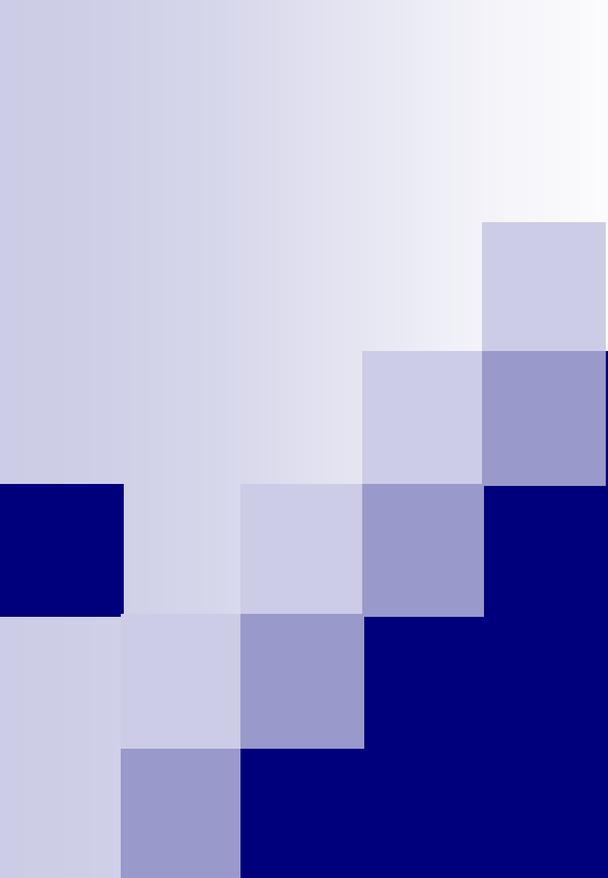


# Tema 1: Introducción



# 1.1 Introducción al Análisis de Algoritmos

# Tiempo Abstracto de Ejecución (TAE)

- ¿Qué analizar en un algoritmo?
  - Corrección.
  - Uso de memoria.
  - **Rendimiento (tiempo de ejecución).**
- ¿Cómo medir el rendimiento?
  - Mediante un reloj (tiempo puro de ejecución)
    - *Intuitivo pero problemático ...*
  - **Analizando el pseudocódigo: tiempo abstracto de ejecución (tae)**

# ¿Cómo medir el tae?

- Opción 1: Contar el número de sentencias básicas (líneas acabadas en ; que no son llamadas a función) que ejecuta el algoritmo dada una entrada I.
  - Opción compleja
  - Depende de cómo se escriba el pseudocódigo
  - Aporta información innecesaria.
- Esto es, damos un tae de 1 a las sentencias básicas
- Pero vamos a ver un ejemplo

# Ejemplo de Cálculo del TAE

- Ejemplo: multiplicar matrices cuadradas (pseudocódigo)

```
matriz MM(matriz A, matriz B, dim N)
```

```
  para i de 1 a N:
```

```
    para j de 1 a N:
```

```
      c[i, j] = 0.;
```

```
      para k de 1 a N:
```

```
        c[i, j] += a[i, k] * b[k, j];
```

```
  devolver C
```

# Cálculo del TAE de MM

$$tae_{MM}(A, B, N) = \sum_{i=1}^N tae(ite(i)) =$$

$$\sum_{i=1}^N \sum_{j=1}^N tae(ite(j)) =$$

$$\sum_{i=1}^N \sum_{j=1}^N (1 + \sum_{k=1}^N tae(ite(k)))$$

$$\leq \sum_{i=1}^N \sum_{j=1}^N (1 + N)$$

$$= N^2 (1 + N)$$

matriz MM(matriz A, matriz B, dim N)

para i de 1 a N:

para j de 1 a N:

c[i, j] = 0.;

para k de 1 a N:

c[i, j] += a[i, k] \* b[k, j];

devolver C

Hemos llegado a un tae de la forma  $f(N) = f(\text{tamaño})$

# Cálculo del TAE: SelectSort

```
void SelectSort(Tabla T, ind P, ind U)
    i=P;
    mientras i<U:
        min=i ;
        para j de i+1 a U :
            si T[j]<T[min] :
                min=j ;
        swap(T[i],T[min]) ;    i++;
```

## □ Más corto:

```
void SelectSort(Tabla T, ind P, ind U)
    para i de P a U-1:
        min= min(T, i, U);
        swap(T[i],T[min]);
```

# Funcionamiento de SelectSort

- Ejemplo. SelectSort
- Entrada:  $T=[4,3,2,1]$ ,  $P=1$ ,  $U=4$ .

Iteración	Operaciones	Tabla
$i=1$	$\text{min}=4$ $\text{swap}(T[1],T[4])$	$[1,3,2,4]$
$i=2$	$\text{min}=3$ $\text{swap}(T[2],T[3])$	$[1,2,3,4]$
$i=3$	$\text{min}=3$ $\text{swap}(T[3],T[3])$	$[1,2,3,4]$

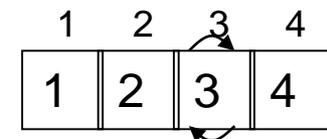
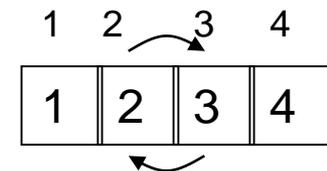
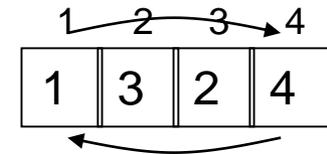
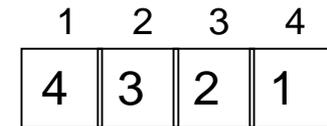
# Funcionamiento de SelectSort II

□ En más detalle:

Entrada:  $T=[4,3,2,1]$ ,  $P=1$ ,  $U=4$ .

