Juan Sebastián Garizao Puerto

8977555

Ingeniería de Sistemas y Computación

15/02/2023

Punto 1

```
void algoritmo1(int n){
  1 int i, 1 j = 1;  ⟶ 2
    for(i = n * n; i > 0; i = i / 2){ ⟶ 2 log_2 n + 2
        int suma = i + j; 2 log_2 n + 1
        printf("Suma %d\n", suma); 2 log_2 n + 1
        ++j; 2 log_2 n + 1
    }
}
```

a.) $n \cdot n = 2^x$

$\log_2 n^2 = \log_2 2^x$

$2 \log_2 n = x$

$T(n) = 2 + 2 \log_2 n + 2 + 3(2 \log_2 n + 1)$

$= 4 + 2 \log_2 n + 6 \log_2 n + 3$

$= 8 \log_2 n + 7$

$T(n) = O(\log_2 n)$

b.) Lo que se obtiene con $n = 8$

Mensajes impresos

1.) $j = 1$

$i = n \cdot n = 8 \cdot 8 = 64$

"Suma 65"

2.) $j = 2$

$i = 64 \div 2 = 32$

"Suma 34"

3.) $j = 3$

$i = 32 \div 2 = 16$

"Suma 19"

4.) j=4
$i = 16 \div 2 = 8$ "Suma 12"

5.) j=5 "Suma 9"
$i = 8 \div 2 = 4$

6.) j=6 "Suma 8"
$i = 4 \div 2 = 2$

7.) j=7 "Suma 8"
$i = 2 \div 2 = 1$

8.) j=7
$i = 1 \div 2 = 0$

Se acaba el ciclo

Punto 2

```
int algoritmo2(int n){
    int res = 1, i, j;
    for(i = 1; i <= 2 * n; i += 4)
        for(j = 1; j * j <= n; j++)
            res += 2;

    return res;
}
```

int res = 1, i, j; $\longrightarrow 3$

for(i = 1; i <= 2 * n; i += 4) $\frac{n}{2} + 1$ o $\frac{2n+1}{4} + 1 \longrightarrow$ operación techo

for(j = 1; j * j <= n; j++) $\left(\frac{n}{2}\right)\left(\sqrt{n}\right) \longrightarrow$ operación techo

res += 2; $\left(\frac{n}{2}\right)\left(\sqrt{n}\right) - \left(\frac{n}{2}\right)$ o $\left(\frac{2n+1}{4}\right)\left(\sqrt{n}\right) - \left(\frac{2n+1}{4}\right) \longrightarrow$ operación techo

return res; $\longrightarrow 1$

$$1 + 4x = 2n$$
$$4x = 2n - 1$$

Para impares
$$x = \frac{2n+1}{4}$$

$$x^2 = n$$
$$x^2 = n$$
$$x = \sqrt{n}$$

Para pares $\frac{2n}{4} = \frac{n}{2}$

$$t(n) = \left(\frac{n}{2} \cdot \sqrt{n}\right) = \frac{\sqrt{n^3}}{2} = \frac{n^{\frac{3}{2}}}{2}$$

$$O(n^{\frac{3}{2}})$$

Entra al ciclo principal

Evaluaciones

res = 1
n = 8
i = 1

$1 <= 16$ ✓

Entra al ciclo
j = ~~1~~ ~~4~~ 3
res = ~~1~~ 5

$1 <= 8$ ✓
$4 <= 8$ ✓
$9 <= 8$ X

Sale
i = 5

$5 <= 16$ ✓

Entra
j = ~~1~~ ~~4~~ 3
res = ~~5~~ 9

$1 <= 8$ ✓
$4 <= 8$ ✓
$9 <= 8$ X

Sale
i = 9

$9 <= 16$ ✓

Entra
j = ~~1~~ ~~4~~ 3
res = ~~9~~ 13

$1 <= 8$ ✓
$4 <= 8$ ✓
$9 <= 8$ X

Sale
i = 13

$13 <= 16$ ✓

Entra
j = ~~1~~ ~~4~~ 3
res = ~~13~~ 17

$1 <= 8$ ✓
$4 <= 8$ ✓
$9 <= 8$ X

Sale
i = 17

$17 <= 16$ X

Sale del ciclo principal y retorna 17

Punto 3

```
void algoritmo3(int n){
    int i, j, k;
    for(i = n; i > 1; i--)
        for(j = 1; j <= n; j++)
            for(k = 1; k <= i; k++)
                printf("Vida cruel!!\n");
}
```

$1, 2, 3 \longrightarrow 3$

$\longrightarrow n$

$(n-1)(n+1) = n^2 - 1$

$(n)\left(\sum_{i=1}^{n} i+1\right)$

$\hookrightarrow (n)\left(\sum_{i=1}^{n-1} i+1\right)$

$$t(n) = 3 + n + n^2 - 1 + (n)\left(\sum_{i=1}^{n} i+1\right) + (n)\left(\sum_{i=1}^{n-1} i+1\right)$$

$$= n^2 + n + 2 + n\left(\sum_{i=1}^{n} i + \sum_{i=1}^{n} 1\right) + (n)\left(\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1\right)$$

$$= n^2 + n + 2 + n\cdot\left(\frac{n^2+n}{2} + n\right) + n\cdot\left(\frac{n^2-n}{2} + n - 1\right)$$

