



POLITÉCNICO COLOMBIANO
JAIME ISAZA CADAVID
INSTITUCIÓN UNIVERSITARIA

Educación para
vivir mejor

Algoritmos y Programación 4

Clase 11
Análisis de Algoritmos



GOBERNACIÓN DE ANTIOQUIA



VIGILADA MINEDUCACIÓN

Análisis de desempeño

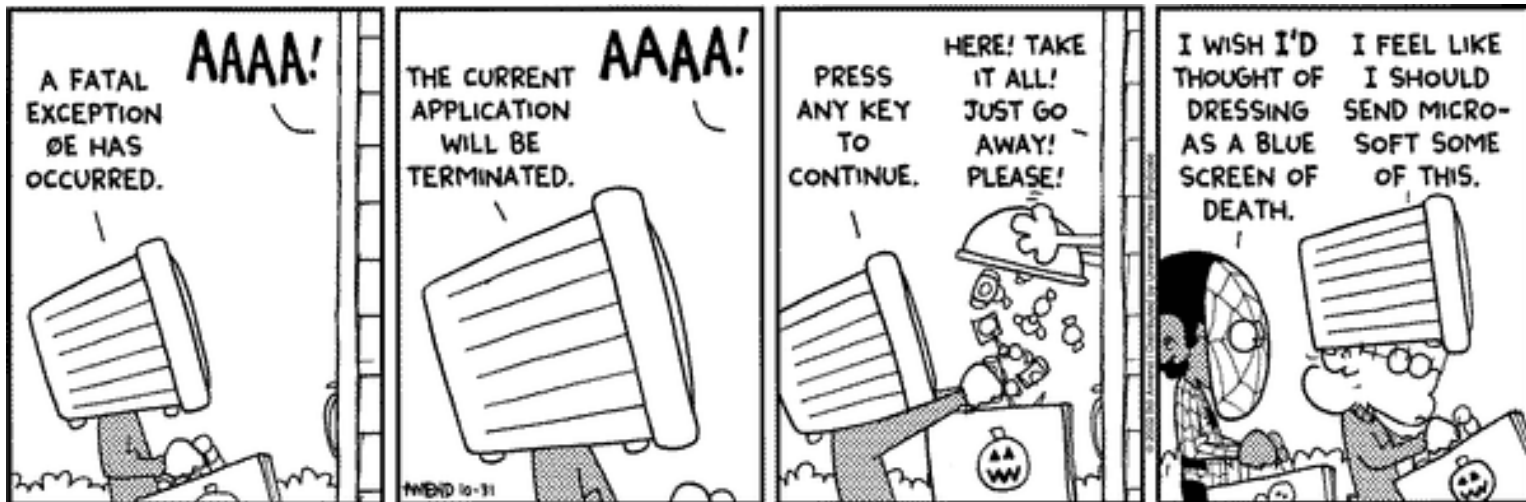


Imagen tomada de <https://foxtrot.com/>



Para que analizar algoritmos

- Conforme aumenta la experiencia en el uso de computadores, estos comienzan a emplearlos para resolver problemas complejos o procesar grandes volúmenes de información.
- En este proceso, es común que surjan preguntas como las siguientes:
 - ¿Cuánto tiempo tomará la ejecución de mi programa?
 - ¿Por qué mi programa se queda sin memoria?



Programador

Necesita desarrollar una solución que trabaje



Cliente

Desea resolver el problema de manera eficiente



Científico

Desea aprender sobre algo



Estudiante

Para que analizar algoritmos

Razones:

- Predecir desempeño
- Comparar algoritmos.
- Brindar garantías:
- Comprender bases teóricas.
- **Razón principal practica:** Evitar bugs de desempeño



Realidad

El cliente obtiene [o experimenta] un rendimiento deficiente debido a que el programador no comprendió las características de rendimiento



[link](#)



[link](#)



Desafío

Pregunta: ¿Mi programa podrá resolver una entrada práctica de gran tamaño?

¿Por qué mi programa
es tan lento?



¿Por qué mi programa
se queda sin memoria?

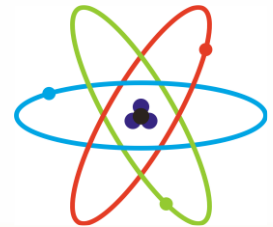
"Esta es la idea fundamental: debemos usar el **método científico** para entender el rendimiento".

Donald Knuth (1970)



Método científico aplicado al análisis de algoritmos

- El **método científico** son un conjunto de técnicas comúnmente aceptadas y universalmente utilizadas por los científicos para desarrollar conocimientos sobre el mundo natural.
- El método científico nos proporciona un **framework** (marco de trabajo) para predecir el desempeño y comparar algoritmos.
- **Pasos:**
 1. **Observar** alguna característica del mundo natural, generalmente con mediciones precisas.
 2. **Formular** un modelo que sea consistente con las observaciones.
 3. **Predecir** eventos usando la hipótesis.
 4. **Verificar** las predicciones realizando [o mediante] observaciones adicionales.
 5. **Validar** repitiendo hasta que la hipótesis y las observaciones concuerden.
- **Principios:**
 - Los experimentos deben ser **reproducibles**
 - Las hipótesis que formulamos deben ser **falsificables**.