Bucle 1

- $i=1$ 
  - Bucle 2
    - $\min = 1$
    - $j=2$  ¿ $T[2]<T[1]$ ? Si  $\rightarrow \min=2$
    - $j=3$  ¿ $T[3]<T[2]$ ? Si  $\rightarrow \min=3$
    - $j=4$  ¿ $T[4]<T[3]$ ? Si  $\rightarrow \min=4$
    - $\min=4$ ,  $\text{swap}(T[1],T[4])$
- $i=2$ 
  - Bucle 2
    - $\min = 2$
    - $j=3$  ¿ $T[3]<T[2]$ ? Si  $\rightarrow \min=3$
    - $j=4$  ¿ $T[4]<T[3]$ ? No  $\rightarrow \min=3$
    - $\min=3$ ,  $\text{swap}(T[2],T[3])$
- $i=3$ 
  - Bucle 2
    - $\min = 3$
    - ¿ $T[4]<T[3]$ ? No  $\rightarrow \min=3$
    - $\min=3$ ,  $\text{swap}(T[3],T[3])$
- $i=4$  Fin ( $i=U$ )



# TAE de SelectSort

- SelectSort funciona con 2 bucles anidados

```
void SelectSort(Tabla T, ind P, ind U)
```

```
    i=P;
```

```
    mientras i<U:
```

```
        min=i;
```

```
        para j de i+1 a U:
```

```
            si T[j]<T[min] :
```

```
                min=j;
```

```
        swap(T[i],T[min]);
```

```
        i++;
```

Bucle 1

Bucle 2

***Vamos a intentar calcular su TAE***

# TAE de SelectSort II

## ■ Cálculo de $t_{ae}_{\text{Selectsort}}(T;P,U)$ :

```
void SelectSort(Tabla T, ind P, ind U)
```

```
    i=P;
```

```
    mientras i<U:
```

Bucle de P a U-1

```
        min=i;
```

```
        para j de i+1 a U:
```

Bucle de i+1 a U

```
            si T[j]<T[min]:
```

```
                min=j;
```

```
            swap(T[i],T[min]);
```

```
        i++;
```

bucle 1  
U-P  
iteraciones

bucle 2  
U-i  
iteraciones

***Parece que el número de bucles y su tamaño determina el tae***

# TAE de SelectSort III

Observando que  $N=U-P+1$  es el tamaño de  $T$ , se tiene

$$\begin{aligned}
 tae_{SS}(T, P, U) &= 1 + \sum_{i=P}^{U-1} tae(iter(i)) = 1 + \sum_{i=P}^{U-1} (4 + tae_{bucle\ 2}(T, i, U)) \\
 &\leq 1 + \sum_{i=P}^{U-1} (4 + 3(U - i)) = 1 + 4(U - P) + 3 \sum_{j=1}^{U-P} j \\
 &= 1 + 4(U - P) + 3(U - P + 1)(U - P) / 2 \\
 &= 1 + 4(N - 1) + 3(N - 1)N / 2 \\
 &= 3N^2 / 2 + 5N / 2 - 3
 \end{aligned}$$

$$N = U - P + 1$$

$$\sum_{i=1}^N i = (N + 1)N / 2$$

Hemos llegado a un tae de la forma  $f(N)$

Pero hay cosas sueltas, sobre todo en los coeficientes

Ejemplo: es el tae de swap 1? ó 3?

# ¿Cómo simplificar y normalizar el TAE?

- **Observación:** el TAE de MM y SS está dominado por el bucle más interno
- **Opción 2.**
  - Definir una **operación básica** y contar el número de veces que la ejecuta el algoritmo **A** sobre una entrada **I** ( $n_A(I)$ )
  - Tomar como tiempo abstracto de ejecución del algoritmo **A**, para la entrada **I** el número de **operaciones básicas** que ejecuta **A** sobre **I**:  $\mathbf{tae}_A(I) = n_A(I)$  .
- **Operación básica:**
  - Se ha de encontrar en el bucle más interno del algoritmo (es la operación que más veces se ejecuta).
  - Debe ser representativa del algoritmo (pero también de otros que resuelven el mismo problema).

# Vuelta al psc de MM

- Ejemplo: multiplicar matrices (pseudocódigo)

matriz MM(matriz A, matriz B, dim N)

para i de 1 a N:

para j de 1 a N:

$c[i, j] = 0.$ ;

para k de 1 a N:

$c[i, j] += a[i, k] * b[k, j];$

devolver C

- OB: \*

- $n_{MM}(A, B, N) = N^3$



# Vuelta al psc de SelectSort

```
void SelectSort(Tabla T, ind P, ind U)
    i=P;
    mientras i<U:
        min=i ;
        para j de i+1 a U :
            si T[j]<T[min] :
                min=j ;
        swap(T[i],T[min]) ;
    i++;
```

# Operación básica de SelectSort

- Operación básica
  - Debe estar en el bucle más interno
  - Ser “representativa” del algoritmo.
- Un buen candidato a ser OB de SelectSort es la operación **si  $T[j] < T[\min]$ ; .**
  - Esta operación se denomina “comparación de claves” (CDC).
  - Efectivamente, se encuentra en el bucle más interno.
  - Es representativa de SelectSort y de otros muchos algoritmos de ordenación

# Análisis del rendimiento de SelectSort

## ■ Cálculo de $n_{\text{SelectSort}}(T, P, U)$

```
void SelectSort(Tabla T, ind P, ind U)
```

```
    i=P;
```

```
    mientras i<U:
```

Bucle de P a U-1

```
        min=i;
```

```
        para j de i+1 a U:
```

Bucle de i+1 a U

```
            si T[j]<T[min]:
```

```
                min=j;
```

```
            swap(T[i],T[min]);
```

```
        i++;
```

bucle 1  
U-P  
iteraciones

bucle 2  
U-i  
iteraciones

$$\begin{aligned}
 n_{\text{SelectSort}}(T, 1, N) &= \sum_{i=1}^{N-1} n_{\text{bucle 2}}(T, i, N) = \sum_{i=1}^{N-1} \sum_{j=i+1}^N 1 = \sum_{i=1}^{N-1} (N-i) = \sum_{j=1}^{N-1} j \\
 &= \frac{N(N-1)}{2} = \frac{N^2}{2} - \frac{N}{2}
 \end{aligned}$$



# Resumiendo ...

- Hemos sido capaces de evaluar una primera versión del tae de SelectSort
- El resultado, básicamente  $\text{tae} = N^2$ , es razonable pues involucra el tamaño de tabla y el número de bucles, pero tiene elementos **ambiguos**
- La **OB** elimina esas ambigüedades
- El tae depende del **número de bucles** y de su **tamaño**
- El análisis es fácil cuando el tae del bucle **no varía** en cada iteración
- Si el tae del bucle depende de cómo se entre en él, introducimos **sumatorios** en el análisis
- Primera idea:  **$\text{tae} \sim \text{tamaño}^{\text{num. bucles}}$**  ?

