UNLP. Facultad de Informática.
Algoritmos y Estructuras de Datos
Cursada 2017

Práctica 5
Tiempos de Ejecución

Ejercicio 1

Dadas las siguientes funciones ordenarlas según sus velocidades de crecimiento utilizando el graficador de funciones que se encuentra en https://www.desmos.com/calculator

Dado que el graficador solo trabaja con logaritmos en base 10 (log o $\log_{10}$) utilice el cambio de base para convertir los logaritmos (tal como se explica en: http://es.wikipedia.org/wiki/Logaritmo#Cambio_de_base). Básicamente:

$$Log_a x = \frac{Log_b x}{Log_b a} \Leftrightarrow a \neq 1, b \neq 1$$

a. $T1(n) = n*\log_2(n)$
b. $T2(n)= (1/3)^n$
c. $T3(n) =2n+n^2$
d. $T4(n) = (3/2)^n$
e. $T5(n) = (\log_2(n))^2$
f. $T6(n) = \log_2(n)$
g. $T7(n) = n+\log_2(n)$
h. $T8(n) = n^{1/2}$
i. $T9(n) = 3^n$
j. $T10(n) = 2^n$

Ejercicio 2

Determinar si las siguientes sentencias son verdaderas o falsas, justificando la respuesta utilizando notación Big-Oh.

a. 3^n es de $O(2^n)$
b. $n/\log_2(n)$ es de $O(\log_2(n))$
c. $n^{1/2} + 10^{20}$ es de $O(n^{1/2})$
d. $n + \log_2(n)$ es de $O(n)$
e. Si p(n) es un polinomio de grado k, entonces p(n) es $O(n^k)$. Mostrar que $p(n)=3n^5 + 8n^4 + 2n +1$ es $O(n^5)$
f. $\begin{cases} 3n+17, n<100 \\ 317, n \geq 100 \end{cases}$ tiene orden lineal

g. $\begin{cases} n^2, n \le 100 \\ n, n > 100 \end{cases}$ tiene orden cuadrático

h. 2^{n+1} es de O (2^n)

i. 2^{2n} es de O (2^n)

Ejercicio 3

Se necesita generar una permutación random de los n primeros números enteros. Por ejemplo [4,3,1,0,2] es una permutación legal, pero [0,4,1,2,4] no lo es, porque un número está duplicado (el 4) y otro no está (el 3). Presentamos tres algoritmos para solucionar este problema. Asumimos la existencia de un generador de números random, ran_int (i,j) el cual genera en tiempo constante, enteros entre i y j inclusive con igual probabilidad (esto significa que puede retornar el mismo valor más de una vez). También suponemos el mensaje swap() que intercambia dos datos entre sí. Si bien los tres algoritmos generan permutaciones random legales, tenga presente que por la forma en que utilizan la funcion ran_int **algunos de ellos podrían no terminar nunca**, mientras otro sí.

```
public class Ejercicio4 {
	private static Random rand = new Random();

	public static int[] randomUno(int n) {
		int i, x = 0, k;
		int[] a = new int[n];
		for (i = 0; i < n; i++) {
			boolean seguirBuscando = true;
			while (seguirBuscando) {
				x = ran_int(0, n - 1);
				seguirBuscando = false;
				for (k = 0; k < i && !seguirBuscando; k++)
					if (x == a[k])
						seguirBuscando = true;
			}
			a[i] = x;
		}
		return a;
	}

	public static int[] randomDos(int n) {
		int i, x;
		int[] a = new int[n];
		boolean[] used = new boolean[n];
		for (i = 0; i < n; i++) used[i] = false;
		for (i = 0; i < n; i++) {
			x = ran_int(0, n - 1);
			while (used[x]) x = ran_int(0, n - 1);
			a[i] = x;
			used[x] = true;
		}
		return a;
	}

	public static int[] randomTres(int n) {
		int i;
		int[] a = new int[n];
		for (i = 0; i < n; i++) a[i] = i;
```

```
		for (i = 1; i < n; i++) swap(a, i, ran_int(0, i - 1));
		return a;
	}

	private static void swap(int[] a, int i, int j) {
		int aux;
		aux = a[i]; a[i] = a[j]; a[j] = aux;
	}

	/** Genera en tiempo constante, enteros entre i y j con igual probabilidad.
	 */
	private static int ran_int(int a, int b) {
		if (b < a || a < 0 || b < 0) throw new IllegalArgumentException("Parametros
invalidos");
		return a + (rand.nextInt(b - a + 1));
	}

	public static void main(String[] args) {
		System.out.println(Arrays.toString(randomUno(1000)));
		System.out.println(Arrays.toString(randomDos(1000)));
		System.out.println(Arrays.toString(randomTres(1000)));
	}
}
```

