

# Análisis de Algoritmos II

# Análisis aproximado

Para determinar aproximadamente el tiempo de ejecución se:

- Identifican los distintos bloques, que requieren un tiempo de ejecución constante
- Se determina la frecuencia de ejecución

Se multiplican los tiempos de cada bloque por su frecuencia y se suman todos los resultados para el estimado final.

# Ejemplos de análisis

Encontrar el máximo de un vector

```
double max(double[] a) {  
    double x = a[0];  
    for(int i=1; i<a.length; i++)  
        if (a[i]>x) x=a[i];  
    return x;  
}
```

$\sim N$

Encontrar parejas que sumen 0

```
int paresCero(double[] a) {  
    int conteo=0;  
    for(int i=0; i<a.length; i++)  
        for(int j=0; j<a.length; j++)  
            if (i!=j && a[i]+a[j]==0)  
                conteo++;  
    return conteo;  
}
```

$\sim N^2$

Obtener todos los subconjuntos de un vector  $\sim 2^N$

# Ejemplo

- Análisis aproximado de ThreeSum
- Observamos que generalmente el orden de crecimiento depende de las instrucciones del ciclo más interno, o alternativamente, la operación elemental de mayor frecuencia.
- El orden de crecimiento obtenido por el análisis aproximado es *independiente de la implementación*.

# Modelo de costo

- Partiendo de la observación de que el tiempo aproximado depende del ciclo más interno, buscamos identificar cuales son las operaciones básicas dentro de este ciclo.

Ejemplo: En ThreeSum

- Modelo de costo accesos al arreglo: 3 por iteración
- Frecuencia del condicional:  $N^3/6$
- Número total de accesos:  $3 \cdot (N^3/6) = N^3/2$

# Ejemplo: Modelos de costo

Evaluar un polinomio, método por definición:

```
public static double evalPoly(double[] a, double x)
{
    double s = 0;
    for(int i=0; i<a.length; i++)
        s += a[i]*Math.pow(x,i);
    return s;
}
```

Modelos de costo:

- Accesos a memoria,  $a[i]$  :  $d+1$
- Operaciones suma (double) :  $d+1$
- Operaciones producto (double) :  $d+1+d(d-1)/2$

# Ejemplo: Modelos de costo

Evaluar un polinomio, método por **regla de Horner**:

```
public static double horner(double[] a, double x)
{
    double s = a[a.length-1];
    for(int i=a.length-2; i>=0; i--)
        s = s*x + a[i];
    return s;
}
```

Modelos de costo:

- Accesos a memoria,  $a[i]$  :  $d+1$
- Operaciones suma (double) :  $d$
- Operaciones producto (double) :  $d$

# Metodología general de análisis

1. Se identifica el modelo de la entrada (Cuál es el tamaño  $N$  de la entrada).
2. Se define el modelo de costo para el ciclo más interno.
3. Se calcula la frecuencia de ejecución de las operaciones asociadas al modelo de costo en el ciclo más interno.



# Notación Big-O

- Es una notación ampliamente utilizada para describir el comportamiento *asintótico de funciones*, es decir para valores grandes del tamaño de la entrada.
- Decimos que  $T(N)$  esta en el orden  $O( f(N) )$  si:

*Para  $N$  suficientemente grandes,  
existe una constante  $c > 0$  tal que*

$$T(N) \leq c * f(N)$$

- En otras palabras  $T(N)$  está acotada superiormente por un factor constante de  $f(N)$  para todo  $N$  grande.

# Experimentos de doblado de la entrada

- Aplicable en casos en los que los algoritmos crecen como funciones de potencia

$$N^b$$

1. Se mide experimentalmente duplicando el tamaño de la entrada en cada caso.
2. Si el cociente entre tiempos sucesivos es  $2^b$ , entonces el algoritmo tiene un orden de crecimiento  $N^b$ .