# Ejemplo: Búsqueda Binaria

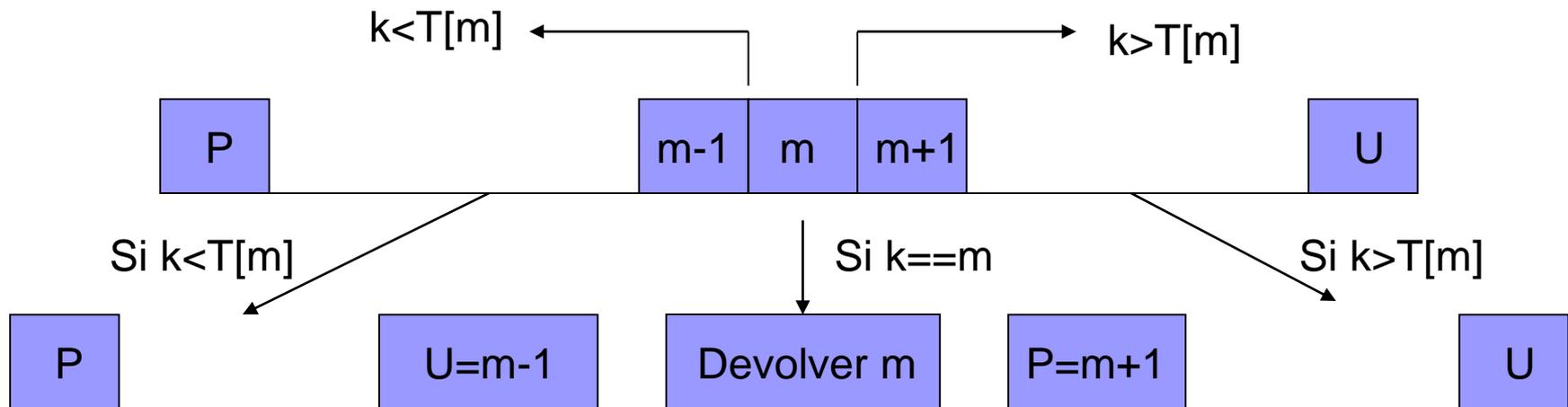
## □ Búsqueda Binaria

```
ind BBin(Tabla T, ind P, ind U, clave k)
  mientras  $P \leq U$  :
     $m = (P + U) / 2$ ;
    si  $T[m] == k$  :
      devolver m;
    else si  $k < T[m]$  :
       $U = m - 1$ ;
    else :
       $P = m + 1$ ;
  devolver error;
```

**Importante:** BBin sólo funciona si la tabla T está ordenada.

# ¿Cómo funciona BBin?

- En cada iteración o hay retorno o la tabla se reduce



- El tamaño de la tabla es aproximadamente la mitad tras cada iteración
- Si se sale del bucle  $P \leq U$  porque  $P > U$  significa que la clave buscada  $k$  no está en  $T$ .



# OB para BBin

- Mirando el único bucle, un buen candidato a ser OB es la operación de comparación:

```
si T[m]==k :
```

```
.....
```

```
else si k < T[m] :
```

```
.....
```

```
else :
```

```
.....
```

- Aunque hay dos sólo la contamos una vez
- Se trata de nuevo de una operación de comparación de clave

# Análisis de rendimiento de BBin

- $n_{\text{BBin}}(T,P,U,k)$ = número de iteraciones ( $P < U$ )  $\leq$  número de veces que se puede dividir un número  $N$  entre 2.
- Tras cada iteración el nuevo tamaño de la tabla es menor que la mitad del previo. Por tanto:

$$N \rightarrow \frac{N}{2} \rightarrow \frac{N}{2^2} \rightarrow \dots \rightarrow \boxed{\frac{N}{2^k} \leq 1}$$

$k$ =máximo de divisiones por 2=número máximo de iteraciones

$$2^{k-1} < N \leq 2^k \Rightarrow k = \lceil \log_2 N \rceil \geq n_{\text{BBin}}(T, 1, N, k)$$

- **Ejercicio:** Realizar el cálculo anterior, contando todas la sentencias básicas en lugar de solamente la OB
- **Solución** (opinable):  $n_{\text{BBin}} \leq 5 \lceil \log_2 N \rceil + 2$

# Observaciones sobre el tae

- Observación 1: Parece que para cualquier algoritmo **A**, a sus entradas **I** se les puede asignar un **tamaño de entrada** ( $\tau(I)$ ) y se puede encontrar una cierta **función** de **N**,  $f_A(N)$  tal que:

$$\text{tae}_A(I) = n_A(I) \leq f_A(\tau(I))$$

- En los ejemplos vistos

A	I	$\tau$	$f_A$
SelectSort	(T,P,U)	U-P+1	$N^2/2 - N/2$
BBin	(T,P,U,k)	U-P+1	$\lceil \lg N \rceil$

- Observación 2: en el tae hay un **término dominante**; los demás importan menos

# Observaciones sobre tae

- Observación 3: El tae permite
  - **Generalizar los tiempos de ejecución** en función del tamaño de la entrada.
  - Estimar **tiempos reales** de ejecución.
- Ejemplo: SelectSort con  $I$  tal que  $\tau(I)=N$  e  $I'$  tal que  $\tau(I')=2N$ 
  - $\text{tae}_{\text{SSort}}(I) = N^2/2 + \dots$  y
  - $\text{tae}_{\text{SSort}}(I') = (2N)^2/2 + \dots = 4N^2/2 + \dots \sim 4\text{tae}_{\text{SSort}}(I)$
- En general si  $\tau(I')=kN$  se tiene  $\text{tae}_{\text{SSort}}(I') \sim k^2\text{tae}_{\text{SSort}}(I)$
- Si una tabla con  $N=1000$  tarda 1seg. en ser ordenada,  
Entonces

Tamaño	1000	2000	10000	100000
Tiempo real	1s	4s	100s	10000s

# Ejemplo: método de la Burbuja

- Ordenación por burbuja (BurbujaSort\_v1).

```
BurbujaSort_v1(Tabla T, ind P, ind U)
  para i de U a P+1:
    para j de P a i-1:
      si T[j]>T[j+1]:
        swap(T[j],T[j+1]);
```

- Índice j: burbuja que intenta subir. Si no puede, cambiamos de burbuja
- Tras cada iteración en i el máximo de la subtabla está en la posición i
- La tabla se va ordenando de derecha a izquierda