a. Determinar que algoritmos podrían no terminar nunca. Calcular el tiempo de ejecución para el / los algoritmos que terminan y demostrar que orden tienen utilizando la notación Big-Oh.

Ejercicio 4

a.- Considerando que un algoritmo requiere ***f(n)*** operaciones para resolver un problema y la computadora procesa 100 operaciones por segundo.

Si f(n) es:

i. log10 n
ii. $\sqrt{n}$

Determine el tiempo en segundos requerido por el algoritmo para resolver un problema de tamaño *n*=10000.

b.- Suponga que Ud. tiene un algoritmo ALGO-1 con un tiempo de ejecución exacto de $10n^2$. ¿En cuánto se hace más lento ALGO-1 cuando el tamaño de la entrada n aumenta:……….?

a) El doble
b) El triple

Ejercicio 5

Para cada uno de los siguientes fragmentos de código, determine en forma intuitiva el orden de ejecución:

`for(int i = 0; i< n; i++)` `    sum++;`	`for(int i = 0; i< n; i+=2)` `    sum++;`

`for(int i = 0; i< n; i++)`
`    for(int j = 0; j< n; j++)`
`        sum++;`
	`for(int i = 0; i< n; i++)`
	`    for(int j = 0; j< n*n; j++)`
	`        sum++;`

`for(int i = 0; i< n; i++)`
`    for(int j = 0; j< n; j++)`
`        sum++;`
`for(int i = 0; i< n; i++)`
`    sum++`
	`for(int i = 0; i< n/2; i++)`
	`    for(int j = 0; j< n/2; j++)`
	`        sum++;`

Ejercicio 6
Para cada uno de los algoritmos presentados:

a. Expresar en función de n el tiempo de ejecución.
b. Establecer el orden de dicha función usando notación big-Oh.

En el caso de ser necesario tenga presente que:

$$\sum_{i=1}^{n} i^4 = \frac{n(n+1)\ (6n^3+9n^2+n-1)}{30}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)\ (2n+1)}{6}$$

1.

```
public static void uno (int n)  {
        int  i, j, k ;
        int [] [] a, b, c;
        a = new int [n] [n];
        b = new int [n] [n];
        c = new int [n] [n];
        for ( i=1; i<=n-1; i++) {
                for ( j=i+1; j<=n; j++) {
                        for ( k=1; k<=j; k++)  {
                                c[i][j] = c[i][j]+ a[i][j]*b[i][j];
                        }
                }
        }
}
```

2.

```
public static void dos (int n){
        int i, j, k, sum;
        sum = 0;
        for ( i=1; i<=n; i++) {
                for ( j=1; j <= i*i; j++) {
                        for ( k=1; k<= j; k++)  {
```

```
                                    sum = sum + 1;
                              }
                        }
                  }
            }
```

Ejercicio 7

Para cada uno de los algoritmos presentados calcule el T(n).

a. Expresar en función de n el tiempo de ejecución.

b. Establecer el orden de dicha función usando notación big-Oh.

1.

```
int c = 1;
while ( c < n ) {
    algo_de_O(1);
    c = 2 * c;
}
```

2.

```
int c = n;
while ( c > 1 ) {
algo_de_O(1);
        c = c / 2;
}
```

3.

```
int x=1;
for (int i = 1; i < n; i = i+4)
        for (int j = 1; j < n; j = j+|n/4|)
                for (int k = 1; k < n; k = k*2)
                        x = x+1;
```

4.

```
j = 1;
while (j <= n) {
    for (i = n*n; i >=1; i = i-3)
          x=x+1;
    j = j*2;
}
```

Ejercicio 8

a. Exprese la función del tiempo de ejecución de cada uno de los siguientes algoritmos, resuélvala y calcule el orden.

b. Comparar el tiempo de ejecución del método 'rec2' con el del método 'rec1'.

