Universidad Catolica San Pablo

Ciencias de la Computacion

Lista de Ejercicios

Autor:
Angel Josue Loayza Huarachi

Mayo 2022

Índice

1. Ejercicio 1

Utilizando las reglas de notacion asintoticas, demuestre las expresiones son verdaderas o falsas, justificando con la propia demostracion matematica y tambien con un breve comentario en caso sea necesario. Asuma que toda funcion presentada es positiva.

- $max(f(n), g(n)) = \theta(f(n) + g(n))$

  $$f(n) = max(f(n), g(n))$$
  $$g(n) = max(f(n), g(n))$$

  Entonces

  $$C_1 g(n) \leq f(n) \leq C_2 g(n)$$
  $$C_1 * (f(n) + g(n)) \leq max(f(n), g(n)) \leq C_2 * (f(n) + g(n))$$

  1. Sabemos que $f(n) + g(n) \leq max(f(n), g(n))$.
     Entonces: $max(f(n), g(n)) = O(f(n) + g(n))$
  2. Sabemos que $f(n) \leq f(n) + g(n)$ y que $g(n) \leq f(n) + g(n)$, pues sea cual sea el mayor entre f(n) o g(n), siempre seran menores a la suma de estos 2; por ello, cada uno de ellos tambien se considedra menor.
     Entonces: $max(f(n), g(n)) = \Omega(f(n) + g(n))$ (limite asintotico inferior)

  Pero, para ser verdad, C_1 debe ser ≥ 1 para no volverla negativa o cero. Y C_2 debe ser ≥ 2, para evitar volverlo negativo, cero, y definitivamente mayor a C_1
  Comentario:
  El limite asintotico superior sera verdadero para todo $C_1 \geq 1$.
  El limite asintotico inferior sera verdadero para todo $C_2 \geq 2$.

- $(n + a)^b = \Theta(n^b) | a, b \epsilon R^+, b > 0$

  $$f(n) = (n + a)^b$$
  $$g(n) = n^b$$

  Entonces

  $$C_1 g(n) \leq f(n) \leq C_2 g(n)$$
  $$C_1 * n^b \leq (n + a)^b \leq C_2 * n^b$$

  Delimitamos a:

  $$Si a = 0$$
  $$C_1 * n^b = (n + 0)^b$$
  $$C_1 * n^b = n^b$$

  El cual cumple para $C_1 = 1$. Entonces $a \leq 0$ o $|a|$
  Tomar en cuenta:

  - Si $|a| \leq n$. Entonces $n + a \leq n + |a| \leq 2n$. Pues como n es mator que .ª", su valor absoluto sera mayor o igual. Y de igual forma, el doble de n, siempre sera mayor que la suma de estos.

- Si $|a| \leq \frac{1}{2}$. Entonces $n + a \geq n - |a| \geq \frac{1}{2}n$. Pues como n es mayor que -a-, su valor absoluto sera mayor o igual. Y de igual forma, el doble de n, siempre sera mayor que la suma de estos.

Entonces tenemos:

- Inecuaciones:
$$n + a \leq n + |a| \leq 2n$$
$$n + a \geq n - |a| \geq \frac{1}{2}n$$
Entonces:
$$\frac{1}{2}n \leq n + a \leq 2n$$
$$(\frac{1}{2}n)^b \leq (n + a)^b \leq (2n)^b$$
$$(\frac{1}{2})^b (n)^b \leq (n + a)^b \leq (2)^b (n)^b$$

- Condicionales:
$$Si\, |a| \leq n$$
$$Si\, |a| \leq \frac{1}{2}n \to 2\, |a| \leq n$$

Comentario:
Para $C_1 = (\frac{1}{2})^b$, $C_2 = (2)^b$ y $n_0 = 2\,|a|$, satisface la ecuacion.