Ejemplo: 3-SUM

- Dados **N** números enteros distintos, ¿cuántas ternas suman exactamente cero?

```
30
-30
-20
-10
40
0
10
15
```

```
Leyendo datos del archivo: 8ints.txt
Arreglo de 8 enteros leído.
Contando ternas que suman 0...
4
30 -30 0
30 -20 -10
-30 -10 40
-10 0 10
```

	a[i]	a[j]	a[k]	sum
1	30	-40	10	0
2	30	-20	-10	0
3	-40	40	0	0
4	-10	0	10	0



3-SUM: Algoritmo a fuerza bruta

- Dados **N** números enteros distintos, ¿cuántas ternas suman exactamente cero?

```
def countTriples(a):  
    n = len(a)  
    count = 0  
    for i in range(n):  
        for j in range(i+1, n):  
            for k in range(j+1, n):  
                if (a[i] + a[j] + a[k]) == 0:  
                    count += 1  
  
    return count  
  
def main():  
    file = "8ints.txt"  
    print(f"Leyendo datos del archivo: {file}")  
    a = readInt1D(file)  
    print(f"Arreglo de {len(a)} enteros leído.")  
    print(f"Contando ternas que suman 0... ")  
    count = countTriples(a)  
    print(count)  
    if count < 10:  
        writeTriples(a)  
  
if __name__ == '__main__':  
    main()
```

Selección de cada terna

Verificación de la condición
para la terna seleccionada



Midiendo el tiempo de ejecución

- Para determinar el rendimiento de un programa, el primer desafío consiste en determinar cómo realizar mediciones cuantitativas del tiempo de ejecución.
- Existen diversas herramientas disponibles para ayudarnos a obtener aproximaciones, donde la mas simple es emplear un **cronometro** sencillo y realizar las mediciones **manualmente**.

```
# --- Inicio del cronometro ---  
# Operaciones (toman un tiempo determinado)  
# --- Detención del cronometro ---
```



Midiendo el tiempo

- Dados **N** números enteros distintos, ¿cuántas ternas suman exactamente cero?

```

Leyendo datos del archivo: 1Kints.txt
Arreglo de 1000 enteros leído.
Contando ternas que suman 0...
70

```

00:28 = 28 seg



```

Leyendo datos del archivo: 2Kints.txt
Arreglo de 2000 enteros leído.
Contando ternas que suman 0...
528

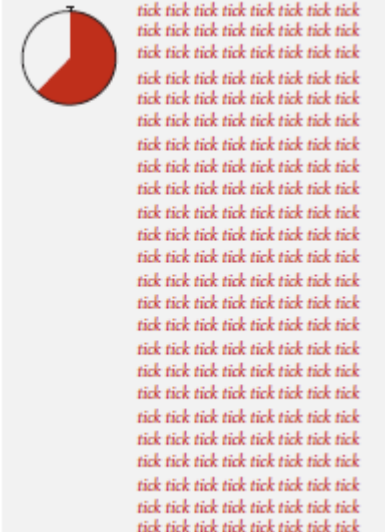
```

03:45 = 225 seg



```
Leyendo datos del archivo: 4Kints.txt
Arreglo de 4000 enteros leído.
Contando ternas que suman 0...
4039
```

31:37 = 1897 seg



Midiendo el tiempo de ejecución

- Como realizar las mediciones manualmente de tiempo es muy engorroso, es mejor recurrir a una **forma automática**, empleando **marcas de tiempo**.
- Las marcas de tiempo son capturas de tiempo realizadas al principio y al final para determinar el tiempo que toma la ejecución de las operaciones entre estas.
- Para realizar la medición la forma mas simple se emplea un cronómetro el cual es modelado como la clase Stopwatch (stopwatch.py) cuyo API se muestra a continuación:

API	operación	Descripción
	Stopwatch()	Crea un objeto Stopwatch (iniciándolo en start)
	Watch.elapsedTime()	Obtiene el tiempo (en segundos) desde que el cronometro fue creado



Midiendo el tiempo de ejecución

- A continuación se muestra como realizar la medición del tiempo empleando un objeto de tipo Stopwatch.
 - El "tiempo de inicio" se captura al crear el objeto.
 - El "tiempo final" al llamar a `elapsedTime()`.

```
def main():
    file = "8ints.txt"
    print(f"Leyendo datos del archivo: {file}")
    a = readInt1D(file)
    print(f"Arreglo de {len(a)} enteros leído.")
    print(f"Contando ternas que suman 0... ")
    watch = Stopwatch() # ---- Marca de tiempo inicial
    count = countTriples(a)
    elapsed_time = watch.elapsedTime() # ---- Marca de tiempo final
    print(count)
    print(f"Tiempo total {elapsed_time} segundos")
    if count < 10:
        writeTriples(a)

if __name__ == '__main__':
    main()
```



Midiendo el tiempo

- Dados **N** números enteros distintos, ¿cuántas ternas suman exactamente cero?

```
Leyendo datos del archivo: 1Kints.txt
Arreglo de 1000 enteros leído.
Contando ternas que suman 0...
70
Tiempo transcurrido: 26.97974850 segundos
```



```


Leyendo datos del archivo: 2Kints.txt
Arreglo de 2000 enteros leído.
Contando ternas que suman 0...
528
Tiempo transcurrido: 230.80460430
segundos

```



Análisis empírico

Ejecución de los experimentos

- 
1. Comenzar con un tamaño de entrada N moderado.
 2. Medir y registrar el tiempo de ejecución ($T(N)$).
 3. Duplicar el tamaño de la entrada N .
 4. Repetir.
 5. Tabular y graficar los resultados.



