidad de veces que se ejecuta la línea 6 es indefinida debido a que depende de si la guarda del condicional es verdadera o falsa.
- ✓ Por esta razón tenemos que analizar esta situación desde los tres casos:

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

- ✓ La cantidad de veces que se ejecuta la línea 6 es indefinida debido a que depende de si la guarda del condicional es verdadera o falsa.
- ✓ Por esta razón tenemos que analizar esta situación desde los tres casos:
  - En el **mejor** de los casos ningún elemento del vector es impar por lo que la línea 6 no se ejecutaría nunca.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

- ✓ La cantidad de veces que se ejecuta la línea 6 es indefinida debido a que depende de si la guarda del condicional es verdadera o falsa.
- ✓ Por esta razón tenemos que analizar esta situación desde los tres casos:
  - En el **mejor** de los casos ningún elemento del vector es impar por lo que la línea 6 no se ejecutaría nunca.
  - En el caso **promedio**, aproximadamente la mitad de los elementos será impar y la otra mitad par. En este caso la línea se ejecutaría  $n/2$  veces.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

- ✓ La cantidad de veces que se ejecuta la línea 6 es indefinida debido a que depende de si la guarda del condicional es verdadera o falsa.
- ✓ Por esta razón tenemos que analizar esta situación desde los tres casos:
  - En el **mejor** de los casos ningún elemento del vector es impar por lo que la línea 6 no se ejecutaría nunca.
  - En el caso **promedio**, aproximadamente la mitad de los elementos será impar y la otra mitad par. En este caso la línea se ejecutaría  $n/2$  veces.
  - En el **peor** caso todos los elementos del vector son impares por lo que siempre que se ejecute la línea 5 se ejecutará la línea 6. Luego esta línea se ejecutará  $n$  veces.

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

La suma de las cantidades encontradas es entonces:

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

La suma de las cantidades encontradas es entonces:

- ✓ En el mejor de los casos:

$$\text{Número total ejecuciones} = 1 + 1 + (n + 1) + n + \mathbf{0} + n + 1 = 3n + 4$$

- ✓ En el caso promedio:

$$\text{Número total ejecuciones} = 1 + 1 + (n + 1) + n + \mathbf{n/2} + n + 1 = 7/2n + 4$$



# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 5

La suma de las cantidades encontradas es entonces:

- ✓ En el mejor de los casos:

$$\text{Número total ejecuciones} = 1 + 1 + (n + 1) + n + 0 + n + 1 = 3n + 4$$

- ✓ En el caso promedio:

$$\text{Número total ejecuciones} = 1 + 1 + (n + 1) + n + n/2 + n + 1 = 7/2n + 4$$

- ✓ En el peor de los casos:

$$\text{Número total ejecuciones} = 1 + 1 + (n + 1) + n + n + n + 1 = 4n + 4$$

Por lo tanto, la complejidad del algoritmo es, en este algoritmo particular,  $O(n)$ .

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

Considere ahora el siguiente algoritmo más complejo:

```
void algoritmo(int *a, int n){  
    for(int j=1; j<n; j++){  
        int clave = a[j];  
        int i = j-1;  
        while(i>=0 && a[i] > clave){  
            a[i+1] = a[i];  
            i = i-1;  
        }  
        a[i+1] = clave;  
    }  
}
```

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

Considere ahora el siguiente algoritmo más complejo:

```
void algoritmo(int *a, int n){  
    for(int j=1; j<n; j++){  
        int clave = a[j];  
        int i = j-1;  
        while(i>=0 && a[i] > clave){  
            a[i+1] = a[i];  
            i = i-1;  
        }  
        a[i+1] = clave;  
    }  
}
```

Qué cálculo realiza este algoritmo?

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

Se enumeran las líneas y se procede a contabilizar:

# Línea	Línea de código	Número de ejecuciones
1	void algoritmo(int *a, int n){	
2	for(int j=1; j<n; j++){	$n$
3	int clave = a[j];	$n - 1$
4	int i = j-1;	$n - 1$
5	while(i>=0 && a[i] > clave){	$t_1 + t_2 + \dots + t_{n-1}$
6	a[i+1] = a[i];	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
7	i = i-1;	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
8	}	
9	a[i+1] = clave;	$n - 1$
10	}	
11	}	

Donde cada  $t_j$  corresponde al número de veces que se cumple la condición del **while** en la iteración  $j$ .

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

Se enumeran las líneas y se procede a contabilizar:

# Línea	Línea de código	Número de ejecuciones
1	void algoritmo(int *a, int n){	
2	for(int j=1; j<n; j++){	$n$
3	int clave = a[j];	$n - 1$
4	int i = j-1;	$n - 1$
5	while(i>=0 && a[i] > clave){	$t_1 + t_2 + \dots + t_{n-1}$
6	a[i+1] = a[i];	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
7	i = i-1;	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
8	}	
9	a[i+1] = clave;	$n - 1$
10	}	
11	}	

En el **mejor** caso cuánto vale cada  $t_j$ ?