c. Implementar un algoritmo más eficiente que el del método rec3 (es decir que el T(n) sea menor).

```
package estructurasdedatos;

public class Recurrencia {
        static public int rec2(int n){
                if (n <= 1)
                        return 1;
                else
                        return (2 * rec2(n-1));
        }

        static public int rec1(int n){
                if (n <= 1)
                        return 1;
                else
                        return (rec1(n-1) + rec1(n-1));
        }
```

```
        static public int rec3(int n){
                if  ( n == 0 )
                        return  0;
                else {
                        if ( n == 1 )
                                return 1;
                        else
                                return (rec3(n-2) * rec3(n-2));
                }
        }

        static public int potencia_iter(int x, int n){
                int potencia;
                if  (n == 0)
                        potencia = 1;
                else {
                        if  (n == 1)
                                potencia  =  x;
                        else{
                                potencia  = x;
                                for (int  i  = 2 ; i <=  n ; i++) {
                                        potencia *= x ;
                                }
                        }
                }
                return  potencia;
        }

        static public int potencia_rec( int x, int n){
                if( n == 0 )
                        return 1;
                else{
                        if( n == 1)
                                return x;
                        else{
                                if ( (n  %  2 ) == 0)
                                        return  potencia_rec (x * x, n / 2 );
                                else
                                        return potencia_rec (x * x, n / 2) * x;
                        }
                }
        }
}
```

Ejercicio 9
Dado el siguiente método, plantear y resolver la función de recurrencia:

```
int funcion(int n) {
        int x = 0;
        if (n <= 1)
                return 1;
        else {
                for (int i = 1; i < n; i++) {
                        x = 1;
                        while (x < n) {
                                x = x * 2;
                        }
                }
```

```
                return funcion(n/2) + funcion(n/2);
            }
        }
```

Ejercicio 10

a) Calcule el Tiempo de ejecución.
b) Determine el Orden por definición.
c) Indique que valor retorna el algoritmo y compárelo con a.

```
static public int recursivo(int n){
      if (n == 1)
            return  1;
      else
            return (n * recursivo (n-1));
}
```

Ejercicio 11

Intuitivamente determine el orden de ejecución de las siguientes funciones de recurrencia sin resolverlas.

$T(n)=\begin{cases} 1, n\le 1 \\ T(n-1)+c, n\ge 2 \end{cases}$	$T(n)=\begin{cases} 1, n=1 \\ T(n/2)+c, n\ge 2 \end{cases}$
$T(n)=\begin{cases} 1, n=1 \\ 2T(n/2)+c, n\ge 2 \end{cases}$	$T(n)=\begin{cases} 1, n\le 5 \\ T(n-5)+c, n\ge 6 \end{cases}$
$T(n)=\begin{cases} 1, n=1 \\ 2T(n-1)+c, n\ge 2 \end{cases}$	$T(n)=\begin{cases} 1, n\le 7 \\ T(n/8)+c, n\ge 8 \end{cases}$

Ejercicio 12

a.- Calcular analíticamente el T(n), detallando los pasos seguidos para llegar al resultado.

b.- Calcular el O(n) justificando usando la definición de big-OH

1.

$$T(n)=\begin{cases} 2, n=1 \\ T(n-1)+n, n\ge 2 \end{cases}$$

2.

$$T(n)=\begin{cases} 2, n=1 \\ T(n-1)+\frac{n}{2}, n\ge 2 \end{cases}$$

3.

4.

$$T(n) = \begin{cases} 1, n = 1 \\ 4T\left(\frac{n}{2}\right) + n^2, n \geq 2 \end{cases}$$

$$T(n) = \begin{cases} 1, n = 1 \\ 8T\left(\frac{n}{2}\right) + n^3, n \geq 2 \end{cases}$$

5.

$$T(n) = \begin{cases} 1 & \mathrm{n} = 1 \\ 2\mathrm{T}\left(\frac{\mathrm{n}}{4}\right) + \sqrt{\mathrm{n}}, & \mathrm{n} \geq 2 \end{cases}$$

6.

$$T(n) = \begin{cases} 1 & \mathrm{n} = 1 \\ 2\mathrm{T}(n/2) + c, & \mathrm{n} \geq 2 \end{cases}$$