Análisis empírico

- Se mide el tiempo ejecutando el programa para varios tamaños de entrada y midiendo el tiempo de ejecución.

```
def uniformInt(lo, hi):
    """
    Return an integer chosen uniformly
    from the range [lo, hi).
    """
    return random.randrange(lo, hi)

def countTriples(a):
    n = len(a)
    count = 0
    for i in range(n):
        for j in range(i+1, n):
            for k in range(j+1, n):
                if (a[i] + a[j] + a[k]) == 0:
                    count += 1
    return count

def timeTrial(n):
    a = create1D(n, 0)
    for i in range(n):
        a[i] = uniformInt(-1000000, 1000000)
    watch = Stopwatch()
    count = countTriples(a)
    return watch.elapsedTime()
```

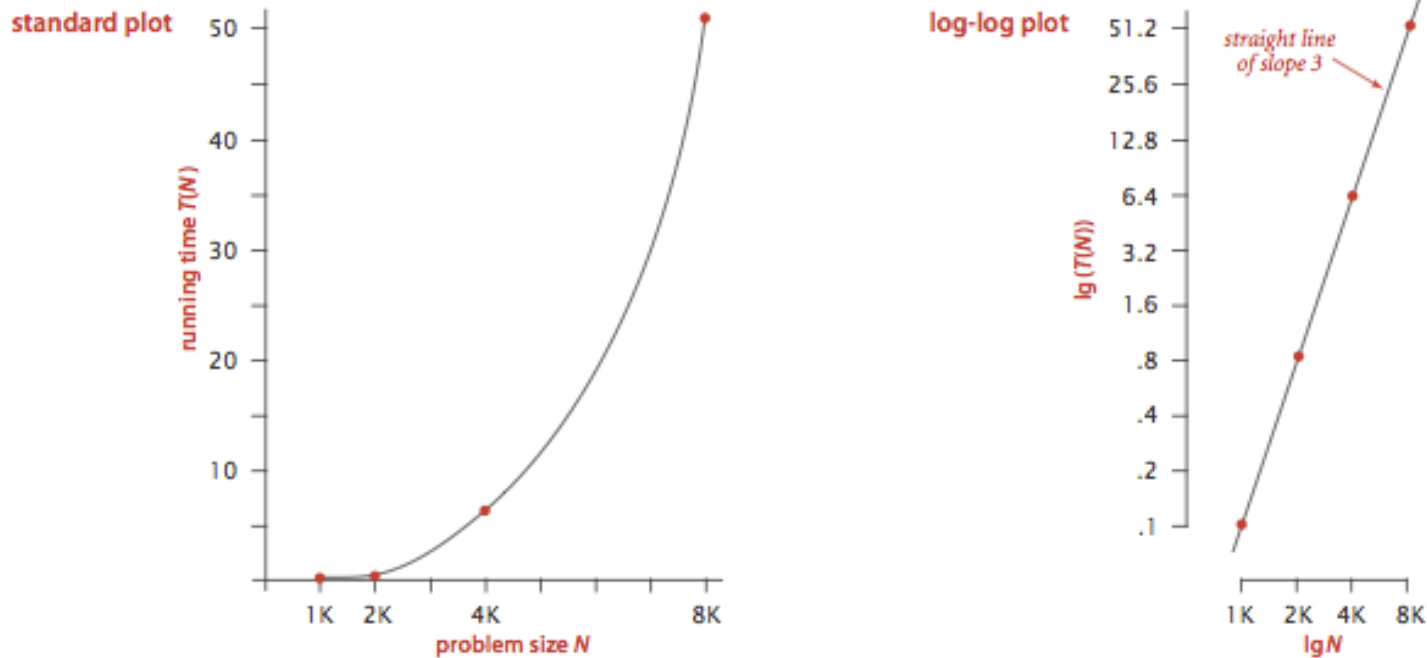
```
n = int(input("Ingrese el tamaño inicial n: "))
while True:
    previous = timeTrial(n // 2)
    current = timeTrial(n)
    ratio = current / previous
    print(f"{n:7d} {current:4.2f}/{previous:4.2f} = {ratio:4.2f}")
    n *= 2
```

```
Ingrese el tamaño inicial n: 250
250 0.42/0.05 = 8.28
500 3.58/0.38 = 9.46
1000 28.45/3.62 = 7.86
2000 229.12/27.95 = 8.20
4000 1932.38/228.12 = 8.47
CTRL + C
```



Análisis empírico

Anatomía de la clase



Analysis of experimental data (the running time of `ThreeSum.count()`)



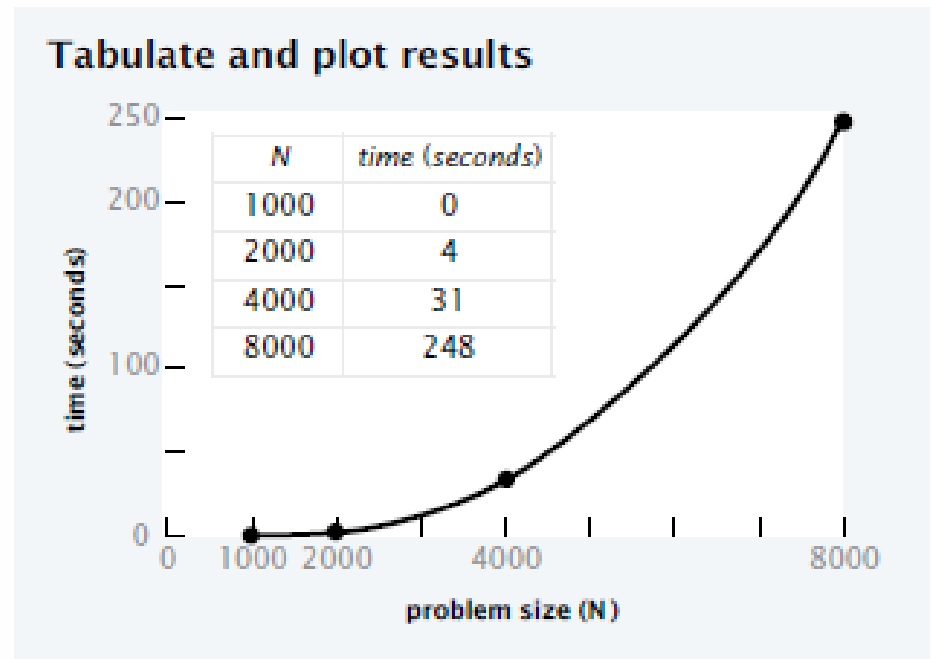
Análisis empírico

Anatomía de la clase

Tabla de resultados. Tiempo de ejecución $T(N)$ vs. tamaño de la entrada N .