- $2^{n+1} = O(2^n)$
$$f(n) = (2)^{n+1}$$
$$g(n) = 2^n$$
Entonces
$$f(n) \leq Cg(n)$$
$$(2)^{n+1} \leq C2^n$$
$$\frac{(2)^{n+1}}{2^n} \leq \frac{C2^n}{2^n}$$
$$\frac{(2)^{n+1}}{2^n} \leq C$$
$$(2)^{(n+1)-n} \leq C$$
$$(2)^1 \leq C$$
$$2 \leq C$$
Tomamos valores C = 3 y n = 0:
$$(2)^{n+1} \leq C2^n$$
$$(2)^1 \leq 3 * 2^0$$
$$2 \leq 3 \to Verdad$$
Comentario:
Satisface la ecuacion cuando $C = 3$ y $n_0 = 0$, $\forall n \geq n_0$

- $2^{2n} = O(2^n)$

$$f(n) = (2)^{2n}$$

$$g(n) = 2^n$$

Entonces

$$f(n) \leq Cg(n)$$

$$(2)^{2n} \leq C2^n$$

$$\frac{(2)^{2n}}{2^n} \leq C$$

$$2^{2n-n} \leq C$$

$$2^n \leq C$$

Tomamos valores n = 1:

$$2^n \leq C$$

$$2 \leq C$$

Tomamos valores n = 2:

$$2^2 \leq C$$

$$4 \leq C$$

Tomamos valores n = 3:

$$2^3 \leq C$$

$$8 \leq C$$

Comentario:
Nunca una constante sera mayor a una funcion exponencial, por lo que 2^{2n} no tendria un limite asintotico superior en 2^2

- $\log 2 \log_2 n = O(\log \log n)$

- Probar por induccion

$$\sum_{i=1}^{n} \frac{1}{i^2} \leq 2 - \frac{1}{n}$$

$$\frac{1}{1} + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + ... + \frac{1}{n^2} \leq 2 - \frac{1}{n}$$

1. Paso Base: Probamos con cualquier numero

$$P(n_0) = P(1)$$

$$n_0 = 1$$

$$P(1) = 2 - \frac{1}{1}$$

$$P(1) = 2 - 1$$

$$P(1) = 1$$

La formula es verdadera para n_0

2. Paso inductivo:
   Si la formula es verdadera
   Entonces tambien es verdadera para n = k y para n = k+1.
   Entonces:
   $$P(k) = P(k+1)$$
   Hipotesis Inductiva
   $$P(k) = \sum_{i=1}^{k} \frac{1}{i^2} \leq 2 - \frac{1}{k}$$
   Tenemos que demostrar
   $$P(k+1) = \sum_{i=1}^{k+1} \frac{1}{i^2} \leq 2 - \frac{1}{k+1}$$
   $$\begin{aligned}\sum_{i=1}^{k+1} &= \sum_{i=1}^{k} \frac{1}{i^2} + \frac{1}{k+1} \\ &= 2 - \frac{1}{n} + \frac{1}{k+1} \\ &= \frac{2n-1}{n} + \frac{1}{k+1} \\ &= \frac{(2n-1)(k+1)+n}{n(k+1)} \\ &= \frac{(2nk+3n-k-1)}{n(k+1)}\end{aligned}$$
   Comentario: Por lo tanto no fue posible derivar la conclusion de la hipotesis. Esto significa que el predicado original es falso

- Probar por induccion

$$[\frac{n}{2}] \begin{cases} \frac{n}{2}, \text{ si n es par} \\ \frac{n+1}{2}, \text{ si n es impar} \end{cases}$$

1. Paso Base (a) - Cuando n es par
   $$P(n_0) = P(2)$$
   $$n_0 = 2$$
   $$P(2) = \frac{2}{2}$$
   $P(2) = 1$ La formula es verdadera para n_0

2. Paso Inductivo (a) - Cuando n es par
   Si la formula es verdadera
   Entonces tambien es verdadera para n = k y para n = k+1.
   Entonces:
   $$P(k) = P(k+1)$$

