Programación 2

La previa:

Tiempo ejecución de programas iterativos

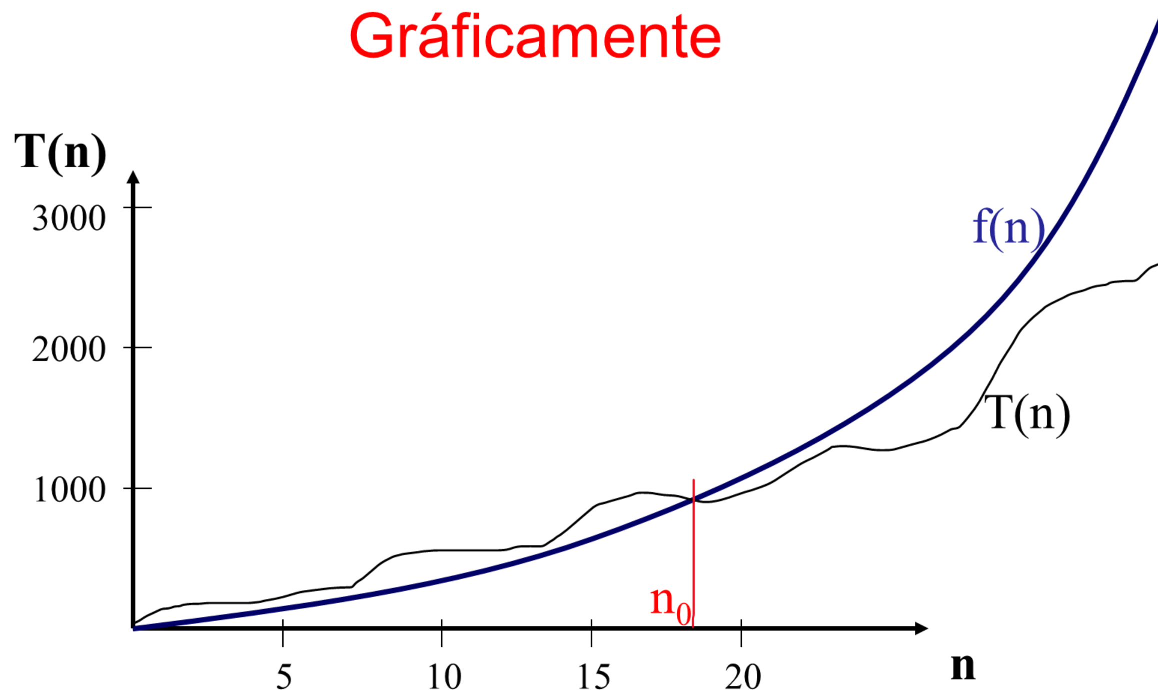
T(n)

- **T(n)** = tiempo de ejecución de un programa con una entrada de tamaño **n**
  = número de instrucciones ejecutadas en un computador idealizado con una entrada de tamaño **n**
- Para el problema de ordenar una secuencia de elementos, n sería la cantidad de elementos
  Ejemplo: $T(n) = c.n^2$, donde *c* es una constante
- $T^{peor}(n)$ = tiempo de ejecución para el peor caso
  $T^{prom}(n)$ = tiempo de ejecución del caso promedio
  Nos centraremos en $T^{peor}(n)$ y lo llamaremos simplemente T(n).

Velocidad de crecimiento - O(n)

T(n) es **O(*f*(n))** "orden *f(n)*" si existen constantes positivas *c* y *n0* tales que $T(n) \leq c.f(n)$ cuando $n \geq n0$. *f*(n) es una **cota superior** para la velocidad (taza) de crecimiento de un programa con tiempo de ejecución T(n)

Gráficamente



Velocidad de crecimiento - O(n)

- T(n) es **O(*f*(n))** "orden *f(n)*" si existen constantes positivas *c* y *n0* tales que $T(n) \leq c.f(n)$ cuando $n \geq n0$. *f*(n) es una **cota superior** para la velocidad (taza) de crecimiento de un programa con tiempo de ejecución T(n)
- Ejemplo:
  - $T(n) = 3n^3 + 2n^2$ es $O(n^3)$
    - Sean $n0 = 0$ y $c = 5$, $3n^3 + 2n^2 \leq 5\, n^3$, para $n \geq 0$
    - También T(n) es $O(n^4)$, pero sería una aseveración más débil que decir que es $O(n^3)$

