Tiempos de ejecución de un algoritmo

Alejandro Giraldo Herrera

Institución universitaria EAM

52410: Ingeniería de software

Ing. Julián Darío Vélez Rodríguez

26 de febrero de 2023

Taller

Hallar los tiempos de ejecución de los algoritmos dados.

No. 1	*MÉTODO*	*PASOS*	*Se ejecuta - VECES-*

1
 	
2
3 4 5
6
7
	int factorial(int n)
	{
	int fact = 1;
	for(int i = n; i > 0; i--)
	fact = fact * i;
	return fact;
	}
		1 int n
		
		2 int fact = 1
		3 int i = n
		4 i > 0
		5 i --
		6 fact = fact * 1
		7 return fact
			1
			
			1
			1
			n + 1
			n
			n
			1

Tiempo de ejecución: $T(n) = 3n + 5$			

No. 2	*MÉTODO*	*PASOS*	*Se ejecuta - VECES-*

1
 	
2
3
4
5
6
	public void ejemplo9(int n)
	{
	int x = 0, i;
	i = 1;
	while (i < n) {
	x++;
	i = i * 2;
	}
	}
		1 int n
		
		2 int x = 0
		3 int i = 1
		4 i < n
		5 x++
		6 i = i * 2
			1
			
			1
			1
			$\log_2 n + 1$
			$\log_2 n$
			$\log_2 n$

Tiempo de ejecución: $T(n) = 3log_2 n + 4$			

No. 3	*MÉTODO*	*PASOS*	*Se ejecuta - VECES-*
1 2	`public void run(int n, int arrOrd[])`	`1 int n`	`1`
	`{`	`2 int arrOrd[]`	`1`
	`  int temp;`	`3 int i = 1`	`1`
3 4 5	`  for (int i = 1; i < n; i++)`	`4 i < n`	`n`
6 7 8	`    for (int j = 0; j < (n - 1);`	`5 i ++`	`n - 1`
9	`j++)`	`6 int j = 0`	n^2
	`      if (arrOrd[j + 1] < arrOrd[j])`	`7 j < (n - 1)`	n^2
10	`      {`	`8 j ++`	$n^2 - 1$
11	`        temp = arrOrd[j];`	`9 arrOrd[j + 1] < arrOrd[j]`	$n^2 - 1$
12	`        arrOrd[j] = arrOrd[j + 1];`	`IF`	
	`        arrOrd[j + 1] = temp;`	`10 temp = arrOrd[j]`	$n^2 - 1$
	`      }`	`11 arrOrd[j] = arrOrd[j +`	$n^2 - 1$
13 14	`      else {`	`1]`	$n^2 - 1$
15	`        for (int k = 0; k < n; k++)`	`12 arrOrd[j + 1] = temp`	
16	`          temp = arrOrd[k];`	`ELSE`	n^3
	`      }`	`13 int k = 0`	$n^3 + 1$
	`}`	`14 k < n`	n^3
		`15 k ++`	n^3
		`16 temp = arrOrd[k]`	

Tiempo de ejecución:

If(9,12): $T(n) = 3(n^2 - 1)$
Else(13,16): $T(n) = 4n^3 + 1$ ➡️ Más grande

$T(n) = 2 + 2n + 4n(n) - 2 + Else$ ➡️$T(n) = 4n^3 + 4n^2 + 2n + 1$

No. 4	*MÉTODO*	*PASOS*	*Se ejecuta - VECES-*
1 2 3	`public void llamado3(int n, int a, int b)`	`1 int n`	`1`
		`2 int a`	`1`
4 5 6	`{`	`3 int b`	`1`
	`  if (n > a && b >= d)`	`4 n > a`	`1`
7 8 9	`  {`	`5 b >= d`	`INDETERMINACIÓN | 1`
	`    for (int i = 0; i < 2 * n; i += 2)`	`6 n > a && b >= d`	`INDETERMINACIÓN | 1`
10	`    {`	`7 int i = 0`	`1`
	`      if (n > c)`	`8 i < 2 * n`	`(2 * n + 1) / 2 = n + 1`
11	`      {`	`9 i += 2`	`(2 * n) / 2 = n`
12	`        metodo1();`	`10 n > c`	`n`
	`        metodo2();`	`IF`	
	`      }`	`11 metodo1()`	`n`
	`    }`	`12 metodo2()`	`n`
	`  }`	`ELSE`	
13	`  else {`	`13 metodo3()`	`n`
	`    metodo3();`		
	`  }`		
	`}`		

Tiempo de ejecución: La variable ***d*** en el paso 5 y 6 no está definida, lo que ocasiona un error de ejecución. Si ignoramos este detalle, el tiempo de ejecución de este algoritmo es:

metodo1() ➡️$T(n) = n^3$
metodo2() ➡️$T(n) = n^2$
metodo3() ➡️$T(n) = n^5 + 5$

If(11,12) ➡️ $T(n) = metodo1() + metodo2()$ ➡️ $T(n) = n^3 + n^2$ ➡️ $T(n) = n^2(n + 1)$
Else(13) ➡️ $T(n) = metodo3()$ ➡️ $T(n) = n^5 + 5$ ➡️ Más pesado