Hipotesis Inductiva:

$$P(k) = \frac{k}{2} = \frac{k}{2}$$

Tenemos que demostrar

$$P(k+1) = \frac{k+1}{2} = \frac{k+1}{2}$$

$$\frac{k+1}{2} = \frac{k+1}{2}$$

$$k = k, \text{ Si es verdad para un n par}$$

3. Paso Base (b) - Cuando n es impar

$$P(n_0) = P(1)$$

$$n_0 = 1$$

$$P(1) = \frac{1+1}{2}$$

$$P(1) = 1$$

4. Paso Inductivo (b) - Cuando n es impar
   Si la formula es verdadera
   Entonces tambien es verdadera para n = k y para n = k+1.
   Entonces:

$$P(k) = P(k+1)$$

   Hipotesis Inductiva:

$$P(k) = \frac{k}{2} = \frac{k+1}{2}$$

   Tenemos que demostrar

$$P(k+1) = \frac{k+1}{2} = \frac{(k+1)+1}{2}$$

$$\frac{k+1}{2} = \frac{k+2}{2}$$

$$2 \neq 4, \text{ No es verdad para un n impar}$$

- Probar por induccion

$$\forall n \geq 0, n^5 - n \text{ es divisible por } 5$$

1. Paso Base

$$P(n_0) = P(1)$$

$$n_0 = 1$$

$$P(1) = 1^5 - 1$$

$$P(1) = 0 \text{ La formula es verdad para } n_0$$

2. Paso Inductivo
Asimilamos que es verdad
Tiene que ser verdadera para n = k+1.
Entonces:

$$P(k) = P(k+1)$$

Hipotesis Inductiva:

$$P(k) = k^5 - k \text{ es divisible por } 5$$

Demostramos

$$P(k+1) = (k+1)^5 - k - k + 1$$

$$= k^5 + (5k)^4 + (10k)^3 + (10k)^2 + 5k + 1 - k + 1$$
$$= k^5 + (5k)^4 + (10k)^3 + (10k)^2 + 4k + 2$$
$$= k^5 + 5((k)^4 + (2k)^3 + (2k)^2) + 4k + 2$$

Como multiplica por 5, entonces si es divisible por 5

2. Ejercicio 2

La recurrencia $T(n) = 7T(\frac{n}{2}) + n^2$ representa la funcion de complejidad del algoritmo A. Un algoritmo alternativo A' para el mismo problema tiene la siguiente ecuacion $T'(n) = aT'(\frac{n}{4}) + n^2$. Cual es el mayor numero entero a, que hace que el algoritmo A' sea asintoticamente mas rapido que A.

- Caso de T(n) = $7T\frac{n}{2} + n^2$

$$T(n) = 7T\frac{n}{2} + n^2 \to T(\frac{n}{2})$$

$$T(n) = 7(7T(\frac{\frac{n}{2}}{2}) + (\frac{n}{2})^2) + n^2$$

$$T(n) = 7^2T(\frac{n}{2^2}) + 7(\frac{n}{2})^2) + n^2 \to T(\frac{n}{2^2})$$

$$T(n) = 7^2(7T(\frac{\frac{n}{2^2}}{2}) + (\frac{n}{2^2})^2) + 7(\frac{n}{2})^2) + n^2$$

$$T(n) = 7^3T(\frac{n}{2^3}) + 7^2(\frac{n}{2^2})^2 + 7(\frac{n}{2})^2 + n^2$$

Entonces:

$$T(n) < T(\frac{n}{2^{i-1}}) >= 7^iT(\frac{n}{2^i}) + 7^{i-1}(\frac{n}{2^{i-1}})^2 + ... + 7(\frac{n}{2})^2 + n^2$$

- Caso de T'(n) = $aT'(\frac{n}{4}) + n^2$

$$T'(n) = aT'(\frac{n}{4}) + n^2 \to T'(\frac{n}{4})$$

$$T'(n) = a(aT'(\frac{\frac{n}{4}}{4}) + (\frac{n}{4})^2) + n^2$$

$$T'(n) = a^2T'(\frac{n}{4^2}) + a(\frac{n}{4})^2) + n^2 \to T(\frac{n}{4^2})$$

$$T'(n) = a^2(aT'(\frac{\frac{n}{4^2}}{4}) + (\frac{n}{4^4})^2) + a(\frac{n}{4})^2 + n^2$$

$$T'(n) = a^3T'(\frac{n}{4^3}) + a^2(\frac{n}{4^2})^2 + a(\frac{n}{4})^2 + n^2$$

Entonces:

$$T'(n) < T'(\frac{n}{4^{i-1}}) >= a^iT(\frac{n}{4^i}) + a^{i-1}(\frac{n}{4^{i-1}})^2 + ... + a(\frac{n}{4})^2 + n^2$$