Algunas velocidades de crecimiento típicas

Para *n* grande

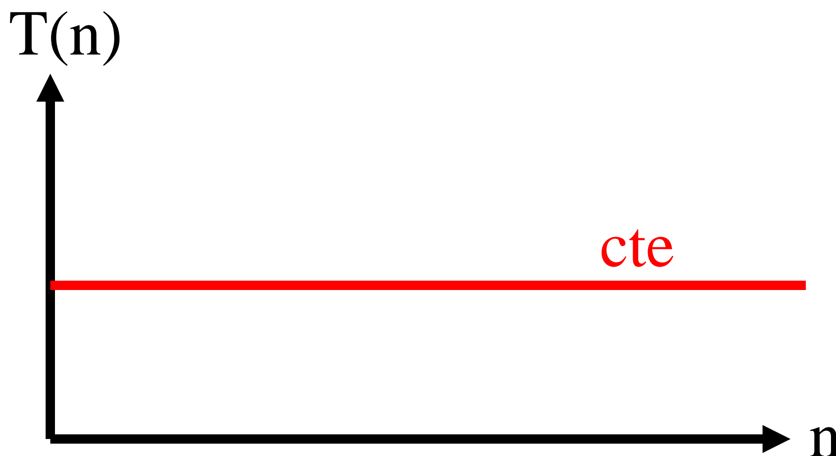
Función	Nombre
c	constante
log(n)	logarítmica
$\log^2(n)$	log-cuadrado
n	lineal
n.log(n)	
n^2	cuadrática
n^3	cúbica
2^n	exponencial

Tiempo constante – O(1)

*T(n) es **O(f(n))** "orden f(n)" si existen constantes positivas c y n0 tales que* *$T(n) \leq c.f(n)$ cuando $n \geq n0$*.

Si T(n) = cte, T(n) es O(1), ya que existe *c* (c=cte) y *n0* (n0=0), tales que:

$$cte \leq cte.1 \text{ cuando } n \geq 0.$$



Cálculo del tiempo de ejecución

- **Regla de la Suma**:

Si $T1(n)$ es $O(f1(n))$ y $T2(n)$ es $O(f2(n))$ entonces **$T1(n)+T2(n)$** es **$O(max\ (f1(n), f2(n)))$**

$\Rightarrow$ Puede usarse para calcular el tiempo de ejecución de una secuencia de pasos de programa.

Ejemplo: supongamos 3 procesos secuenciales con tiempos de ejecución $O(n^2)$, $O(n^3)$ y $O(n.log(n))$. El tiempo de ejecución de la composición es $O(n^3)$.

¿Cómo se demuestra la regla de la Suma?

Cálculo de T(n) - Algunas reglas

- Para una **asignación** (lectura/escritura e instrucciones básicas) es en general O(1) (tiempo constante)
- Para una **secuencia** de pasos se determina por la regla de la suma (dentro de un factor cte, el "máximo")
- Para un " if (***Cond***) ***Sent*** " es el tiempo para ***Sent*** más el tiempo para evaluar ***Cond*** (este último en general O(1) para condiciones simples)
- Para un " if (***Cond***) $\boldsymbol{Sent_1}$ else $\boldsymbol{Sent_2}$ " es el tiempo para evaluar ***Cond*** más el máximo entre los tiempos para $\boldsymbol{Sent_1}$ y $\boldsymbol{Sent_2}$

Cálculo de T(n) - Algunas reglas (cont)

- Para un **ciclo** es la suma, sobre todas las iteraciones del ciclo ($\sum$), del tiempo de ejecución del cuerpo y del empleado para evaluar la condición de terminación (este último suele ser O(1)).

  $\Rightarrow$ A menudo este tiempo es, despreciando factores constantes, el producto del número de iteraciones del ciclo y el mayor tiempo posible para una ejecución del cuerpo.

Ejemplo simple: búsqueda de un elemento en un arreglo

Pensar en un algoritmo simple para este problema

Ejemplo: búsqueda de un elemento en un arreglo

```
bool buscar (int * A, unsigned int n, int x) {
        int i;
        for (i=0; i<n && A[i]!=x; i++);
        return (i!=n);
}
```

- ¿Peor caso?
- ¿Caso promedio?
- ¿Mejor caso?

Ejemplo: búsqueda de un elemento en un arreglo

```
bool buscar (int * A, unsigned int n, int x) {
        for (int i=0; i<n; i++);
                if (A[i]==x) return true;
        return false;
}
```

- ¿Peor caso?
- ¿Caso promedio?
- ¿Mejor caso?

Ejemplo: búsqueda de un elemento en un arreglo

```
bool buscar (int * A, unsigned int n, int x) {
        bool res = false;
        for (i=0; i<n && !res; i++)
                if (A[i]==x) res = true;
        return res;
}
```

- ¿Peor caso?
- ¿Caso promedio?
- ¿Mejor caso?

Ejemplo: búsqueda de un elemento en un arreglo

```
// Pre: el arreglo A está ordenado de menor a mayor
bool buscar (int * A, unsigned int n, int x) {
        for (int i=0; i<n && A[i]<=x; i++);
                if (A[i]==x) return true;
        return false;
}
```

¿Mejora el peor caso?

Cálculo de T(n) - Ejemplos (cont)

Considere el siguiente fragmento de programa:

```
for (i=0; i<n-1; i++)
    for (j=n-1; i<j; j--)
        if (A[j-1] > A[j])  intercambiar (A[j], A[j-1])
```

¿Qué hace?, ¿Cuál es su tiempo de ejecución?

