Universidad de Concepción

Facultad de ingeniería
Departamento de ingeniería informática y Ciencias de la computación

Fundamentos Estructura de Datos y Algoritmos

Análisis de Algoritmos

Roberto Ávila.

Lisette Morales.

Profesora: Cecilia Hernandez

22 de Abril 2019.

Solución problema 1

Una función f(n) es O(g(n)) si existen constantes n0 $\geq$ 0 y c > 0 tal que f(n) c $\times$ g(n) para n $\geq$ n0. Proporcione un análisis O() para cada una de las siguientes funciones. En cada caso, muestre los valores de las constantes n0 y c que hacen cierta su afirmación.

a)

$$f(n) = n^2 + \sqrt[2]{n} - 2$$
$$0 \leq f(n) \leq C * g(n)$$
$$0 \leq n^2 + \sqrt[2]{n} - 2 \leq C * n^2$$

probando para $n = 4$ y $C \geq 0$, se obtiene que:

$$0 \leq 16 + 2 - 2 \leq C * 16$$
$$1 \leq C$$

b)

$$f(n) = 5log_3(log(log(n)/20))$$
$$0 \leq f(n) \leq C * g(n)$$
$$0 \leq 5log_3(log(log(n)/20)) \leq C * logloglog(n)$$
$$0 \leq 5log_3(loglog(n) - log(20)) \leq C * logloglog(n)$$
$$0 \leq 5log_3(\frac{loglog(n)}{log(20)}) \leq C * logloglog(n)$$
$$0 \leq 5(\frac{logloglog(n)}{log_3log(20)}) \leq C * logloglog(n)$$
$$0 \leq 5(\frac{logloglog(n)}{log_3log(20)}) \leq C * logloglog(n)$$

probando para $n = 16$ y $C > 0$, se obtiene que:

$$0 \leq \frac{5}{1{,}3} \leq C * 1$$
$$\approx 3{,}8 \leq C$$

c)

$$f(n) = 5n + 3log(n^2)$$
$$0 \leq f(n) \leq C * g(n)$$
$$0 \leq 5n + 3log(n^2) \leq C * n$$

probando para $n = 2$ y $C \geq 0$, se obtiene que:

$$0 \leq 10 + 6 \leq C * 2$$
$$8 \leq C$$

d)

Pruebe formalmente que O() es una relación transitiva. Esto es si f(n)$\leq$ O(g(n)) g(n) $\leq$ O(h(n)), entonces f(n) $\leq$ O(h(n))

$$0 \leq f(n) \leq C * g(n) \wedge 0 \leq g(n) \leq C * h(n)$$
$$f(n) - C * g(n) \leq 0 \wedge g(n) - C * h(n) \leq 0$$

haciendo $C = 1$, se obtiene que:

$$f(n) - g(n) + g(n) - h(n) \leq 0$$
$$f(n) - h(n) \leq 0$$
$$f(n) \leq O(h(n))$$

Solución problema 2

Determine si las siguientes afirmaciones son verdaderas o falsas. Justifique su respuesta.

a)

$$log(n!) \text{ es } O(log(n)). \text{ Falso}$$

La demostración que se entrega a continuación, fue analizado con el método del limite, en efecto:

$$\lim_{n\to\infty} \frac{log(n!)}{log(n)} = \lim_{n\to\infty} \frac{log(n(n-1)(n-2))}{log(n)} = \lim_{n\to\infty} \frac{log(\frac{(n-1)(n-2)+(n(n-2))+n(n-1))}{n(n-1)(n-2)}}{\frac{1}{n}\log_2 e} = \infty$$

b)

$$4n^3 \text{ es } \omega(n^3). \text{ Verdadero}$$

La demostración que se entrega a continuación, fue analizado con un para todo C< 4, en efecto:

$$0 \leq C * g(n) < f(n)$$
$$0 \leq C * n^3 < 4n^3$$
$$\text{probando para } n = 1 \text{ y } C > 0, \text{ se obtiene que:}$$
$$0 \leq C * 1 < 4$$
$$C < 4$$

c)

$$\sqrt{2}^{log(n)} \text{ es } \theta(\sqrt{n}). \text{ Verdadero}$$

La demostración que se entrega a continuación, fue analizado con el método del limite, en efecto:

$\lim_{n\to\infty} \frac{\sqrt{2}^{log(n)}}{\sqrt{n}} = \lim_{n\to\infty} \frac{\frac{1}{2\sqrt{n}}}{\frac{1}{2\sqrt{n}}} = 1$

d)

$$\sqrt{n} \text{ - } 3log(n^{10)} \text{ es } \Omega(n). \text{ Falso}$$

La demostración que se entrega a continuación, fue analizado con el método del limite, en efecto:

$\lim_{n\to\infty} \frac{\sqrt{n}-3log(n^{10)}}{n} = \lim_{n\to\infty} \frac{\frac{\sqrt{n}*log(2)-60}{n*log(4)}}{1} = 0$

Solución problema 3

Determine qué realiza el siguiente segmento de código y pruebe su correctitud.

```
int y=0, i=0;
while ( i <= n ) {
   y += 2^i;
   i++;
}
cout<< "y = "<< y <<endl;
```

```
Iteraciones:
Para n=0, y=1
para n=1, y=3
para n=2, y=7
para n=3, y=15 ...
```

Loop invariante: El loop se encuentra cuando al seguir la secuencia 1, 3, 7, 15 nos damos cuenta de que el patrón es $2^n * 2$ - 1, por método de inducción matemática, se obtiene que:

Caso base: $2^0 * 2 - 1 = 1$. se cumple, por lo tanto es verdadero. Hipótesis de inducción: $2^k * 2 - 1$ es verdadero Tesis: $2^k + 1 * 2 - 1 = 2^k * 2 * 2 - 1$ =2 * (2^k * 2) -1

$$= 2 * \underbrace{2^k * 2 - 1}_{hipótesis de inducción} \tag{1}$$

La hipótesis de inducción esta multiplicada por una constante por ende, queda demostrado que para el caso k+1 también es verdadero.

Solución problema 4

Proporcione un análisis asintótico de peor caso en notación O() para el tiempo de ejecución de los siguientes fragmentos de programa.

a)

```
int x = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0;j <n*n/3; j++) {
                x += j; } }
```

Para este código en notación O(n) en el peor caso es $O(n^3)$ Demostración:
$\sum_{i=0}^{n} \sum_{j=0}^{n*n/3}(i) = O(\mathrm{n}^3)$

b)

```
int x = 0;
if (n % 3 == 0) {
      i=0;
    while(i < n*n*n){
        x++;
        i++; }
}
else {
    for(int i=0; i<n*n*n; i++) {
        x++;}
}
```

Para este código en notación O(n) en el peor caso es $O(n^3)$
Demostración: $1/3 \sum_{i=0}^{n*n*n}(i) + 2/3 \sum_{i=0}^{n*n*n}(i) = O(\mathrm{n}^3)$

c)

```
int recurse(int n) {
    for (i = 0; i < n*n; i += 2) {
        procesar(i); // O(1) }
    if (n <= 0)
        return 1;
    else
        return recurse(n-3);}
```

Para este código en notación O(n) en el peor caso es $O(n^2)$
Demostración: $\sum_{i=0}^{n*n}(i) = O(\mathrm{n}^2)$

d)

```
int i=n;
int j,k;

    while (i > 1){
        j = i;
        while(j < n){
            k = 0;
            while (k < n){
                k = k + 2;
            }
            j = j*2;
        }
        i = i/2;
}
```

Para este código en notación O(n) en el peor caso es O($n * log^2(n)$)
Demostración: $\sum_{i=n}^{log(n)} * \sum_{j=i}^{log(n)} * \sum_{k=0}^{n} = O(\text{n*log}^2(n))$

Solución problema 5

Considere un arreglo ordenado de números enteros A de tamaño n y un valor entero x ingresado por teclado. Se desea encontrar las posiciones y valores de un par de elementos (A[i],A[j]) en el arreglo A tal que la suma A[i] + A[j] = x, si es que existen. Solo necesita reportar el primer par encontrado. En caso de no encontrarse tal par de elementos, el algoritmo debe retornar que no existen.