N	$time (seconds)$
1000	0
2000	4
4000	31
8000	248

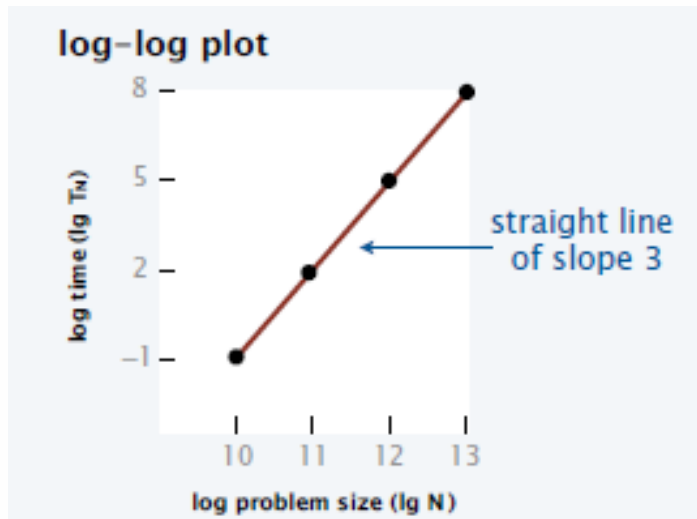
Grafica estandar. Tiempo de ejecución $T(N)$ vs. tamaño de la entrada N .



Análisis empírico

Ajuste de curvas (Curve fitting)

- Graficar en escala log – log .
- Si los puntos están en una línea recta (lo que es a menudo el caso), aplica una ley de potencia— se ajusta una curva de la forma aN^b .
- El exponente b es la pendiente de la línea.
- Resuelva para a con los datos.



Grafica escala logarítmica. $\log(T(N))$ vs. $\log(N)$

N	T_N	$\lg N$	$\lg T_N$	$4.84 \times 10^{-10} \times N^3$
1000	0.5	10	-1	0.5
2000	4	11	2	4
4000	31	12	5	31
8000	248	13	8	248

Expresión matemática

$$T_N = aN^b$$

Con $b = 3$ (Grafica)

$$\log(T_N) = \log(aN^3)$$

$$T_N = aN^3$$

$$\log(T_N) = \log(a) + \log(N^3)$$

$$\log(T_N) = \log(a) + 3\log(N)$$

Con $N = 8000$, $T_N = 248$

$$248 = a \times 8000^3$$

$$a = 4.84 \times 10^{-10}$$

$$T_N = 4.84 \times 10^{-10} \times N^3$$



Predicción y verificación

Tenemos hasta el momento:

- **Hipótesis:** El tiempo de ejecución de ThreeSum esta dado por la expresión:

$$T_N = 4.84 \times 10^{-10} \times N^3$$

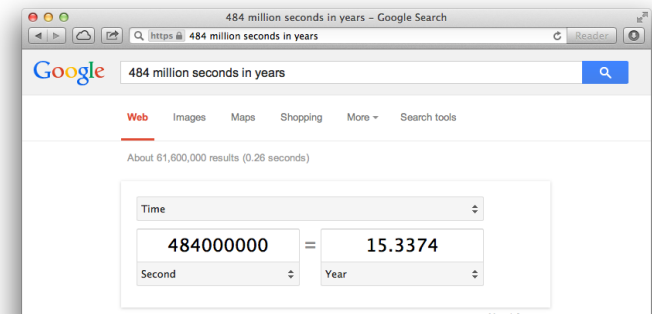
- **Predicción:** Para $N = 16000$ el resultado será de $T_N = 1982 \text{ seg} \approx 33.22 \text{ min}$.

```
% java Generator 1000000 16000 | java ThreeSum  
31903 (1985 seconds)
```



Se comprueba la predicción
con el resultado real.

- **Q:** ¿Qué tanto tiempo le tomaría al programa realizar la operación si $N = 1000000$?
- **A:** $N = 1000000 \rightarrow T_N = 484000000$. Es decir 484 millones de segundos o mas de 15 años.



Predicción y verificación

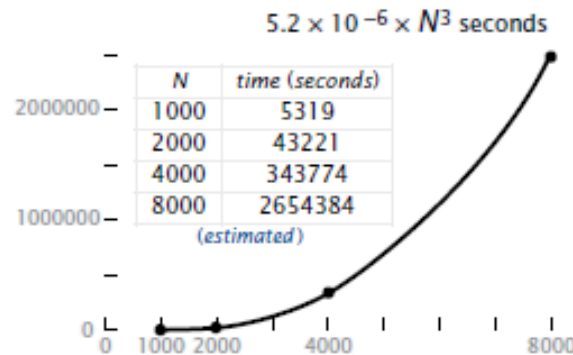
Tenemos hasta el momento:

- **Pregunta:** ¿Qué pasa si ejecutamos el mismo programa en otra maquina?