$T(n) = 7 + n + 1 + n + n + Else$ ➡️ $T(n) = 8 + 3n + n^5 + 5$ ➡️ $T(n) = n^5 + 2n + 13$

No. 5	*MÉTODO*	*PASOS*	*Se ejecuta - VECES-*
1	public void metodo(int[] arreglo)	1 int[] arreglo	1
	{	2 int izquierda = 1	1
2	int izquierda = 1;	3 int derecha = arreglo.length	1
3	int derecha = arreglo.length;	4 int ultimo = arreglo.length - 1	1
4	int ultimo = arreglo.length - 1;	5 derecha >= 0	(n + 1) + 1 = n + 2
		6 metodo2(ultimo, arreglo)	n + 1
5	While {derecha >= 0)	7 izquierda = ultimo + 1	n + 1
	{	8 metodo3()	n + 1
6	metodo2(ultimo, arreglo);	9 derecha-	n + 1
7	izquierda = ultimo + 1;	10 int i = 0	1
8	metodo3();	11 i < arreglo.length	n + 1
9	derecha--;	12 i ++	n
	}	13 arreglo[i]	n
10 11 12	for (int i = 0; i < arreglo.length; i++)	14 int ultimo	1
	{	15 int[] arreglo	1
13	System.out.println(arreglo[i]);	16 int i = arreglo.length - 1	1
	}	17 i > 0	n
	}	18 i --	n - 1
		19 arreglo[i - 1] > arreglo[i]	n - 1
14 15	public void metodo2(int ultimo, int[]	IF	
	arreglo)	20 int aux = arreglo[i]	n - 1
16 17 18	{	21 arreglo[i] = arreglo[i - 1]	n - 1
	for (int i = arreglo.length - 1; i > 0;	22 arreglo[i - 1] = aux	n - 1
19	i--)	23 ultimo = i	n - 1
	{		
20	if (arreglo[i - 1] > arreglo[i])		

21	{
	int aux = arreglo[i];
	arreglo[i] = arreglo[i - 1];
	arreglo[i - 1] = aux;
	último = i;
	}
	}		

24 25	}	24 int ultimo 25 int[] arreglo	1 1
26 27 28	Public void metodo3(int ultimo, int[] arreglo)	26 int j = 1 27 j < arreglo.length	1 ((n - 1) / 2) + 1 = (n+1) / 2
29	{	28 j += 2	(n - 1) / 2
30	for (int j = 1; j < arreglo.length; j += 2)	29 arreglo[j - 1] > arreglo[j] IF	(n - 1) / 2
31	{	30 int aux = arreglo[j]	(n - 1) / 2
32	if (arreglo[j - 1] > arreglo[j])	31 arreglo[j] = arreglo[j - 1]	(n - 1) / 2

33	{
	int aux = arreglo[j];
	arreglo[j] = arreglo[j - 1];
		32 arreglo[j - 1] = aux
		33 ultimo = j
		ELSE
			(n - 1) / 2
			(n - 1) / 2

34	arreglo[j - 1] = aux;	34 aux = arreglo[j] + arreglo[j - 1]	(n - 1) / 2
35	ultimo = j;	35 arreglo[j - 1] = aux	(n - 1) / 2
36	}	36 ultimo = j	(n - 1) / 2

37	else {
	aux = arreglo[j] + arreglo[j - 1];
	arreglo[j - 1] = aux;
	ultimo = j;
	break;
	}
	}
	}
		37 break	1

Tiempo de ejecución: El algoritmo presenta fallas de sintaxis que provocan errores en el tiempo de ejecución. Las fallas se encuentran en el carácter ***W*** y ***{*** en el paso 5, y en el carácter ***P*** en el paso 24. Si se ignoran estos errores, el tiempo de ejecución del algoritmo es:

metodo() ➡$T(n) = 5(n+1) + 2n + 5 + n + 2$ ➡ $T(n) = 5n + 5 + 3n + 7$ ➡ $T(n) = 8n + 12$
metodo2() ➡$T(n) = 6(n-1) + n + 3$ ➡ $T(n) = 6n - 6 + n + 3$ ➡ $T(n) = 7n - 3$
metodo3-If() ➡ $T(n) = 4((n-1)/2)$ ➡ $T(n) = 2(n-1)$ ➡ $T(n) = 2n - 2$ ➡ Más pesado
metodo3-Else() ➡ $T(n) = 3((n-1)/2) + 1$
metodo3() ➡ $T(n) = 3 + ((n+1)/2) + 2((n-1)/2) + 2n - 2$ ➡ $T(n) = ((n+1)/2) + 3n$

$T(n) = 7n + 11 + (n-1)(7n-2) + (n-1)(((n+1)/2) + 3n)$ ➡ $T(n) = 7n + 11 + 7n^2 - 2n - 7n + 2 + ((n^2-1)/2) + 3n^2 - 3n$
➡ $T(n) = 10n^2 + ((n^2-1)/2) - 5n + 13$