# Funcionamiento de la Burbuja

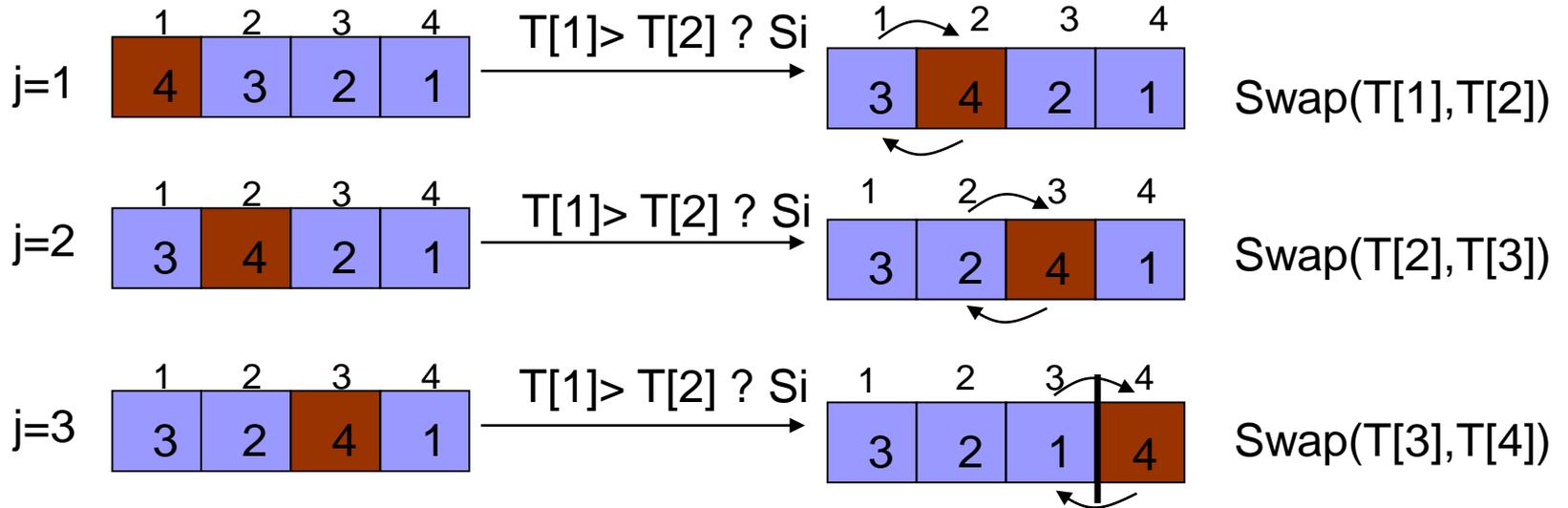
## ■ BubbleSort\_v1

Entrada:  $T=[4,3,2,1]$ ,  $P=1$ ,  $U=4$ .

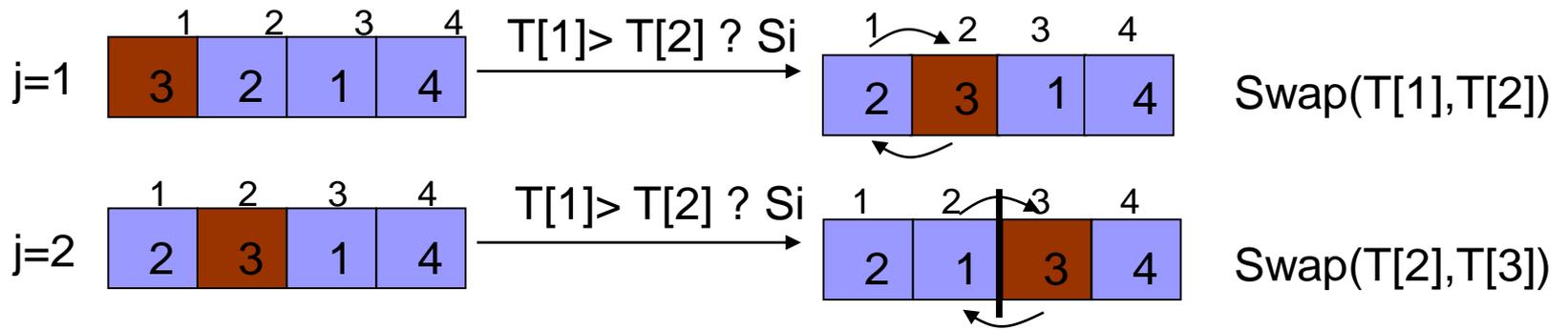
Burbuja

i	j	Operaciones	Tabla
4	1	$T[1] > T[2]$ ? Si swap( $T[1], T[2]$ )	3, 4, 2, 1 ↑ ↑
4	2	$T[2] > T[3]$ ? Si swap( $T[2], T[3]$ )	3, 2, 4, 1 ↑ ↑
4	3	$T[3] > T[4]$ ? Si swap( $T[3], T[4]$ )	3, 2, 1, 4 ↑ ↑
3	1	$T[1] > T[2]$ ? Si swap( $T[1], T[2]$ )	2, 3, 1, 4 ↑ ↑
3	2	$T[2] > T[3]$ ? Si swap( $T[2], T[3]$ )	2, 1, 3, 4 ↑ ↑
2	1	$T[1] > T[2]$ ? Si swap( $T[1], T[2]$ )	2, 1, 3, 4 ↑ ↑