1970s



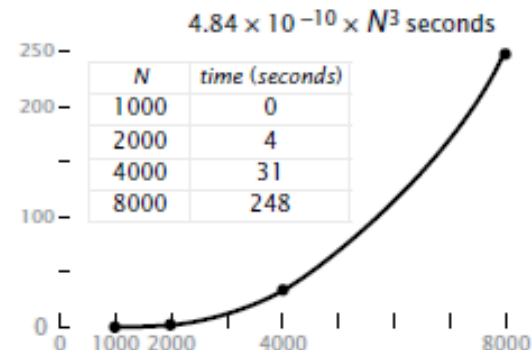
VAX 11/780



2010s: 10,000+ times faster



Macbook Air

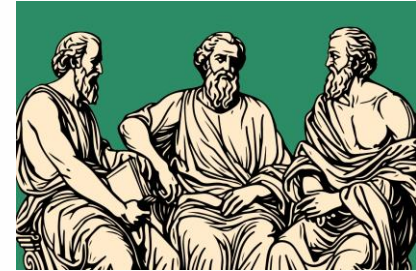


- **Otra hipótesis:** El tiempo de ejecución en diferentes maquina difieren solo por un factor constante.

Modelos matemáticos para el tiempo de ejecución

- **Q:** ¿Podemos escribir una fórmula precisa para el tiempo de ejecución de un programa de computadora?

A: (Sabiduría popular, década de 1960) **No**, es demasiado complicado.



A: (D. E. Knuth, 1968–presente) **¡Sí!**

- Determinar el conjunto de operaciones.
- Hallar el **costo** de cada operación (depende del computador y del software de sistema).
- Hallar la **frecuencia de ejecución** de cada operación (depende del algoritmo y de las entradas).
- **Tiempo de ejecución total:**



$$\text{Tiempo de ejecución total} = \sum_{op} c_{op} \times f_{op}$$



Modelos matemáticos para el tiempo de ejecución

Costo de las operaciones básicas

- **Reto:** ¿Cómo estimar las constantes?
- **Observación:** Las operaciones mas primitivas toman un tiempo constante.

operation	example	nanoseconds [†]	operation	example	nanoseconds [†]
integer add	<code>a + b</code>	2.1	variable declaration	<code>int a</code>	c_1
integer multiply	<code>a * b</code>	2.4	assignment statement	<code>a = b</code>	c_2
integer divide	<code>a / b</code>	5.4	integer compare	<code>a < b</code>	c_3
floating-point add	<code>a + b</code>	4.6	array element access	<code>a[i]</code>	c_4
floating-point multiply	<code>a * b</code>	4.2	array length	<code>a.length</code>	c_5
floating-point divide	<code>a / b</code>	13.5	1D array allocation	<code>new int[N]</code>	$c_6 N$
sine	<code>Math.sin(theta)</code>	91.3	2D array allocation	<code>new int[N][N]</code>	$c_7 N^2$
arctangent	<code>Math.atan2(y, x)</code>	129.0			
...			

[†] Running OS X on Macbook Pro 2.2GHz with 2GB RAM

- **Advertencia.** Las operaciones no primitivas suelen tomar más que tiempo constante.



Modelos matemáticos para el tiempo de ejecución

Calentamiento: 1-sum

```
public static int count(int[] a)
{
    int N = a.length;
    int cnt = 0;
    for (int i = 0; i < N; i++)
        if (a[i] == 0)
            cnt++;
    return cnt;
}
```

Note that frequency of increments depends on input.

operation	cost	frequency
function call/return	20 ns	1
variable declaration	2 ns	2
assignment	1 ns	2
less than compare	1/2 ns	$N + 1$
equal to compare	1/2 ns	N
array access	1/2 ns	N
increment	1/2 ns	between N and $2N$

representative estimates (with some poetic license); knowing exact values may require study and experimentation.

- **Q:** ¿Formula para el tiempo total de ejecución?
- **A:** $cN + 26.5 \text{ nanoseg}$, donde c esta entre 2 y 2.5, dependiendo de la entrada



Modelos matemáticos para el tiempo de ejecución

Calentamiento: 2-sum

```
public static int count(int[] a)
{
    int N = a.length;
    int cnt = 0;
    for (int i = 0; i < N; i++)
        for (int j = i+1; j < N; j++)
            if (a[i] + a[j] == 0)
                cnt++;
    return cnt;
}
```

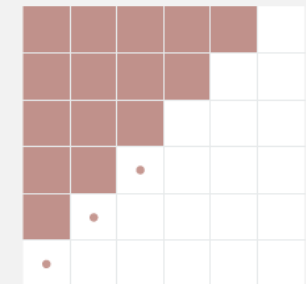
operation	cost	frequency
function call/return	20 ns	1
variable declaration	2 ns	$N + 2$
assignment	1 ns	$N + 2$
less than compare	1/2 ns	$(N + 1)(N + 2)/2$
equal to compare	1/2 ns	$N(N - 1)/2$
array access	1/2 ns	$N(N - 1)$
increment	1/2 ns	between $N(N + 1)/2$ and N^2

exact counts tedious to derive

$$\# i < j = \binom{N}{2} = \frac{N(N - 1)}{2}$$

- **Q:** ¿Formula para el tiempo total de ejecución?
- **A:** $c_1 N^2 + c_2 N + c_3 \text{ nanoseg}$, donde... (la expresión se vuelve mas complicada)

Pf. [n even]



$$0 + 1 + 2 + \dots + (N - 1) = \frac{1}{2}N^2 - \frac{1}{2}N$$

half of square half of diagonal



Modelos matemáticos para el tiempo de ejecución

Simplificando los cálculos

Notación asintótica (\sim)

- Usar solo el término de crecimiento más rápido.
- Ignorar los términos de crecimiento más lento.