Comentario: El valor del entero –a– para que el algoritmo A' sea asintoticamente mas rapido que A es, el mismo al cuadrado (a^2), pues::

$$A = 7^iT(\frac{n}{2^i}) + ...$$

$$A' = a^iT(\frac{n}{4^i}) + ... \to a^iT(\frac{n}{(2^2)^i}) + ...$$

3. Ejercicio 3

Dada las siguientes ecuaciones de recurrencia, resolver utilizando expansion de recurrencia y teorema maestro. En ambos casos colocar todo el desarrollo no obviar partes. Si en caso no se pueda aplicar el teorema maestro, explique el motivo. Asuma que T(n) es constante para $n < 2$.

- $T(n) = 4T(\frac{n}{2}) + n$

$$a = 4$$

$$b = 2$$

$$f(n) = n$$

De tal forma que :

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

Caso 1 Teorema Maestro

$$f(n) = O(n^{\log_b a - \varepsilon})$$

$$n = O(n^{2-\varepsilon}), \text{ para e} = 1$$

$$n = O(n)$$

Entonces:

$$T(n) = O(n^2)$$

- $T(n) = 4T(\frac{n}{2}) + n^2$

$$a = 4$$

$$b = 2$$

$$f(n) = n^2$$

De tal forma que :

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

Caso 2 Teorema Maestro

$$f(n) = \Theta(n^{\log_b a})$$

$$n^2 = \Theta(n^2)$$

Entonces:

$$T(n) = \Theta(n^{\log_b a} * \log n)$$

$$T(n) = \Theta(n^2 * \log n)$$

- $T(n) = 4T(\frac{n}{2}) + n^3$

$$a = 4$$

$$b = 2$$

$$f(n) = n^3$$

De tal forma que :

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

Caso 3 Teorema Maestro

$$af(\frac{n}{b}) < cf(n)$$

$$4(\frac{n}{2})^3 < cf(n)$$

$$2n^3 < cn^3$$

$$2 < c, \text{ Por lo que C} = 3 \text{ cumple}$$

Entonces:

$$F(n) = \Omega(n^{\log_b a+\varepsilon}), \text{ para e} = 1$$

$$T(n) = \Theta(n^3)$$

- $T(n) = 4T(\frac{9n}{10}) + n$

$$a = 1$$

$$b = 10/9$$

$$f(n) = n$$

De tal forma que :

$$n^{\log_b a} = n^{\log_{10/9} 1} = n^0 = 1$$

Caso 3 Teorema Maestro

$$af(\frac{n}{b}) < cf(n)$$

$$1(\frac{9n}{10}) < cn$$

$$\frac{9n}{10} < cn$$

$$\frac{9}{10} < c, \text{ Por lo que C} = 1 \text{ cumple}$$

Entonces:

$$F(n) = \Omega(n^{\log_b a+\varepsilon}), \text{ para e} = 1$$

$$T(n) = \Theta(n)$$

- $T(n) = 16T(\frac{n}{4}) + n^2$

$$a = 16$$
$$b = 4$$
$$f(n) = n^2$$

De tal forma que :

$$n^{\log_b a} = n^{\log_4 16} = n^{\sqrt[1]{2}} = \sqrt{n}$$

Caso 3 Teorema Maestro

$$af(\frac{n}{b}) < cf(n)$$
$$16(\frac{n}{4}) < cn^2$$
$$c = 2, \text{ cumple}$$

Entonces:

$$F(n) = \Omega(n^{\log_b a+\varepsilon}), \text{ para e} = 1$$
$$T(n) = \Theta(n^2)$$

- $T(n) = 7T(\frac{n}{3}) + n^2$

$$a = 7$$
$$b = 3$$
$$f(n) = n^2$$

De tal forma que :

$$n^{\log_b a} = n^{\log_3 7} = n^{1{,}777}$$

Caso 3 Teorema Maestro

$$af(\frac{n}{b}) < cf(n)$$
$$7(\frac{n}{3}) < cn^2$$
$$c = 1, \text{ no cumple}$$
$$c = 2, \text{ si cumple}$$