a) O(n^2)

Escriba un pseudo código para un algoritmo O(n2) que resuelva el problema.

```
busqueda1(int x , vector A, int m)
for i=0 to lenght [A] do
    for j=0 to lenght [A] do
        if A[i]+A[j]== x then
            pos.first ← i;
            pos.second ← j;
           return pos;
        end if
    end for
end for
pos.first ← -1;
pos.second ← -1;
return pos;
```

b)O($nlog(n)$)

Ahora diseñe un algoritmo O(n log(n)) que resuelva el problema y escriba su pseudo código.

```
busqueda2 (int x , vector A , int m )
for i=0 to lenght [A] do
    resto ← x-A[i];
    while inf <= sup  do
        mid ← (inf + sup)/2
        if A[mid] == resto and i ! = mid  then
            pos.first ← i;
            pos.second ← mid;
            return pos;
        end if
        if resto < A[mid] then
            sup ← mid-1;
        else
            inf ← mid+1;
        end if
    end while
end for
pos.first ← -1;
pos.second ← -1;
return pos;
```

c)O(n)

Ahora diseñe un algoritmo O(n) que resuelva el problema y escriba su pseudo código.

```
busqueda3(int x , vector A , int m )
while  inf < sup do
    if A[inf] + A[sup] == x  then
        pos.first ← inf;
        pos.second ← sup;
        return pos;
        if A[inf]+A[sup] < x then
            inf++;
        else
            sup - -;
        end if
    end if
end while
pos.first ← -1;
pos.second ← -1;
return pos;
```

d) Gráficos

Implemente los tres algoritmos usando C++ definiendo una función para cada algoritmo. Además considere dos tipos de entrada posible. Un tipo donde los elementos del arreglo son elegidos aleatoriamente y un tipo de entrada que corresponda al peor caso de cada algoritmo. Note que para el tipo aleatorio debe ordenar el arreglo antes de aplicar los algoritmos que solucionan el problema propuesto. Construya un gráfico que muestre como varía el tiempo de ejecución en nanosegundos variando el tamaño de la entrada (n). Para ello considere diversos valores para n. Puede utilizar potencias de 2, osea 256, 512, 1024, 2048, 4096, 8192. Solo debe medir el tiempo que corresponde a la ejecución de la función.