Justificación

- Cuando N es grande, los términos ignorados son insignificantes.
- Cuando N es pequeño, todo es insignificante.

Def: $f(N) \sim g(N)$ significa que $f(N)/g(N) \rightarrow 1$ si $N \rightarrow \infty$

Ex. $5/4 N^2 + 13/4 N + 53/2 \sim 5/4 N^2$

\uparrow
1,253,276.5
for $N = 1,000$

\nwarrow 1,250,000
for $N = 1,000$,
within .3%

- **Q:** ¿Fórmula para el tiempo de ejecución de 2-suma cuando el conteo no es grande (caso típico)?
- **A:** $\sim 5/4 N^2$ nanoseg



Modelos matemáticos para el tiempo de ejecución

Modelo matemático para 3-sum

```
public static int count(int[] a)
{
    int N = a.length;
    int cnt = 0;
    for (int i = 0; i < N; i++)
        for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N; k++)
                if (a[i] + a[j] + a[k] == 0)
                    cnt++;
    return cnt;
}
```

operation	cost	frequency
function call/return	20 ns	1
variable declaration	2 ns	$\sim N$
assignment	1 ns	$\sim N$
less than compare	1/2 ns	$\sim N^3/6$
equal to compare	1/2 ns	$\sim N^3/6$
array access	1/2 ns	$\sim N^3/2$
increment	1/2 ns	$\sim N^3/6$

$$\# i < j < k = \binom{N}{3} = \frac{N(N-1)(N-2)}{6} \sim \frac{N^3}{6}$$

← assumes count is not large

- **Q:** ¿Fórmula para el tiempo total de ejecución cuando el valor de retorno no es grande (caso típico)?
- **A:** $\sim N^3/2$ nanoseg

✓ ← matches $4.84 \times 10^{-10} \times N^3$ empirical hypothesis



Truco para contar operaciones

Problema: Contar operaciones es difícil

- El análisis de algoritmos (como lo hacía Knuth) requiere contar la frecuencia de las operaciones. Este conteo casi siempre resulta en sumas discretas (sumatoria, Σ)
 - Un bucle for es una Σ
 - Un bucle triple anidado es una $\Sigma \Sigma \Sigma$
- Calcular el valor exacto de estas sumatorias es complicado (Es necesario tomar un curso de Matemáticas discretas).
- Para facilitar el calculo, **use Integrales como Aproximación**



Truco para contar operaciones

Truco: Usar Integrales como Aproximación

- Para valores de N muy grandes, el comportamiento de una suma discreta \sum es casi idéntico al de su contraparte continua integral (\int)
- Truco: $\sum(\text{Suma discreta}) \approx \int(\text{Integral continua})$
- Lo más importante es que el orden de crecimiento (el término N^2 , N^3 , $\log(N)$, etc.) del resultado de la integral es el mismo que el de la suma.

Ex 1. $1 + 2 + \dots + N$.

$$\sum_{i=1}^N i \sim \int_{x=1}^N x dx \sim \frac{1}{2} N^2$$

Ex 2. $1^k + 2^k + \dots + N^k$.

$$\sum_{i=1}^N i^k \sim \int_{x=1}^N x^k dx \sim \frac{1}{k+1} N^{k+1}$$

Ex 3. $1 + 1/2 + 1/3 + \dots + 1/N$.

$$\sum_{i=1}^N \frac{1}{i} \sim \int_{x=1}^N \frac{1}{x} dx = \ln N$$

Ex 4. 3-sum triple loop.

$$\sum_{i=1}^N \sum_{j=i}^N \sum_{k=j}^N 1 \sim \int_{x=1}^N \int_{y=x}^N \int_{z=y}^N dz dy dx \sim \frac{1}{6} N^3$$



Truco para contar operaciones

Truco: Usar Integrales como Aproximación

- **Ejemplo:** Cual es el valor de: $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots$

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots = \frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \dots = \sum_{i=0}^{\infty} \frac{1}{2^i}$$

$$\sum_{i=0}^{\infty} \frac{1}{2^i} = 2$$

Si se aplica el truco tenemos:

$$\sum_{i=0}^{\infty} \frac{1}{2^i} \rightarrow \int_{x=0}^{\infty} \left(\frac{1}{2}\right)^x dx = \frac{1}{\ln(2)} \approx 1.4427$$