Entonces:

$$F(n) = \Omega(n^{\log_b a+\varepsilon}), \text{ para e} = 0.222$$
$$T(n) = \Theta(n^2)$$

- $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$

$$a = 2$$
$$b = 4$$
$$f(n) = \sqrt{n}$$

De tal forma que :

$$n^{\log_b a} = n^{\log_4 2} = n^{\frac{1}{2}} = \sqrt{n}$$

Caso 2 Teorema Maestro

$$f(n) = \Theta(n^{\log_2 4})$$

$$\sqrt{n} = \Theta(\sqrt{n})$$

Entonces:

$$T(n) = \Theta(n^{\log_2 4} * \log n)$$
$$T(n) = \Theta(\sqrt{n} * \log n)$$

- $T(n) = 4T(\frac{n}{2}) + n^2 \log n$

$$a = 4$$

$$b = 2$$

$$f(n) = n^2 * \log n$$

De tal forma que :

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

Caso 1 Teorema Maestro

$$f(n) = O(n^{\log_b a - \varepsilon})$$

$$n^2 * \log n = O(n^{2-\varepsilon}), \text{ para e } \approx 1$$

Entonces:

$$T(n) = O(n^2 * \log n)$$

- $T(n) = T(n-1) + n$

$$T(n) = T(n-1) + n \rightarrow T(n-1)$$

$$T(n) = (T(n-2) + n) + n$$
$$T(n) = T(n-2) + 2n \rightarrow T(n-2)$$
$$T(n) = (T(n-3) + n) + 2n$$
$$T(n) = T(n-3) + 3n \rightarrow T(n-3)$$
$$...$$
$$T(n) = T(n-(n-1)) + (n-1)n$$
$$T(n) = T(1) + n^2 - n$$
$$T(n) = C + n^2 - n$$
$$T(n) = O(O(c) + O(n^2)) + (-n)$$
$$T(n) = O(c + n^2 + n)$$
$$T(n) = O(n^2)$$

- $T(n) = \sqrt{n} + 1$

$$T(n) = n^{\frac{1}{2}} + n$$

$$T(n) = O(O(n^{\frac{1}{2}}) + O(n))$$

$$T(n) = O(n^{\frac{1}{2}} + n)$$

$$T(n) = O(n)$$

4. Ejercicio 4

Se sabe que, para resolver el problema de ordenacion de numeros, el algoritmo Mergesort posee una complejidad de $\theta(n \log n)$ y el Insertion Sort $O(n^2)$. Sin embargo, los factores constantes del algoritmo Insertion lo tornan mas veloz para vectores de tama˜no n peque˜nos. Por ende tiene sentido realizar una combinacion de ambos, y utilizar el algoritmo de insercion cuando los problemas se tornen lo suficientemente peque˜nos. Considere la siguiente modificacion del Mergesort: $\frac{n}{k}$ sub listas de tama˜no k son ordenadas utilizando el algoritmo de insercion, y luego combinadas utilizando el mecanismo del Mergesort (Merge), siendo k la variable a ser determinada.

Listing 1: InsertionSort.hpp

```
void insertionSort(int arr[], int n)
{
        int a;
        int aux;
        int j;
        for (a = 1; a < n; a++)
        {
                aux = arr[a];
                j = a - 1;
                while (j >= 0 and arr[j] > aux)
                {
                        arr[j + 1] = arr[j];
                        j--;
                }
                arr[j + 1] = aux;
        }
}
```