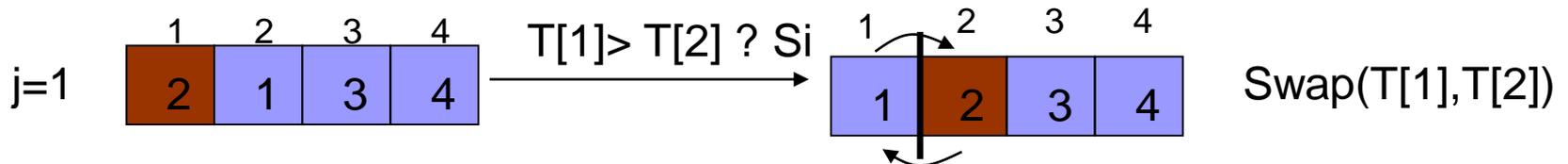
i=4



i=3



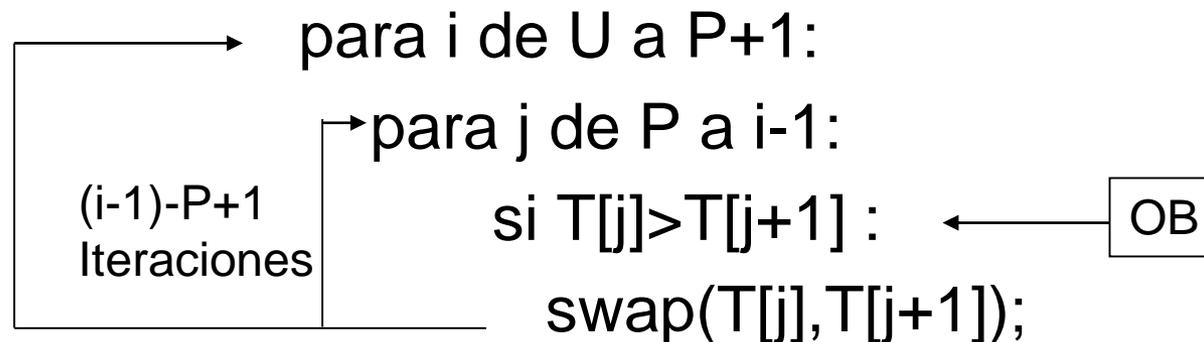
i=2



# Rendimiento de la Burbuja

- OB? CDC!!
- Ordenación por burbuja (BurbujaSort\_v1).

BurbujaSort\_v1(Tabla T, ind P, ind U)



- Entonces

$$n_{BSort\_v1}(T, 1, N) = \sum_{i=2}^N \sum_{j=1}^{i-1} 1 = \sum_{i=2}^N (i-1) = \sum_{i=1}^{N-1} i = \frac{N^2}{2} - \frac{N}{2}$$

# Observaciones sobre el tae de la Burbuja

- El tae de BurbujaSort\_v1 (y de SelectSort) **NO** depende de la entrada (es siempre  $N^2/2 - N/2$ ).
  - Se tarda lo mismo en ordenar una tabla desordenada que una tabla ordenada!!!
- En SelectSort no tiene remedio:
  - No se puede saber si T ya está ordenada



# Mejorando la Burbuja

- En BurbujaSort\_v1 sí se puede saber si T está ordenada
- Ejemplo:  $T = [ 1 \ 2 \ 3 \ 4 ]$ 
  - En la primera iteración no se hacen swaps, pues la subtabla está ordenada
- Se puede añadir un flag para comprobar si se hacen o no swaps en el bucle interno
- Si no se hacen, la subtabla ya está ordenada y también la tabla completa
- Se puede terminar el algoritmo y mejorar así el rendimiento.

# Rendimiento de la V2 de la Burbuja

- Ordenación por burbuja (BurbujaSort\_v2).

```
BurbujaSort_v2(Tabla T, ind P, ind U)
```

```
Flag=1; i=U;
```

```
mientras (Flag==1 && i≥P+1) :
```

```
Flag=0;
```

```
para j de P a i-1 :
```

```
si T[j]>T[j+1] :
```

```
swap(T[j],T[j+1]); Flag=1;
```

```
i--;
```

- Ahora se tiene  $n_{BSort\_v2}([1,2,\dots,N])=N-1$  y para el caso peor se sigue teniendo  $n_{BSort\_v2}(T,1,N) \leq N^2/2 - N/2$



# Pero ...

- Estamos trabajando con algoritmos de manera individual ...
- Pero queremos comparar unos algoritmos con otros
- Q: ¿Cómo comparar dos algoritmos?

# Comparación de Algoritmos

- Sólo posible sobre algoritmos “parecidos”
  1. Que resuelvan el mismo problema (ordenación, búsqueda)
  2. Que tengan la misma operación básica.
- Agrupamos los algoritmos en familias **F** que cumplan las condiciones 1 y 2.
- Ejemplo  
 $F = \{\text{Algoritmos de ordenación por comparación de clave}\}$   
 $= \{\text{InsertSort, SelectSort, BubbleSort, QuickSort, MergeSort, \dots}\}$

# Comparación de Algoritmos II

- Método a seguir
  1. Para todo algoritmo  $A \in \mathbf{F}$  encontrar  $f_A(\mathbf{N})$  tal que
$$n_A(I) \leq f_A(\tau(I))$$
con  $\tau(I)$  el tamaño de entrada
  2. Diremos que  $A_1$  es mejor que  $A_2$  si
$$f_{A_1}(\mathbf{N})$$
 es “menor” que  $f_{A_2}(\mathbf{N})$
- Sólo tiene sentido la comparación para entradas “grandes” = **asintótica**
- Sólo compararemos  $f_{A_1}(\mathbf{N})$  y  $f_{A_2}(\mathbf{N})$  cuando  $\mathbf{N}$  es grande (es decir para  $N \rightarrow \infty$ ):

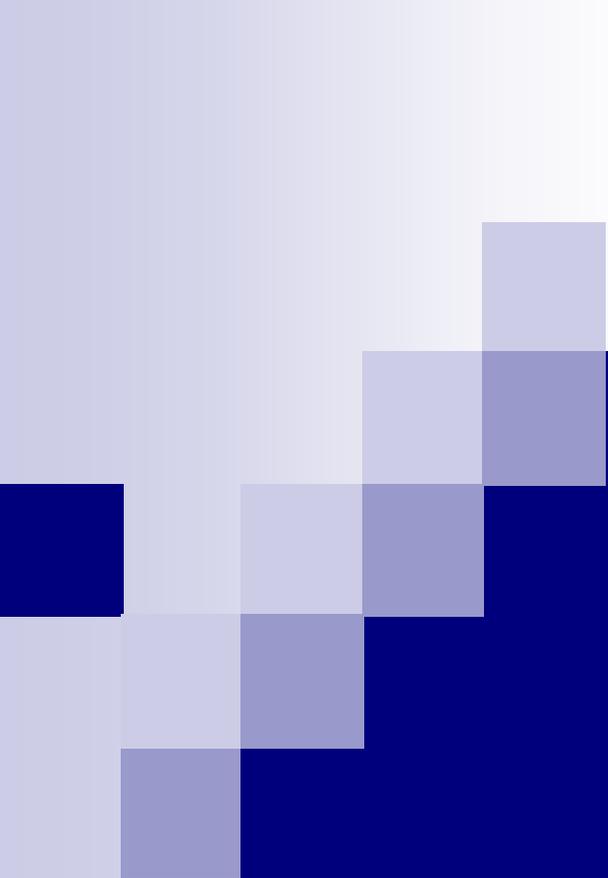
# En esta sección hemos aprendido...