- **Advertencia:** El truco de la integral no siempre trabaja

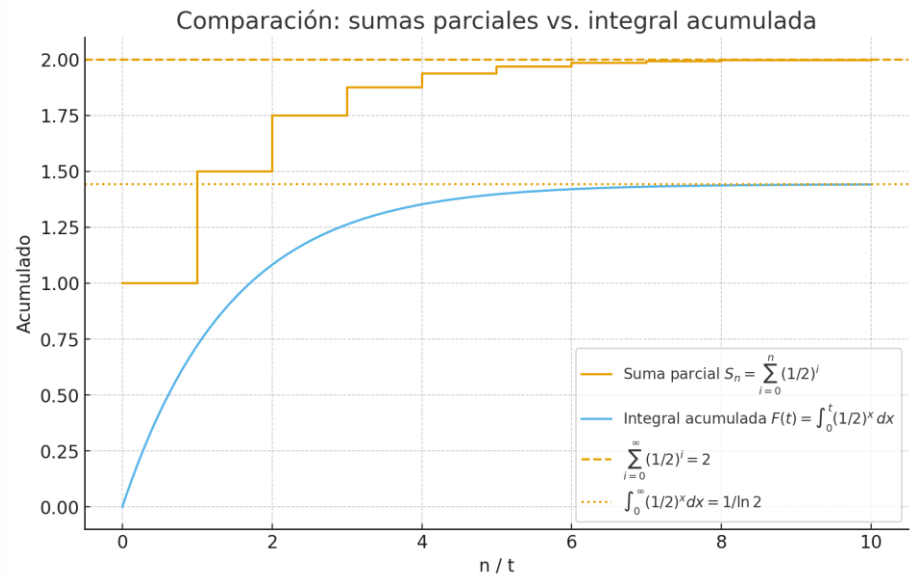
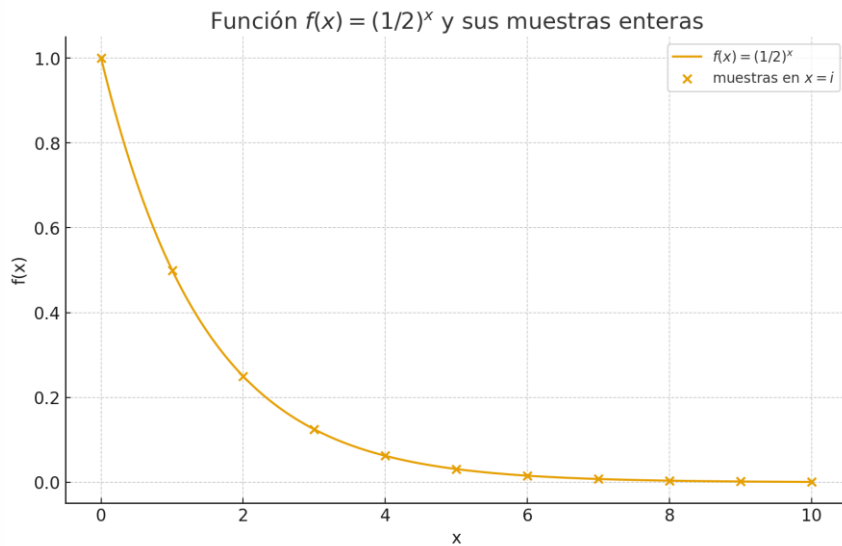


Truco para contar operaciones

Truco: Usar Integrales como Aproximación

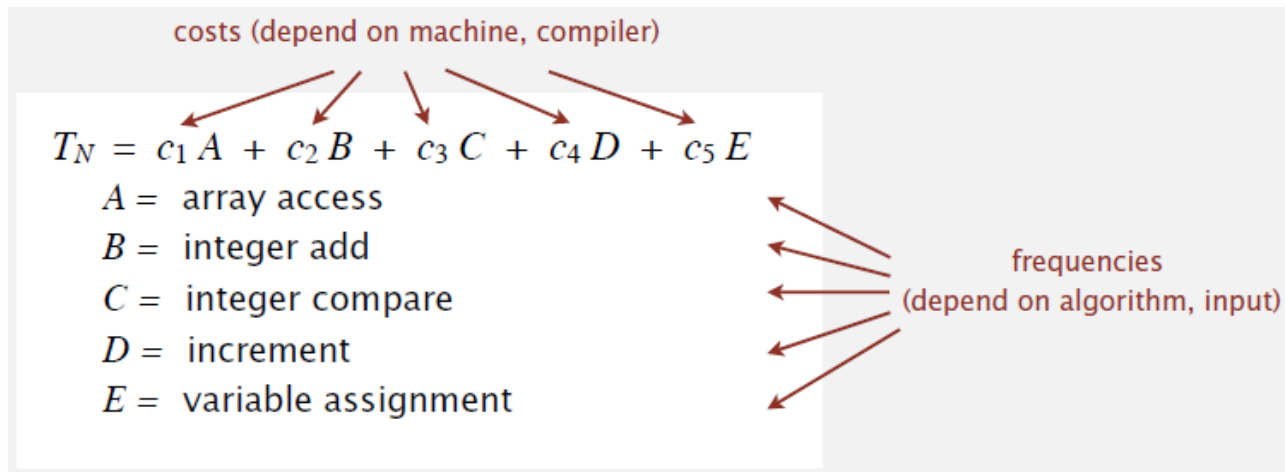
$$\sum_{i=0}^{\infty} \frac{1}{2^i} = 2$$

$$\int_{x=0}^{\infty} \left(\frac{1}{2}\right)^x dx = \frac{1}{\ln(2)} \approx 1.4427$$



Modelos matemáticos para el tiempo de ejecución

- **En principio**, existen modelos matemáticos para el tiempo de ejecución
- **En la practica**,
 - Las formulas pueden ser complicadas.
 - Podría requerirse matemáticas avanzadas.
 - Es mejor dejar los modelos exactos para los expertos.



- **Conclusión**: Usaremos **modelos aproximados**: $T(N) \sim cN^3$

Contextualizando todo

Método científico

- **Observar** alguna característica del mundo natural.
- **Hipotetizar** un modelo consistente con las observaciones.
- **Predecir** eventos usando la hipótesis.
- **Verificar** las predicciones realizando (o mediante) observaciones adicionales.
- **Validar** refinando hasta que la hipótesis y las observaciones concuerden.

Análisis empírico de programas

- La "característica del mundo natural" es el tiempo que toma un programa en un computador.
- Ajustar una curva a los datos experimentales para obtener una fórmula del tiempo de ejecución como una función de **N**.
- Útil para predecir, pero no para explicar.

Análisis matemático de Algoritmos

- Analizar el algoritmo para desarrollar una fórmula para el tiempo de ejecución como una función de **N**.
- Útil para predecir y explicar.
- Podría involucrar matemáticas avanzadas.
- Aplica a cualquier computador.

