

Práctico 2: Evaluación de Algoritmos

Año 2026

Ejercicio 1:

Desarrolle, en pseudocódigo, un algoritmo que dado un número entero $n > 0$, devuelva la suma de los dobles de los números entre 1 y n . (Ej.: $\text{SumaDoble}(3) = 2 * 1 + 2 * 2 + 2 * 3 = 12$)

- a) ¿Qué tipo de operaciones matemáticas realiza su algoritmo?
- b) ¿Cuántas operaciones de cada tipo realiza su algoritmo?

Ejercicio 2:

Suponga que tiene un conjunto de números almacenados en un archivo, pero no se sabe cuántos hay. Diseñe un algoritmo, en pseudocódigo, que calcule el promedio de los mismos.

- a) ¿Qué tipo de operaciones realiza su algoritmo?
- b) ¿Cuántas operaciones de cada tipo realiza su algoritmo?
- c) Si lo que se quiere medir es la *cantidad de sumas realizadas* ¿Cuál sería el costo de su algoritmo?

Ejercicio 3:

Desarrolle, en pseudocódigo, un algoritmo que cuente las letras mayúsculas que hay en un archivo de texto.

- a) ¿Cuántas comparaciones realiza su algoritmo?
- b) ¿Cuál es la menor cantidad de incrementos al contador que se puede hacer?
- c) ¿Cuál es el mayor número de incrementos? (Considerar que N es la cantidad de caracteres del archivo).
- d) Si la función de costo elegida fuera *cantidad de comparaciones* ¿Cuál sería el costo de su algoritmo?

Ejercicio 4:

Dados los siguientes vectores, que muestran los costos obtenidos al ejecutar los algoritmos A_1 y A_2 sobre el universo de datos $d_1 \dots d_7$:

- a) $\overline{c}_1 = (5, 34, 65, 12, 25, 21, 9)$ y $\overline{c}_2 = (35, 43, 80, 20, 38, 40, 19)$
- b) $\overline{c}_1 = (5, 34, 65, 12, 25, 21, 9)$ y $\overline{c}_2 = (5, 3, 55, 10, 18, 20, 7)$
- c) $\overline{c}_1 = (35, 47, 65, 11, 28, 21, 19)$ y $\overline{c}_2 = (35, 31, 80, 20, 18, 40, 19)$

Se pide analizar, en cada caso, los vectores de costos y decidir qué algoritmo es el mejor, justificando su respuesta. Si en algún caso no puede decidir sólo con este análisis, explique qué se necesitaría para lograr una decisión adecuada.

Ejercicio 5:

- 1) Explique cuándo es necesario utilizar una *función de evaluación* para comparar dos algoritmos.
- 2) ¿Cualquier función puede ser utilizada como una *función de evaluación*?
- 3) Explique las propiedades que debe cumplir una *función de evaluación*.
- 4) Demostrar que cada una de las siguientes funciones puede utilizarse como una *función de evaluación*.

a) $e(c) = \max_{1 \leq i \leq N} c_i$

b) $e(c) = \sum_{i=1}^N \frac{c_i}{N}$

c) $e(c) = \sum_{i=1}^N p_i \cdot c_i$ donde $p_i \in \mathbb{R}^+$, $0 < p_i \leq 1$ y $\sum_{i=1}^N p_i = 1$

Ejercicio 6:

Las siguientes funciones representan el costo de resolución de un problema, decidir cuáles pertenecen a $O(n^2)$, justificando su respuesta en cada caso.

a) $f_1 = 0,01n^2 + 80$

b) $f_2 = 0,1n^3 + n$

c) $f_3 = 500n$

d) $f_4 = n^{1,9999}$

e) $f_5 = n^2 + 1000n$

Ejercicio 7:

Clasificar cada una de las siguientes funciones como $O(1)$, $O(n)$, $O(n^2)$, $O(\log n)$ o $O(n \log n)$, de acuerdo a cuál sea la opción más adecuada (leer *Ordenamiento de funciones* al final del práctico):

a) $5n + 7$

b) $n^2 - 5n$

c) $5 \log n + 10$

d) $n(\log n + n)$

e) $n(\log n + 1)$

f) $3 + \sin n\pi$

Ejercicio 8:

Decidir si son verdaderas o falsas las siguientes afirmaciones. Justificar su respuesta demostrando lo que afirma:

- a) $3n^2 + 7n + 4$ es $O(n^2)$
- b) n^i es $O(n^j)$ si $i < j$
- c) $n!$ es $O((n+1)!)$
- d) $(n+a)^2$ es $O(n^2)$
- e) $2^{n+1} \in O(2^n)$
- f) $2^{2n} \in O(2^n)$

Ejercicio 9:

Tres algoritmos que resuelven el mismo problema, con entrada de tamaño n , tienen las siguientes funciones de evaluación para sus costos:

Algoritmo A: $4n + 10$

Algoritmo B: $2n + 40$

Algoritmo C: $n^2 + 5$

- a) Ilustrar comparativamente el desempeño de los tres algoritmos, graficando sus funciones de evaluación juntas en un eje de coordenadas.
- b) Determinar, para cada algoritmo, el rango de valores de n para los cuales ese algoritmo se desempeña más eficientemente que los otros dos algoritmos.
- c) Dar la velocidad de crecimiento en notación O para la función que representa el costo de cada uno de los algoritmos.
- d) ¿En qué caso elegiría su algoritmo en función de la velocidad de crecimiento dada en notación O y estaría seguro de que la elección es la correcta?

Ejercicio 10:

Sabiendo que dos algoritmos A_1 y A_2 requieren n^2 días y n^3 segundos respectivamente para resolver un problema de tamaño n .

- a) ¿Cuál de los dos algoritmos elegiría? ¿Por qué?
- b) ¿Existen algunos valores de n para los cuales el algoritmo A_1 tendrá un mejor desempeño que el algoritmo A_2 ? Si es así, diga cuáles son.

De lo analizado, en a) y b) ¿Bajo qué condiciones está seguro que sólo con mirar los órdenes de las funciones que representan los costos, es suficiente para decidir cuál es el mejor algoritmo?



Ejercicio 11:

Suponga que $T_1(n)$ y $T_2(n)$ son tiempos de ejecución de dos trozos de programa P_1 y P_2 tal que $T_1(n) \in O(f)$ y $T_2(n) \in O(g)$. Demostrar que:

- a) **Regla de la suma:** si se ejecuta P_1 seguido de P_2 , su tiempo de ejecución $T_1(n) + T_2(n)$ es $O(\max(f, g))$.
- b) **Regla del producto:** si se ejecutan anidados P_1 y P_2 su tiempo de ejecución $T_1(n) \times T_2(n)$, es $O(f \times g)$.

Ejercicio 12:

Sabiendo que cada sentencia utiliza, para su ejecución, un tiempo T_i encontrar las reglas que permitan calcular el tiempo máximo de ejecución en cada uno de los siguientes casos (tener en cuenta las reglas de la suma y del producto):

- a) Sentencias simples como asignación, lectura, escritura. ¿Hay alguna excepción?
- b) Una secuencia de sentencias.
- c) Una selección **sin ELSE**.
- d) Una selección **con ELSE**.
- e) Una iteración.

Ejercicio 13:

Escribir en forma de algoritmo el método clásico para transformar un número natural n representado en binario a su representación decimal. ¿Cuál es el costo, expresado en notación O , de este algoritmo? Especifique las hipótesis que considere necesarias.

Ejercicio 14:

Dado el siguiente algoritmo:

```
función triangular(var n: entero): entero
var i, j, k, suma: entero;
empezar
  suma := 0;
  para i := 1 hasta n hacer
    para j := 1 hasta i hacer
      para k := 1 hasta j hacer
        suma := suma + (i * j MOD k)
  retornar suma;
fin;
```

- a) Calcular la cantidad exacta de ejecuciones de la instrucción del bloque interno de los **para** anidados ($\text{suma} := \text{suma} + \dots$) en función de n , resolviendo la sumatoria correspondiente.
- b) Determinar la cota del esfuerzo máximo en notación O .

Ejercicio 15:

Expresar en notación O las siguientes recursiones:

- a) $x(n) = x(n-1) + 5$ para $n > 1$, 0 si $n = 1$
- b) $x(n) = 3x(n-1)$ para $n > 1$, 4 si $n = 1$
- c) $x(n) = x(n-1) + n$ para $n > 0$, 0 si $n = 0$
- d) $x(n) = x(n/2) + 1$ para $n > 1$, 1 si $n = 1$

Ejercicio 16:

Obtener en notación O la cota para el esfuerzo *máximo* de cada uno de los siguientes algoritmos, considerando como función de costo el tiempo de ejecución de cada sentencia (usar las reglas planteadas en los **Ejercicios 11 y 12**):

a) procedure **buscar**(var v: array [1..15] of integer, e:integer);
 var i: integer;
 begin
 i := 1;
 while i<15 and e<>v[i] do
 i := i + 1
 if e = v[i] then
 imprimir i
 else
 imprimir "No se encontró"
 end

b) procedure **multmat**(n:integer);
 var i, j, k : integer;
 begin
 for i := 1 to n do
 for j := 1 to n do
 begin
 C[i,j] := 0;
 for k := 1 to n do
 C[i,j] := C[i,j] + A[i,k] * B[k,j];
 end
 end
 end
 end

c) procedure **nada2**(var A: array [0..n-1] of integer);
 var i, j, temp : integer;
 begin
 for i := 1 to 1000 do
 for j := 2000 downto 1 do
 begin
 temp := A[j MOD n];
 A[j MOD n] := A[i MOD n];
 A[i MOD n] := temp;
 end
 end
 end