# Ejemplo

- Sea un algoritmo con  $T(N) = a N^b (\lg N)^c$ .
- La relación  $T(2N)/T(N)$  sería:

$$\begin{aligned}\frac{T(2N)}{T(N)} &= \frac{a(2N)^b(\lg 2N)^c}{aN^b(\lg N)^c} \\ &= 2^b \left( \frac{\lg 2 + \lg n}{\lg N} \right)^c \\ &= 2^b \left( 1 + \frac{\lg 2}{\lg N} \right)^c \\ &\approx 2^b\end{aligned}$$

# Ejemplo:

## Prueba de doblado con ThreeSum

| N     | T(N)  | T(2N) / T(N) |
|-------|-------|--------------|
| 500   | 0.1   |              |
| 1000  | 0.1   | 1.0          |
| 2000  | 0.6   | 6.0          |
| 4000  | 4.7   | 7.8          |
| 8000  | 36.3  | 7.7          |
| 16000 | 287.0 | 7.9          |

Observamos que el cociente  $T(2N)/T(N)$  tiende a  $7.9=2^{2.98}$   
Bastante cercano respecto al exponente esperado teóricamente (3)

# Estimando tiempos para entradas mayores

Una vez los tiempos convergen, se puede estimar el tiempo para entradas mayores:

- Se duplica  $N$  y se multiplica por  $2^b$  el tiempo

# Limitantes de la aproximación de doblado de entrada

- Constantes grandes
- Loop interior no dominante
- Tiempo de instrucción no constante
- Consideraciones del sistema (e.g. Mem. virtual)
- Dependencia en los valores de la entrada
- Múltiples parámetros

# Manejando la dependencia en las entradas

- Modelos de la entrada imprecisos: Hacer análisis más detallados
- Estimar tiempos de peor caso: Esto provee una garantía de como se comportará el algoritmo
- Utilizar algoritmos randomizados, por ejemplo randomizar los elementos del vector de entrada
- Realizar análisis amortizado

# Análisis de peor caso

- Ejemplo: Las operaciones de las estructuras Bag, Stack, Queue implementadas con una lista enlazada requieren tiempo constante en el peor caso.
- Ejemplo: El algoritmo ThreeSum requiere un tiempo total en el orden de  $N^3$  en el peor caso.



# Ejemplo: Búsqueda secuencial

- Dado un objeto x, determinar en que posición de un arreglo se encuentra.

```
int buscar(T[] datos, T item) {  
    for(int i=0; i<datos.length; i++)  
        if (item.equals(datos[i]))  
            return i;  
    return -1;  
}
```

Modelo de costo:  
La comparación

- Peor caso: No se encuentra, hace N comparaciones.
- Mejor caso: Aparece en la 1ª posición, hace 1 comparación.
- Caso medio: Hace  $(1+\dots+N)/N$  comparaciones.

# Análisis amortizado

- Si una operación costosa se ejecuta con poca frecuencia, se puede *amortizar* su costo sobre el tiempo total requerido.
- Ejemplo: La pila implementada con arreglo y con la operación `resize()`. Se hace una secuencia de  $N$  operaciones `push`. Suponer que el arreglo inicial es de tamaño 4. El número total de accesos al arreglo sería

$$N + 4 + 8 + \dots + 2N = 5N - 4$$

# Análisis de memoria

- Es muy dependiente de la arquitectura del hardware y del lenguaje de programación.
- Se asumen los siguientes modelos (Java, arquitectura de 64bits)

| Tipo primitivo | Memoria (bytes) |
|----------------|-----------------|
| boolean        | 1               |
| byte           | 1               |
| char           | 2               |
| int            | 4               |
| long           | 8               |
| float          | 4               |
| double         | 8               |
| referencia     | 8               |

# Representación de objetos

- Un objeto requiere 12 bytes de overhead (referencia al objeto class, información para el garbage collector, datos de sincronización)
- Luego del objeto se guardan las variables de instancia
- Finalmente se agrega un padding para que el total sea múltiplo de 8 bytes ( = 64bits)