Despreciando algunos factores constantes, podemos calcular T(n) y luego O(n) para: $T(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} cte$

Nota: Para este problema existen algoritmos O(n.log(n))

Cálculo de T(n) - Ejercicios

Considere el siguiente fragmento de código:

```
...
if (n%2==0)
        for (i=0; i<n; i++)
                P(i,i); // donde P tiene O(1) peor caso
else    for (i=0; i<n; i++)
                for (j=0; j<n; j++)
`                       P(i,j);
...
```

¿Cuál es su orden (O) de tiempo de ejecución en el peor caso?

Más ejercicios...

Considere la siguiente función en C++, definida sobre un arreglo de enteros de tamaño n:

```
bool F (int * A, unsigned int n)
{       bool res = true;
        int i, j;
        for (i=0; i<n; i++)
                for (j=0; j<n; j++)
                        if (i<j) res = res && (A[j]<A[i]);
        return res;
}
```

a) ¿Qué retorna (conceptualmente) la función F, dado un arreglo de enteros de tamaño n?

b) ¿Cuál es el orden (O) de tiempo de ejecución para el peor caso de la función F.

c) El problema que resuelve F, ¿podría resolverse en un menor orden de tiempo de ejecución en el peor caso?.

Más ejercicios...

```
bool F (int * A, unsigned int n)
{    int i;
     for (i=1; i<n && A[i-1]>A[i]; i++);
     return (i==n);
}
```

Más ejercicios...

Considere la siguiente función en C++, definida sobre un arreglo de enteros de tamaño n:

```
bool F (int * A, unsigned int n)
{      bool res = true;
       int i, j;
       for (i=0; i<n; i++)
             for (j=0; j<n; j++)
                    if (i!=j) res = res && (A[i]!=A[j]);

       return res;
}
```

a) ¿Qué retorna (conceptualmente) la función F, dado un arreglo de enteros de tamaño n?

b) ¿Cuál es el orden (O) de tiempo de ejecución para el peor caso de la función F.

c) Si se sabe que el arreglo A sólo puede contener valores enteros en el rango [0 : n-1], el problema que resuelve F ¿podría resolverse en un menor orden de tiempo de ejecución en el peor caso?

Más ejercicios...

```
bool F (int * A, unsigned int n)
{   int i;
    int * Pertenece = new bool[n]; // memoria extra
    for (i=0; i<n; i++)
        Pertenece[i] = false;
    for (i=0; i<n && !Pertence[A[i]]; i++)
        Pertence[A[i]] = true;
    delete [] Pertenece;
    return (i==n);
}
```

Más ejercicios...

Considere la siguiente función en C++, definida sobre un arreglo de enteros de tamaño n:

```
bool F (int * A, int * B, int n)
{       bool res = true;
        int i, j;
        for (i=0; i<n; i++)
                for (j=0; j<n; j++)
                        if (i+j == n-1)
                                res = res && (A[i]==B[j]);
        return res;
}
```

a) ¿Qué retorna (conceptualmente) la función F, dado un arreglo de enteros de tamaño n?

b) ¿Cuál es el orden (O) de tiempo de ejecución para el peor caso de la función F.

c) El problema que resuelve F ¿podría resolverse en un menor orden de tiempo de ejecución en el peor caso?

Más ejercicios...

```
bool F (int * A, unsigned int n)
{  bool res =  true;
   int i;
   for (i=0; i<n; i++)
         res = res && (A[i]==B[n-1-i]);
   return res;
}
```

Más ejercicios...

```
bool F (int * A, unsigned int n)
{  bool res =  true;
   int i;
   for (i=0; i<n && res; i++)  // ¿mejora el peor caso?
         res = res && (A[i]==B[n-1-i]);
   return res;
}
```

Más ejercicios... (*sorting*)

Desarrolle algoritmos de ordenación sobre arreglos de enteros de tamaño n siguiendo las siguientes estrategias:

- Busca el mínimo elemento, lo pone al inicio y luego aplica la misma estrategia a los restantes elementos (todos salvo el primero).
- Recorre el arreglo e inserta de manera ordenada cada elemento en un nuevo arreglo originalmente vacío.

Calcule el tiempo de ejecución y el orden (O) del primer algoritmo, conocido como *select sort*, y del segundo algoritmo, conocido como *insert sort*.

Selection Sort

Consideremos un arreglo **lista** de largo n ([0 : n-1]), n>0

```
for (int i = 0 ; i < n-1 ; i++){
    int pos_min = i;
    for (int j = i + 1 ; j < n ; j++){
        if (lista[j] < lista[pos_min]){
            pos_min = j;
        }
    }
    intercambiar (lista, i, pos_min); /* intercambia de
    lista los elementos en las posiciones i y pos_min */
}
```