- La medida básica de **eficacia** de algoritmos
- El concepto de **operación básica**
- El funcionamiento de algunos algoritmos simples (BBin, BurbujaSort, SelectSort).
- El cálculo del **tae** de los algoritmos simples anteriores como **número de ejecuciones de su OB.**
- Cómo enfocar la **comparación del algoritmos**



# Herramientas y técnicas a trabajar ...

- Sumatorios
- Estimación del número de iteraciones en bucles
- Buen conocimiento del funcionamiento del algoritmo a analizar
- Problemas a resolver: los recomendados de la sección 1 de las hojas de ejercicios y problemas (al menos!!)



## 1.2 Estimación del crecimiento de funciones

# Comparación asintótica de funciones: o

- Definición 1 ( $f=o(g)$ ,  $f \ll g$ ): Si  $f$  y  $g$  son funciones positivas, decimos que  **$f=o(g)$**  si

$$\lim_{N \rightarrow \infty} \frac{f(N)}{g(N)} = 0$$

- Ejemplos

$$f=N^k, \quad g=N^{k+\varepsilon}$$

$$f=\log(N), \quad g=N^\varepsilon$$

$$f=(\log(N))^k, \quad g=N^\varepsilon$$

(ejercicio; se puede hacer por L'Hôpital)

# Comp. asintótica de funciones: O

- Definición 2 ( $f=O(g)$ ,  $f \leq g$ ):  **$f=O(g)$**  si existen  $N_0$  y  $C > 0$  tales que para todo  $N \geq N_0$ ,  $f(N) \leq Cg(N)$

- Ejemplo

$$f=N^2, \quad g=N^2+\sqrt{N}$$

Y también  **$g = O(f)$**

- Interpretación:  $g$  es mayor o igual que  $f$   
***“eventualmente y con ayuda”***

# Observaciones sobre $f=O(g)$

- $f=o(g) \Rightarrow f=O(g)$

- El recíproco a lo anterior es falso:

$$f=O(g) \not\Rightarrow f=o(g)$$

- Las constantes “no importan” en  $O$ , es decir, si

$$f=O(g) \Rightarrow f=O(kg) \text{ donde } k > 0.$$

- Si  $f=O(g)$  y  $h=o(g)$  entonces  $f=O(g+h)$ , pues  $g+h = O(g)$

- Decir  $f=O(N^2+N)$  no añade precisión

- Basta con  $f=O(N^2)$

# Comp. asintótica de funciones: $\Theta$

- Definición 3 ( $f = \Omega(g)$ ,  $f \geq g$ )

$$f = \Omega(g) \text{ si } g = O(f)$$

- Definición 4 ( $f = \Theta(g)$ ,  $f = g$ )

$$f = \Theta(g) \text{ si } f = O(g) \text{ y } f = \Omega(g)$$

- Observamos que

$$f = \Theta(g) \Leftrightarrow g = \Theta(f)$$

- Además si  $\lim_{N \rightarrow \infty} \frac{f(N)}{g(N)} = L \neq 0$ , entonces  $f = \Theta(g)$

# Comp. asintótica de funciones: $\sim$

- Definición 5 ( $f \sim g$ ): Decimos que  $f$  es asintóticamente equivalente a  $g$  si

$$\lim_{N \rightarrow \infty} \frac{f(N)}{g(N)} = 1$$

- Ejemplo

$$f(N) = N^2 + \sqrt{N} + \log N, \quad g(N) = N^2$$

- Observación

$$f \sim g \Rightarrow f = \Theta(g)$$

***Pero el recíproco es falso***

# Comp. asint. de funciones: $f=g+O(h)$

- Definición 6 ( $f=g+O(h)$ )

Si  $h=o(g)$  entonces decimos que  $f=g+O(h)$

si  $|f-g|=O(h)$

- Ejemplo

$$f(N) = N^2 + \sqrt{N} + \log(N), \quad g(N) = N^2.$$

Entonces  $f = g + O(\sqrt{N})$

# Comparación asintótica de funciones

- Tenemos una escala de **precisión decreciente**
  - Si  $f=g+O(h)$  entonces  $f\sim g$
  - Si  $f\sim g$ , entonces  $f=\Theta(g)$
  - Si  $f=\Theta(g)$  entonces  $f=O(g)$
- Pero *los recíprocos son falsos*

# Comparación asintótica de funciones II

Además:

- Si  $f=O(g)$  y  $f' = O(g')$ , entonces

$$f + f' = O(g + g')$$

$$f f' = O(g g')$$

- **Consecuencia:** si  $P(N)$  es un polinomio de grado  $k$  se tiene que

$$P(N)=a_k N^k+O(N^{k-1})$$

# Crec. de progs. aritméticas y geométricas

Ya hemos visto

$$f(N) = S_N = \sum_{i=1}^N i = \frac{N(N+1)}{2} = \frac{N^2}{2} + O(N)$$

Progresión geométrica (**muy importante!!**)

$$S_N = \sum_{i=1}^N x^i = \frac{x^{N+1} - x}{x-1} = \frac{UR - P}{R-1}$$

Observaciones:

- Si  $x = 1$   $S_N = N$
- Si  $x > 1$   $S_N = \Theta(x^N)$
- Si  $x < 1$   $S_N = \frac{x - x^{N+1}}{1-x} \xrightarrow{N \rightarrow \infty} \frac{x}{1-x}$

# Crecimiento de funciones derivadas

- Serie derivada

$$S_N = \sum_{i=1}^N ix^i = x \frac{d}{dx} \sum_{i=1}^N x^i = \Theta(Nx^N)$$

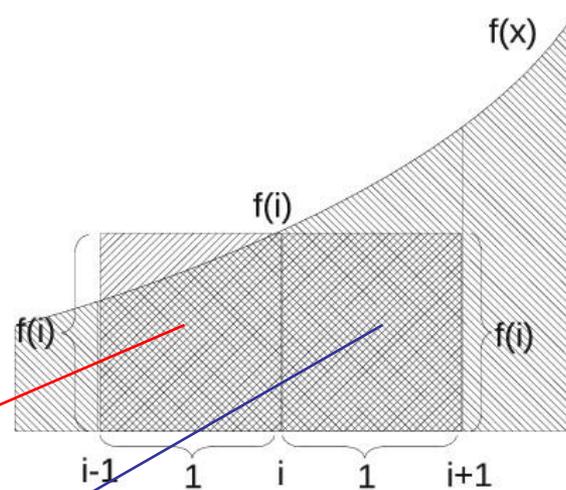
- Suma de potencias cúbicas

$$S_N = \sum_{i=1}^N i^2 = \frac{N(N+1)(2N+1)}{6} = \frac{N^3}{3} + O(N^2)$$

- Nos interesa el crecimiento y no tanto la fórmula cerrada
- Se puede simplificar?