Elementos del arreglo aleatorios

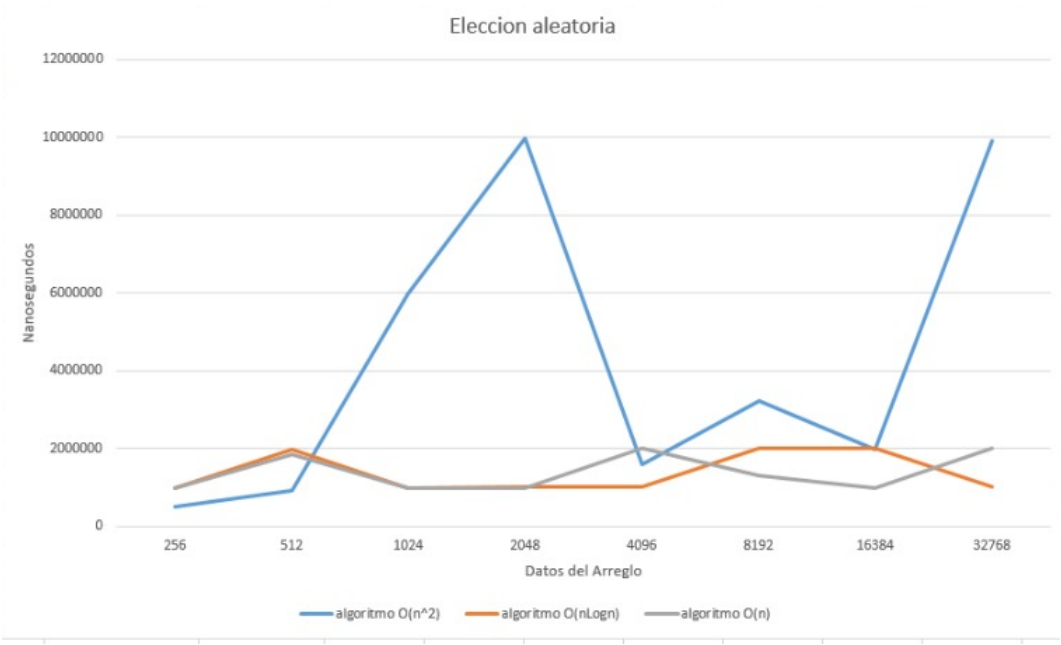

Figura 1: Representación del comportamiento de los algoritmos implementados cuando la entrada es aleatoria

Elementos del peor caso

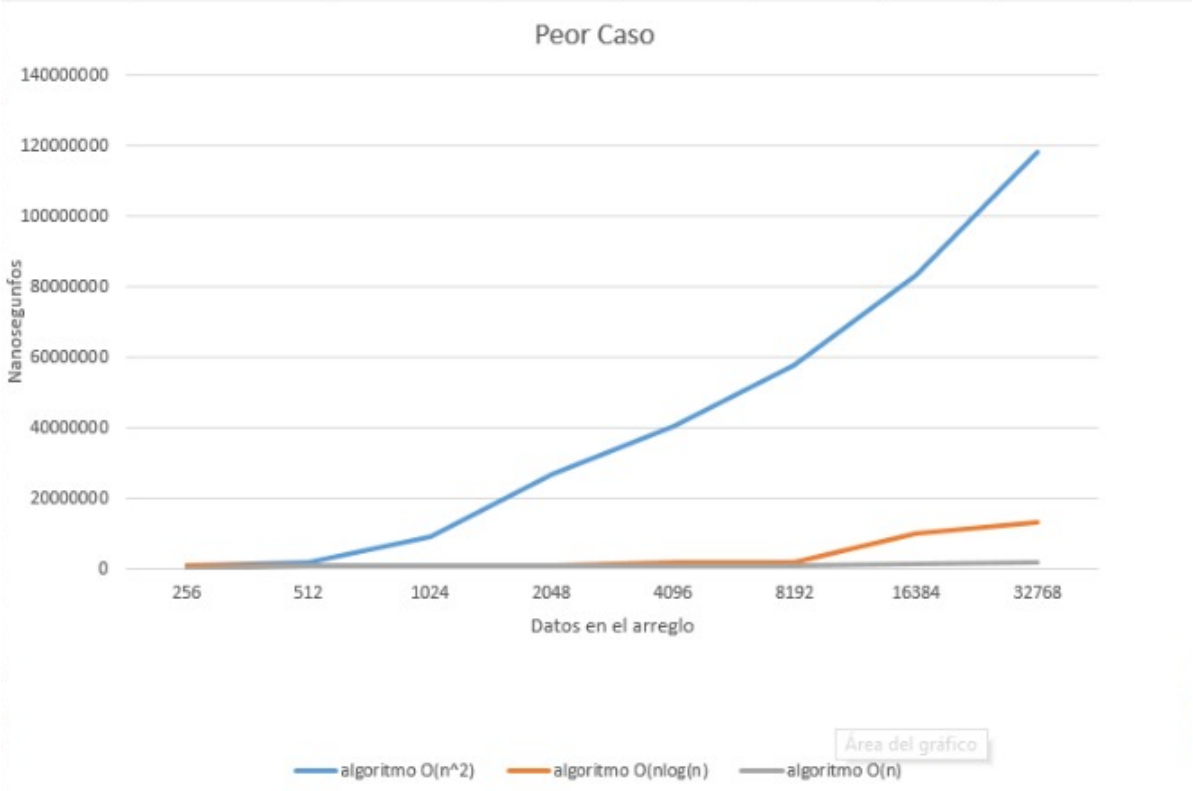

Figura 2: Representación del comportamiento de los algoritmos implementados cuando se considera el peor caso, es decir, cuando el elemento no se encuentra en el arreglo