Listing 2: InsertionSort.hpp

```
void mergeSort(int* A, int n)
{
        //base
        if (n == 1)
        {
                return;
        }
        //recurisivo
        int centro = n / 2;
        int* aux1izq = new int[centro];
        int* aux2der = new int[n - centro];

        //recorremos la primera parte
        for (int i = 0; i < centro; i++)
        {
                aux1izq[i] = A[i];
```

```
        }
        //recorremos la otra parte
        for (int i = centro; i < n; i++)
        {
                aux2der[i-centro] = A[i];
        }
        mergeSort(aux1izq, centro); //evaluacion
            recursivamente la primera mitad
        mergeSort(aux2der, n - centro); //evaluacion
            recursiva de la segunda mitad
    mergeaux(A, aux1izq, centro, aux2der); //obtenido de
        la tarea "Algoritmos de ordenacion"
}
```

5. Ejercicio 5

Para los problemas presentados, cree un codigo en $C++$, presente la complejidad del algoritmo del cual es instancia el codigo presentado [1pt]. Cada codigo debera estar en latex es decir deben utilizar la biblioteca listings a b. Adicionalmente, deberan correr y colocar screenshots del codigo corriendo con las instancias que se les pida por cada problema.

- Desarrolle un programa "Recursivo" para generar la descomposici´on de un n´umero entero positivo, en la suma de todos los posibles factores. La presentaci´on de los factores debe estar ordenada de mayor a menor. Por ejemplo para n = 5

Listing 3: DescompositionNumber.hpp - Sum decomposition algorithm

```
#include <iostream>
#include <vector>
#include <stdio.h>
#pragma once

using namespace std;

/* Example
5
4 1
3 2
3 1 1
2 2 1
2 1 1 1
1 1 1 1 1
*/

void printVector(vector <int> v)
{
        for (auto i = v.begin(); i != v.end(); ++i)
```

```
                cout << *i << " ";
}

void printarr(int *A, int i)
{
        //for (int j = i; j < sizeof(A)/ sizeof(A[0])
            ; j++)
        for (int j = i; j > 0; j--)
        //for (int j = 0; j < i; j++)
        {
                cout << A[j-1] << " ";
        }
        cout << endl;
}

void descompositionNumber(int n, int i, int tam)
{
        static int* arr = new int[tam];
        if (n == 0)
        {
                printarr(arr, i);
        }

        if (n > 0)
        {
                int j;
                for (j = tam; j > 0; j--)
                {
                        arr[i] = j;
                        descompositionNumber(n - 1, i
                            + 1, j);
                }
        }
}
```

- Dado un vector con n numeros enteros, determine la maxima suma en un subvector contiguo de ese vector. Si todos los n´umeros fueran negativos asuma que la suma es 0.

Listing 4: maxSum.hpp - Maximum sum in a vector

```
#include <iostream>
#pragma once

using namespace std;

int maxSum(int A[], int n)
{
        int aux1 = 0;                    //C
        int aux2 = 0;                    //C
```

```
int sum1 = 0;                          //C
int result = 0; // para la respuesta final //
    C

//maxima suma entre todos los numeros
for (int i = 0; i < n; i++)            //N
{
        sum1 = sum1 + A[i];            //N
}

int aux3 = sum1;                       //C
int sum2 = sum1;                       //C

// COMPROBACION DE IZQUIERDA A DERECHA
for (int i = 0; i < n; i++)            //N
{
        //vamos restando a la suma total los
            elementos del vector
        aux3 = aux3 - A[i];            //N
        if (aux3 > sum1)               //N
        {
                //reemplazamos sum1 porque
                    encontramos un numero
                    mayor
                sum1 = aux3;           //N
                // marcamos donde encontramos
                     un mayor

                aux1 = i;              //N
        }
}

int aux4 = sum2;                       //C

// COMPROBACION DE DERECHA A IZQUIERDA

for (int i = n - 1; i >= 0; i--)       //N
{
        aux4 = aux4 - A[i];            //N
        if (aux4 > sum2)               //N
        {
                sum2 = aux4;           //N
                // marcamos donde encontramos
                     un mayor
                aux2 = i;              //N
        }
}

for (int i = aux1 + 1; i <= aux2 - 1; i++)
```

```
            //N
        {
                result = result + A[i];             //N
        }

        return result;

}
/* Complejidad
O( 7 O(C) + 14 O(N) )
O( O(C) + O(N) )
O( O(N) )
complejidad de O(N)
*/
```

Figura 1: Sum decomposition algorithm and Maximum sum in a vector