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

Se enumeran las líneas y se procede a contabilizar:

# Línea	Línea de código	Número de ejecuciones
1	void algoritmo(int *a, int n){	
2	for(int j=1; j<n; j++){	$n$
3	int clave = a[j];	$n - 1$
4	int i = j-1;	$n - 1$
5	while(i>=0 && a[i] > clave){	$t_1 + t_2 + \dots + t_{n-1}$
6	a[i+1] = a[i];	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
7	i = i-1;	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
8	}	
9	a[i+1] = clave;	$n - 1$
10	}	
11	}	

En el **mejor** caso,  $t_j = 1$ .

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

Se enumeran las líneas y se procede a contabilizar:

# Línea	Línea de código	Número de ejecuciones
1	void algoritmo(int *a, int n){	
2	for(int j=1; j<n; j++){	$n$
3	int clave = a[j];	$n - 1$
4	int i = j-1;	$n - 1$
5	while(i>=0 && a[i] > clave){	$t_1 + t_2 + \dots + t_{n-1}$
6	a[i+1] = a[i];	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
7	i = i-1;	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
8	}	
9	a[i+1] = clave;	$n - 1$
10	}	
11	}	

En el **peor** caso cuánto vale cada  $t_j$ ?

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

Se enumeran las líneas y se procede a contabilizar:

# Línea	Línea de código	Número de ejecuciones
1	void algoritmo(int *a, int n){	
2	for(int j=1; j<n; j++){	$n$
3	int clave = a[j];	$n - 1$
4	int i = j-1;	$n - 1$
5	while(i>=0 && a[i] > clave){	$t_1 + t_2 + \dots + t_{n-1}$
6	a[i+1] = a[i];	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
7	i = i-1;	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
8	}	
9	a[i+1] = clave;	$n - 1$
10	}	
11	}	

En el **peor** caso,  $t_j = j$ .



# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

Se enumeran las líneas y se procede a contabilizar:

# Línea	Línea de código	Número de ejecuciones
1	void algoritmo(int *a, int n){	
2	for(int j=1; j<n; j++){	$n$
3	int clave = a[j];	$n - 1$
4	int i = j-1;	$n - 1$
5	while(i>=0 && a[i] > clave){	$t_1 + t_2 + \dots + t_{n-1}$
6	a[i+1] = a[i];	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
7	i = i-1;	$(t_1 - 1) + \dots + (t_{n-1} - 1)$
8	}	
9	a[i+1] = clave;	$n - 1$
10	}	
11	}	

En el caso promedio, se supone que se necesitan  $j/2$  comparaciones, esto es,  $t_j = j/2$ .

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

La suma de las cantidades encontradas es entonces:

✓ En el mejor caso:

$$\begin{aligned}\text{Total} &= n + (n - 1) + (n - 1) + \sum_{j=1}^{n-1} \mathbf{1} + \sum_{j=1}^{n-1} \mathbf{0} + \sum_{j=1}^{n-1} \mathbf{0} + (n - 1) \\ &= n + (n - 1) + (n - 1) + (n - 1) + (n - 1) = 5n - 4\end{aligned}$$

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

La suma de las cantidades encontradas es entonces:

✓ En el mejor caso:

$$\begin{aligned}\text{Total} &= n + (n - 1) + (n - 1) + \sum_{j=1}^{n-1} \mathbf{1} + \sum_{j=1}^{n-1} \mathbf{0} + \sum_{j=1}^{n-1} \mathbf{0} + (n - 1) \\ &= n + (n - 1) + (n - 1) + (n - 1) + (n - 1) = 5n - 4\end{aligned}$$

Por lo tanto, la complejidad del algoritmo es en el mejor caso  $O(n)$ .

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

La suma de las cantidades encontradas es entonces:

✓ En el peor caso:

$$\begin{aligned}
 \text{Total} &= n + (n - 1) + (n - 1) + \sum_{j=1}^{n-1} j + \sum_{j=1}^{n-1} j - 1 + \sum_{j=1}^{n-1} j - 1 + (n - 1) \\
 &= n + 3(n - 1) + n(n + 1)/2 + 2(n(n + 1)/2 - (n - 1)) \\
 &= n + 3(n - 1) + n(n + 1)/2 + n(n + 1) - 2(n - 1) \\
 &= n + (n - 1) + n(n + 1)/2 + n(n + 1) \\
 &= 2n - 1 + n^2/2 + n/2 + n^2 + n = 3n^2/2 + 7n/2 - 1
 \end{aligned}$$

# Cálculo de complejidad

## Cálculo de complejidad por conteo - Ejemplo 6

La suma de las cantidades encontradas es entonces:

✓ En el peor caso:

$$\begin{aligned}
 \text{Total} &= n + (n-1) + (n-1) + \sum_{j=1}^{n-1} j + \sum_{j=1}^{n-1} j - 1 + \sum_{j=1}^{n-1} j - 1 + (n-1) \\
 &= n + 3(n-1) + n(n+1)/2 + 2(n(n+1)/2 - (n-1)) \\
 &= n + 3(n-1) + n(n+1)/2 + n(n+1) - 2(n-1) \\
 &= n + (n-1) + n(n+1)/2 + n(n+1) \\
 &= 2n - 1 + n^2/2 + n/2 + n^2 + n = 3n^2/2 + 7n/2 - 1
 \end{aligned}$$

Por lo tanto, la complejidad del algoritmo es en el peor caso  $O(n^2)$ .