- d) procedure **muy-impar**(n: integer);
 var i, j, x,y : integer;
 begin
 for i := 1 to n do
 if odd(i) then
 begin
 for j:=i to n do
 x:=x+1;
 for j:=1 to i do
 y:=y+1;
 end
 end
 end
 end
- e) procedure **nada1**(var T: array [1..n] of integer);
 var i, j, minj, minx : integer;
 begin
 for i := 1 to n - 1 do
 begin
 minj := i;
 minx := T[i];
 for j := i + 1 to n do
 begin
 if T[j] < minx then
 begin
 minj := j;
 minx := x;
 end
 T[minj] := T[i];
 T[i] := T[minx];
 end
 end
 end
 end
 end
- f) function **recursiva1**(n: integer): integer;
 begin
 if n <= 1 then
 return(1);
 else
 return (**recursiva1**(n-1) + **recursiva1**(n-1));
 end
- g) function **recursiva2**(n: integer): integer;
 var i, y : integer;
 begin
 if n <= 1 then
 return(n+1);
 else
 begin
 for i:=1 to n do
 y:=y+1;
 return (**recursiva2**(n/2) + **recursiva2**(n/2));
 end
 end
 end

Ejercicio 17:

Obtener la cota superior del el esfuerzo máximo, en notación O , para los dos algoritmos que resuelven "la suma de los primeros n cubos". Utilizar las reglas planteadas en los **Ejercicios 11 y 12**.

- a)** function **N_cubos_rec**(n: entero): entero;
 var i : entero;
 begin
 if n = 1 then
 return(1);
 else
 i:=n;
 return (N_cubos_rec(n-1) +(i * i * i));
 end
- b)** function **N_cubos_iter**(n: entero): entero;
 var i, R : entero;
 begin
 R := 1;
 for i:=2 to n do
 R := R +(i * i * i);
 return(R);
 end

A partir de los resultados obtenidos responda:

- i)** Basándose en el esfuerzo en notación O ¿Prefiere la versión iterativa o la versión recursiva?
- ii)** ¿Qué motivaría que elija una versión por sobre la otra?



Ejercicios Adicionales

Ejercicio 1:

Ordenar las siguientes funciones en orden creciente: $f(n)$ estará antes que $g(n)$ en su lista si y sólo si $f(n) \in O(g(n))$, o sea que la velocidad de crecimiento de $f(n)$ no es mayor que la de $g(n)$:

$n, \sqrt{n}, \log n, \log \log n, \log^2 n, \frac{n}{\log n}, \left(\frac{1}{3}\right)^n, \left(\frac{3}{2}\right)^n, 17, 2^n, n^3 - 100n^2, n^{0.1}, n^2, n + \log n, 1000000.$

Nota: En la última página existe un apéndice con algunas reglas útiles para ordenar funciones por su velocidad de crecimiento.

Ejercicio 2:

Demostrar que:

- a) La relación $R \subseteq \mathcal{F} \times \mathcal{F}$ tal que $f R g \Leftrightarrow f$ es $O(g)$ es transitiva.
- b) $f \in O(g)$ si sólo si $g \in \Omega(f)$.
- c) La notación Θ es reflexiva, simétrica y transitiva.

Ordenamiento de funciones de acuerdo a su velocidad de crecimiento

Potencias

Las potencias de n son ordenadas de acuerdo al exponente: n^a es $O(n^b)$ si y sólo si $a \leq b$.

Logaritmos

El orden de $\log n$ es independiente de la base tomada para los logaritmos; es decir, $\log_a n$ es $O(\log_b n)$ para todo $a, b > 1$.

Un logaritmo crece más lentamente que cualquier potencia positiva de n : $\log n$ es $O(n^a)$ para cualquier $a > 0$, pero n^a nunca es $O(\log n)$ para $a > 0$.

Exponenciales

Cualquier potencia n^a es $O(b^n)$ para todo a y para todo $b > 1$, pero b^n nunca es $O(n^a)$ para $b > 1$.

Si $a < b$, entonces a^n es $O(b^n)$, pero b^n no es $O(a^n)$.

Productos

Si $f(n)$ es $O(g(n))$ y $h(n)$ es cualquier función no cero, entonces la función $f(n) \cdot h(n)$ es $O(g(n) \cdot h(n))$.

Regla de la cadena

Las reglas precedentes pueden ser aplicadas recursivamente (una regla de cadena) sustituyendo una función de n por n .

Por ejemplo: $\log \log n$ es $O((\log n)^{\frac{1}{2}})$, o lo que es lo mismo $O(\sqrt{\log n})$.