# Estimaciones de crecimiento de sumas

- ¿Que hacer cuando no tenemos una expresión cerrada para una suma  $S_N = \sum_{i=1}^N f(i)$  ?
- Podemos aproximar la suma mediante integrales.



$f(x)$  creciente

$$\int_{i-1}^i f(x) dx \leq f(i) \leq \int_i^{i+1} f(x) dx \Rightarrow \sum_{i=1}^N \int_{i-1}^i f(x) dx \leq \sum_{i=1}^N f(i) \leq \sum_{i=1}^N \int_i^{i+1} f(x) dx \Rightarrow$$

$$\int_0^N f(x) dx \leq S_N \leq \int_1^{N+1} f(x) dx$$

# Estimaciones de crecimiento de sumas II

- Análogamente, si  $f(x)$  es decreciente

$$\int_0^N f(x)dx \geq S_N \geq \int_1^{N+1} f(x)dx$$

- Algunas sumas que se estiman por este método

$$S_N = \sum_{i=1}^N i^k \rightarrow S_N \sim \frac{N^{k+1}}{k+1} \quad \text{o también} \quad S_N = \frac{N^{k+1}}{k+1} + O(N^k)$$

$$S_N = \sum_{i=1}^N \log(i) = \log(N!) \rightarrow S_N \sim N \log(N)$$

$$H_N = \sum_{i=1}^N \frac{1}{i} \rightarrow H_N \sim \log(N) \quad (\text{N - ésimo número armónico})$$

**Nota:** Se recomienda seguir en la pizarra o en los apuntes el método por el cual se han obtenido estas expresiones, ya que cada una de ellas presenta peculiaridades sobre la aplicación directa de la fórmula de la acotación por integrales.



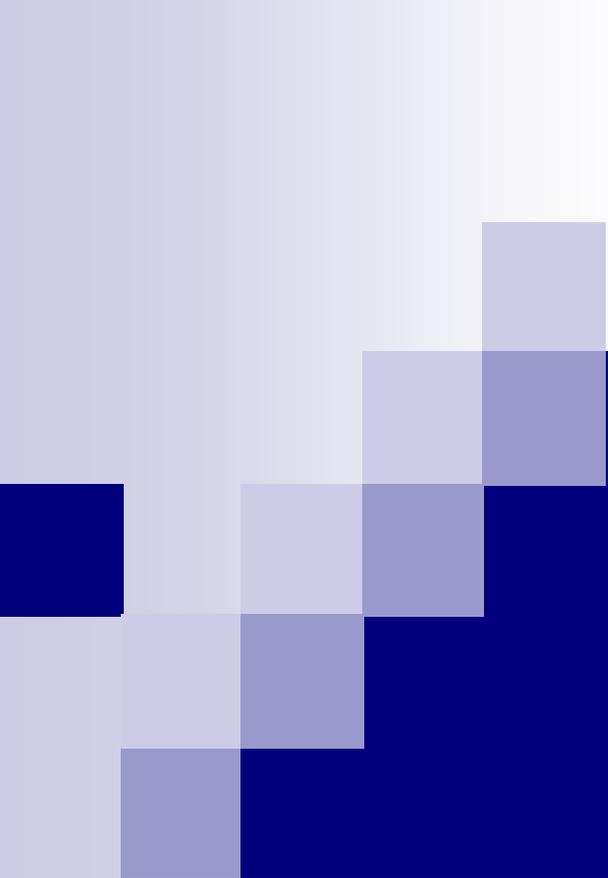
# En esta sección hemos aprendido...

- Los **diferentes tipos** de comparaciones asintóticas entre funciones.
- Cómo **estimar el crecimiento de algunas funciones** mediante fórmulas cerradas o mediante integrales.



# Herramientas y técnicas a trabajar ...

- Trabajo básico con las estimaciones  $o$ ,  $O$ , etc
- Aplicación de las estimaciones  $o$ ,  $O$ , etc.
- Estimación mediante integrales del crecimiento asintótico de sumas
- Problemas a resolver: los recomendados de la sección 2 de las hojas de ejercicios y problemas (al menos!!)



# 1.3 Complejidad de Algoritmos

# Complejidad de Algoritmos

- Hasta ahora, el trabajo realizado por un algoritmo dependía de cada entrada en particular:  $n_A(I) \leq f_A(\tau(I))$ .
- En algunos algoritmos hay un desequilibrio muy grande entre diferentes entradas, por ejemplo

$$N_{\text{BSort\_v2}}([N, N-1, \dots, 1], 1, N) = 1/2(N^2 - N) \quad (\text{mucho})$$

$$N_{\text{BSort\_v2}}([1, 2, \dots, N], 1, N) = (N-1) \quad (\text{muy poco})$$

- Nos gustaría precisar la definición del trabajo que realiza un algoritmo,
- Para ello definimos el **Espacio de entradas** de tamaño **N** de un algoritmo **A** como:

$$E_A(N) = \{I \text{ entrada de } A / \tau(I) = N\}$$

## Ejemplo: BSort

- $E_{\text{BSort}}(N) = \{ (T, P, U) : U - P + 1 = N \} \dots ???$
- Con,  $P, U$  y  $T$  cualquiera este espacio es inmanejable (demasiado grande)  $\Rightarrow$  es necesario hacer este espacio mas pequeño.
- Una simplificación sencilla es tomar  $P=1$  y  $U=N$ .
- Otra simplificación es tomar  $T \in \Sigma_N$  (permutaciones de tamaño  $N$ ), ya que lo importante es el desorden entre los elementos de la tabla.
- Con estas simplificaciones

$$E_{\text{BSort}}(N) = \Sigma_N \quad \text{y} \quad |E_{\text{BSort}}(N)| = N!$$