# Ejemplos

Integer

|          |    |
|----------|----|
| overhead | 12 |
| int x    | 4  |

tamaño = 16

Date

|          |    |
|----------|----|
| overhead | 12 |
| int año  | 4  |
| int mes  | 4  |
| int día  | 4  |

tamaño = 24

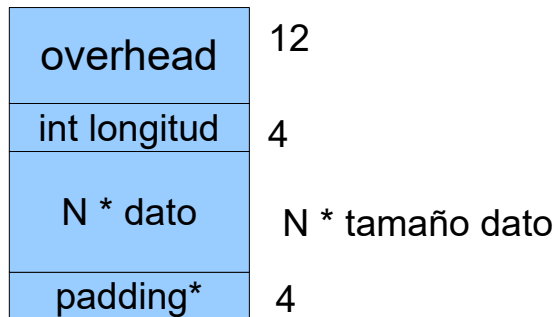
Nodo

|                |    |
|----------------|----|
| overhead       | 12 |
| extra overhead | 8  |
| item           | 8  |
| next           | 8  |
| padding        | 4  |

tamaño = 40

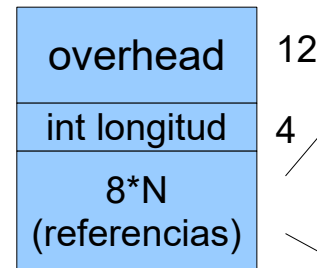
# Arreglos

Arreglo  
(tipo primitivo)



$$\text{tamaño} = 16 + N(\text{tamaño tipo})$$

Arreglo  
(tipo objeto)



Instancias

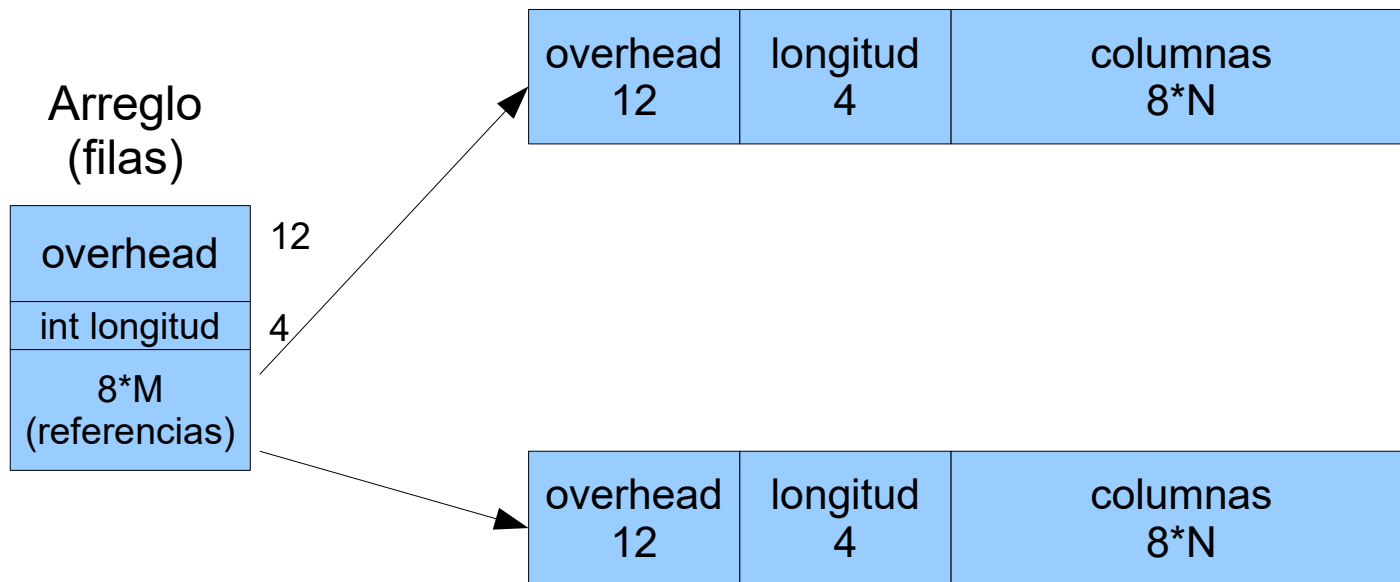


$$\text{tamaño} = 16 + 8*N + N(\text{tamaño instancia})$$

# Arreglos bidimensionales

Ejemplo:

```
double[][] matriz = new double[M][N];
```



$$\text{tamaño total} = 16 + 8M + M(16+8N)$$

# Strings (Java 7 y superior)