# Plan

- 1 Generalidades
- 2 Cálculo de complejidad por conteo
- 3 Cálculo de complejidad por inspección**

# Cálculo de complejidad

## Cálculo de complejidad por inspección

- ✓ Hallar la complejidad por inspección o tanteo, es un método más rápido pero impreciso y, si no se cuenta con la suficiente experiencia, poco confiable.

# Cálculo de complejidad

## Cálculo de complejidad por inspección

- ✓ Hallar la complejidad por inspección o tanteo, es un método más rápido pero impreciso y, si no se cuenta con la suficiente experiencia, poco confiable.
- ✓ Simplemente se mira la estructura del algoritmo y se siguen las tres siguientes reglas:



# Cálculo de complejidad

## Cálculo de complejidad por inspección

- ✓ Hallar la complejidad por inspección o tanteo, es un método más rápido pero impreciso y, si no se cuenta con la suficiente experiencia, poco confiable.
- ✓ Simplemente se mira la estructura del algoritmo y se siguen las tres siguientes reglas:
  1. La complejidad de una asignación es  $O(1)$ .

# Cálculo de complejidad

## Cálculo de complejidad por inspección

- ✓ Hallar la complejidad por inspección o tanteo, es un método más rápido pero impreciso y, si no se cuenta con la suficiente experiencia, poco confiable.
- ✓ Simplemente se mira la estructura del algoritmo y se siguen las tres siguientes reglas:
  1. La complejidad de una asignación es  $O(1)$ .
  2. La complejidad de un condicional es 1 más el máximo entre la complejidad del cuerpo del condicional cuando la guarda es positiva y el cuerpo del condicional cuando la guarda es negativa.

# Cálculo de complejidad

## Cálculo de complejidad por inspección

- ✓ Hallar la complejidad por inspección o tanteo, es un método más rápido pero impreciso y, si no se cuenta con la suficiente experiencia, poco confiable.
- ✓ Simplemente se mira la estructura del algoritmo y se siguen las tres siguientes reglas:
  1. La complejidad de una asignación es  $O(1)$ .
  2. La complejidad de un condicional es 1 más el máximo entre la complejidad del cuerpo del condicional cuando la guarda es positiva y el cuerpo del condicional cuando la guarda es negativa.
  3. La complejidad de un ciclo es el número de veces que se ejecuta el ciclo multiplicado por la complejidad del cuerpo del ciclo.

# Cálculo de complejidad

## Complejidades

A continuación se resumen las diferentes complejidades que puede tener un algoritmo:

Complejidad	Nombre
$O(1)$	Constante
$O(\log n)$	Logarítmica
$O(n)$	Lineal
$O(n \log n)$	
$O(n^2)$	Cuadrática
$O(n^3)$	Cúbica
$O(n^c), c > 3$	Polinomial
$O(2^n)$	
$O(3^n)$	
$O(c^n), c > 3$	Exponencial
$O(n!)$	Factorial
$O(n^n)$	

# Cálculo de complejidad

## Complejidades

A continuación se resumen las diferentes complejidades que puede tener un algoritmo:

Complejidad	Nombre
$O(1)$	Constante
$O(\log n)$	Logarítmica
$O(n)$	Lineal
$O(n \log n)$	
$O(n^2)$	Cuadrática
$O(n^3)$	Cúbica
$O(n^c), c > 3$	Polinomial
$O(2^n)$	
$O(3^n)$	
$O(c^n), c > 3$	Exponencial
$O(n!)$	Factorial
$O(n^n)$	

Se dice que un problema es tratable si su complejidad es polinomial o menor.

# Introducción

## Ejercicio

Calcule la complejidad del siguiente algoritmo:

```
int programa(int n){  
    int s = 0;  
    int i = 1;  
    while(i <= n){  
        int t = 0;  
        int j = 1;  
        while(j <= i){  
            t = t + 1;  
            j = j + 1;  
        }  
        s = s + t;  
        i = i + 1;  
    }  
    return s;  
}
```

# Introducción

## Ejercicio

Calcule la complejidad del siguiente algoritmo:

```
Algoritmo(valores[1..n]){  
    suma=0  
    contador=0  
    for(i=1; i<n; i++){  
        for(j=i+1; j<=n; j++){  
            if(valores[i] < valores[j]){  
                for(h=j; h<=n; h++){  
                    suma += valores[i]  
                }  
            }  
            else{  
                contador++  
                break;  
            }  
        }  
    }  
    return contador  
}
```

# Preguntas

?