## Ejemplo: BBin

- $E_{\text{BBin}}(N) = \{ (T, P, U, k) : k \text{ cualquiera, } U - P + 1 = N, T \text{ ordenada} \}$
- Una simplificación sencilla es tomar  $P=1$  y  $U=N$ .
- Como  $T$  está ordenada, tomamos  $T=[1 \dots N]$
- Como claves en la tabla tomamos  $k=1, \dots, N$
- BBin hace el mismo número de cdcs para cualquier  $k$  no en  $T \Rightarrow$  añadimos la clave de error **otra**
- Con estas simplificaciones  
 $E_{\text{BBin}}(N) = \{ 1, 2, \dots, N, \text{otra} \}$  y  $|E_{\text{BBin}}(N)| = N+1$

# Casos mejor, peor, medio

## Definiciones de complejidad

Caso peor

$$W_A(N) = \max \{n_A(I) / I \in E_A(N)\}$$

Caso mejor

$$B_A(N) = \min \{n_A(I) / I \in E_A(N)\}$$

Caso medio

$$A_A(N) = \sum_{I \in E_A(N)} n_A(I) p(I)$$

donde  $p(I)$  es la probabilidad con la que aparece la entrada  $I$

Observación:  $B_A(N) \leq A_A(N) \leq W_A(N)$

# Complejidad en el caso peor

- ¿Cómo estimar el caso peor de un algoritmo?
  - Paso 1: Encontrar  $f_A(N)$  tal que si  $\tau(I)=N$  se tiene
 
$$n_A(I) \leq f_A(N)$$
  - Paso 2: Encontrar una entrada  $\hat{I}$ , que sea la entrada peor del algoritmo,  $\hat{I}$  tal que
 
$$n_A(\hat{I}) \geq f_A(N)$$
    - Por el paso 1, se tiene  $W_A(N) \leq f_A(N)$
    - Por el paso 2  $W_A(N) \geq f_A(N)$
  - Por tanto  $W_A(N) = f_A(N)$
- El caso mejor se estima de una forma similar
  - se deja como ejercicio escribir ambos pasos para el caso mejor).

# Caso peor de BSort

- Ejemplo:  $W_{\text{BSort2}}(N)$

- (1) Ya vimos que para toda  $\sigma \in \Sigma_N$   
 $n_{\text{BSort2}}(\sigma) \leq N^2/2 + O(N)$

- (2) Si  $\sigma = [N, N-1, N-2, \dots, 1]$  entonces  
 $n_{\text{BSort2}}(\sigma) = N^2/2 + O(N)$

- Por (1) y (2) se tiene

$$W_{\text{BSort2}}(N) = N^2/2 + O(N)$$

- Ejercicio: demostrar que  $B_{\text{BSort2}}(N) = N-1$

# Complejidad en el caso medio

- El caso medio suele ser el mas difícil de calcular.
- En general (aunque no siempre) supondremos equiprobabilidad en el espacio de entradas, es decir

$$p(I)=1/|E_A(N)|$$

- Ejemplos:
  - En BSort  $p(\sigma)=1/N!$ , y
  - En BBin  $p(k)=1/(N+1)$

# Caso medio de búsqueda lineal

- Consideramos la Búsqueda Lineal equiprobable.

```
BLin(tabla T, ind P, ind U, clave k)
    para i de P a U;
        si T[i]==k;
            devolver k;
    devolver ERROR;
```

- Operación básica: CDC (si  $T[i]==k$ )
- $E_{\text{BLin}}(N) = \{1, 2, \dots, N, \text{otra}\}$

# Caso medio de búsqueda lineal II

- $E_{\text{BLin}}(N) = \{1, 2, 3, \dots, N, \text{otro}\}, |E_{\text{BLin}}(N)| = N+1$
- Se tiene
  - $p(k=i) = 1/(N+1) \quad (1 \leq i \leq N)$   
 $p(k=\text{otro}) = 1/(N+1)$
  - Si  $k \neq T[i] \Rightarrow n_{\text{Blin}}(k) = N$   
Si  $k = T[i] \Rightarrow n_{\text{Blin}}(k) = i$
- En consecuencia
  - $W_{\text{Blin}}(N) = N,$
  - $B_{\text{Blin}}(N) = 1.$
  - $A_{\text{Blin}}(N) = N/2 + O(1)$

## Caso medio de búsqueda lineal III

### ■ Ejemplo: B. lineal con éxito no equiprobable.

- En este caso se tiene que  $p(k == T[i]) = \frac{1}{C_N} f(i)$
- $C_N$  es una constante de normalización que garantiza

$$\sum_{i=1}^N \frac{1}{C_N} f(i) = 1, \text{ es decir } C_N = \sum_{i=1}^N f(i)$$

$$\begin{aligned} A_{BLin}(N) &= \sum_{i=1}^N n_{BLin}(k = T[i]) p(k == T[i]) = \sum_{i=1}^N i \frac{1}{C_N} f(i) = \\ &= \frac{1}{C_N} \sum_{i=1}^N i f(i) = \frac{S_N}{C_N}, \text{ donde } S_N = \sum_{i=1}^N i f(i) \end{aligned}$$

- $S_N$  y  $C_N$  se aproximan (por ejemplo, por integrales)
- Se obtiene finalmente una aproximación para A.

# Caso medio de búsqueda lineal IV

- Por ejemplo, si tenemos  $p(k == T[i]) = \frac{1}{C_N} \frac{\log(i)}{i}$

$$\text{entonces } C_N = \sum_{i=1}^N \frac{\log(i)}{i} \quad \text{y} \quad S_N = \sum_{i=1}^N i \frac{\log(i)}{i} = \sum_{i=1}^N \log(i)$$

- Aproximando por integrales se tiene

$$C_N \sim \frac{1}{2} (\log(N))^2 \quad \text{y} \quad S_N \sim N \log(N)$$

- De lo que se **obtiene fácilmente**:

$$A_{BLin}(N) \sim \frac{2N}{\log(N)}$$

- Este último paso no es difícil, pero tampoco inmediato pues hay que **demostrar** la última equivalencia asintótica.



## En esta sección hemos aprendido...

- Las expresiones de los casos mejor, peor y medio de un algoritmo.
- Cómo calcular el caso mejor peor y medio de algunos algoritmos simples (BSort y Blin)



# Herramientas y técnicas a trabajar ...

- Cálculo de casos peores y mejores de algoritmos simples
- Cálculo de caso medio de búsqueda lineal y variantes
- Problemas a resolver: los recomendados de la sección 3 de las hojas de ejercicios y problemas (al menos!!)