$$= n^2 + n + 2 + \frac{n^3+n^2}{2} + n^2 + \frac{n^3-n^2}{2} + n^2 - n$$

$$= \frac{n^3+n^2+n^3-n^2}{2} + 3n^2 + 2$$

$$= \frac{\cancel{2}n^3}{\cancel{2}} + 3n^2 + 2$$

$$= n^3 + 3n^2 + 2$$

$$O(n^3)$$

Mejor caso

```
int algoritmo4(int* valores, int n){
   int suma = 0, contador = 0;  ⟶ 2
   int i, j, h, flag;  ⟶ 4

   for(i = 0; i < n; i++){  ⟶ n + 1
     j = i + 1;  ⟶ n
     flag = 0;  ⟶ n
     while(j < n && flag == 0){ 2n
        if(valores[i] < valores[j]){ n
           for(h = j; h < n; h++){ 0
             suma += valores[i]; 0
           }
        }
        else{ n
           contador++; n
           flag = 1; n
        }
        ++j; n
     }
   }
   return contador;  ⟶ 1
}
```

$t(n) = 2 + 4 + n + 1 + 2n + n + n + n + n + n + 1$

$t(n) = 7 + 8n$

$O(n)$

Peor caso

```
int algoritmo4(int* valores, int n){
    int suma = 0, contador = 0; → 2
    int i, j, h, flag; → 4

    for(i = 0; i < n; i++){ → n + 1
      j = i + 1; → n
      flag = 0; → n
      while(j < n && flag == 0){ $\sum_{i=1}^{n} i$
         if(valores[i] < valores[j]){ $\sum_{i=1}^{n-1} i$
            for(h = j; h < n; h++){ $\sum_{i=1}^{n-1}\sum_{j=1}^{n} j$
              suma += valores[i];
            }              ↳ $\sum_{i=1}^{n-1}\sum_{j=1}^{n-1} j$
         }
         else{ 0
            contador++; 0
            flag = 1; 0
         }
         ++j; $\sum_{i=1}^{n-1} i$
      }
    }
    return contador; → 1
}
```

Por el hecho de que hay dos sumatorias unidas, en algún punto, cuando se apliquen las reglas de sumatorias, habrá una multiplicación entre n y n^2, pues hay una sumatoria que depende de otra, por ende habrá ese n^2 mencionado y luego cuando se ejecute la otra sumatoria, el n^2 se multiplicará por n lo que genera que la complejidad sea: $O(n^3)$ y $t(n)$, la suma de línea por línea del código.

Lo que calcula esta operación es la cantidad de veces que el número previo en una lista es menor al que le sigue

```
void algoritmo5(int n){
  int i = 0;  → 1
  while(i <= n){  → 7
    printf("%d\n", i);  → 6
    i += n / 5;  → 6
  }
}
```

Luego de varias pruebas, se nota que a medida de que el número crece, tiende a 7

$t(n) = 1 + 7 + 6 + 6$

$t(n) = 20$

$O(20)$

6. Fibonacci Recursiva

Tamaño Entrada	Tiempo	Tamaño Entrada	Tiempo
5	0.172 s	35	3.287 s
10	0.115 s	40	33.562 s
15	0.059 s	45	4 min 53.845 s
20	0.079 s	50	61 min 48.731 s
25	0.111 s	60	
30	0.265 s	100	

El valor más alto para el que pude obtener tiempo de ejecución es 50, en el cual ya se demora por lo menos una hora en ejecutarse.

Lo que creo que es la complejidad de este algoritmo es $O(2^n)$ debido al hecho de que cada vez que se sume un número, este abrirá dos ramas más y así sucesivamente, entonces cada que aumente un número se multiplicará por dos.

7. Fibonacci Iterativa

Tamaño Entrada	Tiempo	Tamaño Entrada	Tiempo
5	0.096 s	45	0.159 s
10	0.129 s	50	0.128 s
15	0.097 s	100	0.126 s
20	0.137 s	200	0.096 s
25	0.111 s	500	0.160 s
30	0.175 s	1000	0.145 s
35	0.124 s	5000	0.160 s
40	0.095 s	10000	0.158 s

La complejidad de este algoritmo es $O(n)$ ya que solo hay un ciclo y dentro de ese ciclo simplemente hay dos condiciones.

8. Función mostrarPrimos

Tamaño Entrada	Tiempo Solución Propia	Tiempo Solución Profesores
100	0.129 s	0.143 s
1000	0.158 s	0.111 s
5000	0.293 s	0.113 s
10000	0.660 s	0.159 s
50000	11.455 s	0.347 s
100000	44.295 s	0.440 s
200000	2min 39.758 s	1.130 s

a.) Los tiempos de ejecución son bastante diferentes, en el propio se demora mucho más que en el de los profesores. Esto probablemente se deba a que hice un ciclo más y también las iteraciones se podrían reducir drásticamente con evaluaciones que abarquen más resultados.

b.) En el de los profesores como se deben realizar evaluaciones i·i y se realizan únicamente recorridos hasta N, su complejidad al despejar ese i·i, es $O(\sqrt{n})$, lo cual demuestra su menor tiempo respecto a mi solución, pues la mía como hace tres recorridos y hay un ciclo anidado a otro, su complejidad, a pesar de tener varios ciclos que se ejecutan en relación a n, es $O(n^2)$, lo que también certifica los tiempos previamente dados.