Los **modelos matemáticos** son mas fáciles de formular en ciencias de computación que en otras ciencias.



Orden de crecimiento

Definición

Si $f(N) \sim c \cdot g(N)$ para alguna constante $c > 0$, entonces el orden de crecimiento de $f(N)$ es $g(N)$.

- Ignora el coeficiente principal.
- Ignora los términos de orden inferior.

Ex. $5/4 N^2 + 13/4 N + 53/2 \sim 5/4 N^2$

1,253,276.5 for $N = 1,000$

1,250,000 for $N = 1,000$, within .3%

```
public static int count(int[] a)
{
    int N = a.length;
    int cnt = 0;
    for (int i = 0; i < N; i++)
        for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N; k++)
                if (a[i] + a[j] + a[k] == 0)
                    cnt++;
    return cnt;
}
```

operation	cost	frequency
function call/return	20 ns	1
variable declaration	2 ns	$\sim N$
assignment	1 ns	$\sim N$
less than compare	1/2 ns	$\sim N^3/6$
equal to compare	1/2 ns	$\sim N^3/6$
array access	1/2 ns	$\sim N^3/2$
increment	1/2 ns	$\sim N^3/6$

$\# i < j < k = \binom{N}{3} = \frac{N(N-1)(N-2)}{6} \sim \frac{N^3}{6}$

assumes count is not large

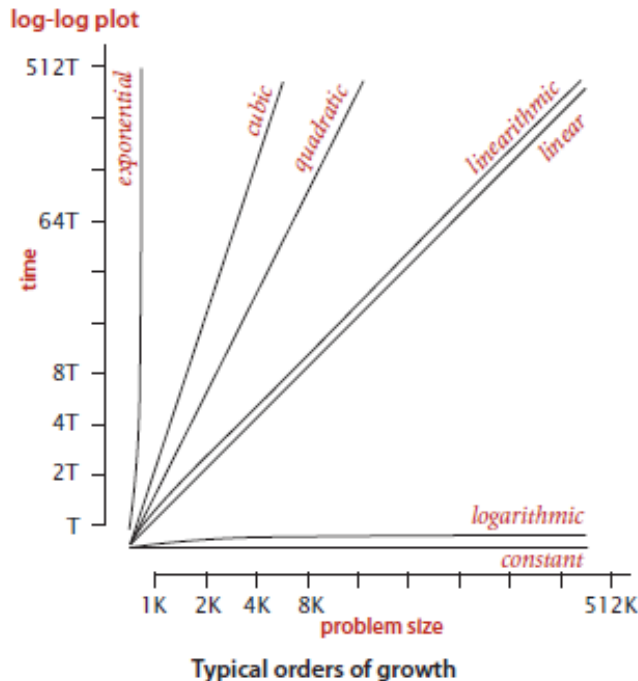
El orden de crecimiento del **tiempo de ejecución** del código analizado es N^3



Tipos de orden de crecimiento

Buena noticia

El conjunto de funciones 1 , $\log N$, N , $N \log N$, N^2 , N^3 y 2^N es suficiente para describir el orden de crecimiento de la mayoría de los algoritmos comunes.



Linear (N)

```
for (int i = 0; i < N; i++)  
...
```

Quadratic (N^2)

```
for (int i = 0; i < N; i++)  
  for (int j = i+1; j < N; j++)  
    ...
```

Cubic (N^3)

```
for (int i = 0; i < N; i++)  
  for (int j = i+1; j < N; j++)  
    for (int k = j+1; k < N; k++)  
      ...
```

Logarithmic ($\log N$)

```
public static void f(int N)  
{  
  if (N == 0) return;  
  ... f(N/2) ...  
}
```

Linearithmic ($N \log N$)

```
public static void f(int N)  
{  
  if (N == 0) return;  
  ... f(N/2) ...  
  ... f(N/2) ...  
  for (int i = 0; i < N; i++)  
    ...  
}
```

Exponential (2^N)

```
public static void f(int N)  
{  
  if (N == 0) return;  
  ... f(N-1) ...  
  ... f(N-1) ...  
}
```

↑
ignore for practical purposes
(infeasible for large N)



Clasificaciones comunes de ordenes de crecimiento

order of growth	name	typical code framework	description	example	$T(2N) / T(N)$
1	constant	<code>a = b + c;</code>	statement	add two numbers	1
$\log N$	logarithmic	<pre>while (N > 1) { N = N / 2; ... }</pre>	divide in half	binary search	~ 1
N	linear	<pre>for (int i = 0; i < N; i++) { ... }</pre>	loop	find the maximum	2
$N \log N$	linearithmic	[see mergesort lecture]	divide and conquer	mergesort	~ 2
N^2	quadratic	<pre>for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }</pre>	double loop	check all pairs	4
N^3	cubic	<pre>for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }</pre>	triple loop	check all triples	8
2^N	exponential	[see combinatorial search lecture]	exhaustive search	check all subsets	$T(N)$



Referencias

Referencias

- Este material ha sido adaptado de las presentaciones disponibles en:
 - <https://introcs.cs.princeton.edu/python/>
 - <https://algs4.cs.princeton.edu/14analysis/>





POLITÉCNICO COLOMBIANO
JAIME ISAZA CADAVID
INSTITUCIÓN UNIVERSITARIA

Educación para
vivir mejor

Gracias



Politécnico Colombiano Jaime Isaza Cadavid



@PolitecnicoJIC



GOBERNACIÓN DE ANTIOQUIA



www.politecnicojic.edu.co / Medellín - Apartadó - Rionegro

VIGILADA MINEDUCACIÓN