UNLP. Facultad de Informática.

Algoritmos y Estructuras de Datos

Práctica 4
Tiempos de Ejecución

Ejercicio 1

Debido a un error en la actualización de sus sistemas, el banco AyED perdió la información del estado de todas sus cuentas. Afortunadamente logran recuperar un backup del día anterior y utilizando las transacciones registradas en las últimas 24hrs podrán reconstruir los saldos. Hay poco tiempo que perder, el sistema bancario debe volver a operar lo antes posible.

Las transacciones se encuentran agrupadas en consultas, una consulta cuenta con un valor y un rango de cuentas consecutivas a las que hay que aplicar este cambio, por ejemplo la consulta (333..688 = 120) implica sumar $120 a todas las cuentas entre la número 333 y la número 688 (inclusive). Entonces, la recuperación de los datos consiste en aplicar todas las consultas sobre el estado de las cuentas recuperado en el backup del día anterior.

El equipo de desarrollo se pone manos a la obra y llega a una solución rápidamente. Toman cada consulta y recorren el rango de cuentas aplicando el valor correspondiente, como muestra el siguiente algoritmo.

```
Consultas.comenzar()
While(!consultas.fin()){
    Consulta = consultas.proximo();
    for(i = consulta.desde; i < consulta.hasta; i++){
        cuenta[i] = cuenta[i] + consulta.valor;
    }
}
```

Escriben la solución en pocos minutos y ponen en marcha el proceso de recuperación. Enseguida se dan cuenta que el proceso va a tardar muchas horas en finalizar, son muchas cuentas y muchos movimientos, la solución aunque simple es ineficiente. Luego de discutir varias ideas llegan a una solución que logra procesar toda la información en pocos segundos.

a.- Para que usted pueda experimentar el tiempo que demora cada uno de los dos algoritmos, lo ponemos a vuestra disposición. Usted debe ejecutar cada uno de los algoritmo, con distinta cantidad de elementos y complete la tabla. Luego haga la gráfica para comparar los tiempos de ambos algoritmos. Tenga encuenta que el algoritmo posee dos constantes CANTIDAD_CUENTAS y CANTIDAD_CONSULTAS, sin embargo, por simplicidad, ambas toman el mismo valor. Solo necesita modificar CANTIDAD_CUENTAS

Nº Cuentas (y consultas)	procesarMovimientos	procesarMovimientosOptimizado
1.000		
10.000		
25.000		
50.000		
100.000		

b.- ¿Por qué **procesarMovimientos** es tan ineficiente? Tenga en cuenta que pueden existir millones de movimientos diarios que abarquen gran parte de las cuentas bancarias.

c.- ¿En qué se diferencia **procesarMovimientosOptimizado**? Observe las operaciones que se realizan para cada consulta.

Aunque los dos algoritmos se encuentran explicados en los comentarios, no es necesario entender su funcionamiento para contestar las preguntas.

Ejercicio 2
La clase BuscadorEnArrayOrdenado del material adicional resuelve el problema de buscar un elemento dentro de un array ordenado. El mismo problema, lo resuelve de dos maneras diferentes: busqueda lineal y búsqueda dicotómica.

Se define la variable cantidadElementos, la cual debe ir modificando para determinar una escala (por ejemplo de a 100.000 o 1.000.000, dependiendo de la capacidad de cada equipo), y realice una tabla del tiempo que tarda en ejecutar ambos algoritmos. Por ejemplo:

N	Lineal	Dicotómica
100.000		
200.000		
300.000		
400.000		
500.000		
600.000		

Ejercicio 3
En la documentación de la clase arrayList que se encuentra en el siguiente link https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Se encuentran las siguientes afirmaciones

- "The size, isEmpty, get, set, iterator, and listIterator operations run in constant time."
- "All of the other operations run in linear time (roughly speaking)"

Explique con sus palabras por qué cree que algunas operaciones se ejecutan en tiempo constante y otras en tiempo lineal.

Ejercicio 4

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 $O(n)$

c. $n^{1/2} + 10^{20}$ es de $O(n^{1/2})$

d. $\begin{cases} 3n+17, n<100 \\ 317, n \geq 100 \end{cases}$ tiene orden lineal

e. Mostrar que $p(n)=3n^5 + 8n^4 + 2n + 1$ es $O(n^5)$

f. Si $p(n)$ es un polinomio de grado k, entonces $p(n)$ es $O(n^k)$.

Ejercicio 5

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.

```
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. Analizar si todos los algoritmos terminan o alguno puede quedar en loop infinito.

b. Describa con palabras la cantidad de operaciones que realizan.

Ejercicio 6

a.- Se tiene un algoritmo A, que se ejecuta sobre una computadora que procesa 10.000 operaciones por segundo. Si el algoritmo A requiere $f(n) = n * \log_{10} n$ operaciones para resolver un problema, determine el tiempo en segundos requerido por el algoritmo para resolver un problema de tamaño n=10.000.

b.- Suponga que tenemos un algoritmo ALGO-1 cuyo tiempo de ejecución exacto es $100n^3$ para un tamaño de entrada n.

i) si el tamaño de la entrada aumenta al doble, es decir, sería 2n, ¿Cuánto más lenta sería la respuesta del algoritmo ALGO-1?

ii) y si aumenta al triple?

Ejercicio 7

Para cada uno de los siguientes fragmentos de código, calcule el tiempo 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 8

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 9

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;
}
```

Ejercicio 10

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 11
a) Calcule el Tiempo de ejecución.
b) Determine el Orden por definición.

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

Ejercicio 12

Resuelva las recurrencias y calcule el orden. Para cada recurrencia se muestra a modo de ejemplo el código correspondiente.

```
int recursivo(int n){
if (n <= 1)
       return  1;
else
       return (recursivo (n-1));

}
```

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

```
int recursivo(int n){
if (n = 1)
       return  1;
else
       return (recursivo (n/2));

}
```

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

```
int recur (int n){
if (n = 1)
       return  1;
else
       return(recur(n/2)+recur(n/2));

}
```

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

```
int recursivo(int n){
if (n <= 5)
       return  1;
else
       return (recursivo (n-5));

}
```

$$T(n) = \begin{cases} 1, n \leq 5 \\ T(n-5) + c, n \geq 6 \end{cases}$$

```
int recur (int n){
if (n = 1)
       return  1;
else
       return(recur(n-1)+recur(n-1));

}
```

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

```
int recursivo(int n){
if (n <= 7)
       return  1;
else
       return (recursivo (n/8));

}
```

$$T(n) = \begin{cases} 1, n \leq 7 \\ T(n/8) + c, n \geq 8 \end{cases}$$

Ejercicio 13

a.- Resolver las siguientes recurrencias. b.- Calcular el O(n) justificando usando la definición de big-OH

1.

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

2.

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

2.

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

3.

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

4.

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

Ejercicio 14

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 15

Calcule el tiempo de ejecución de los métodos buscarLineal y buscarDicotomica de la clase BuscadorEnArrayOrdenado. Compare el tiempo con los valores obtenidos empíricamente en el ejercicio 2

Ejercicio 16

Calcule el tiempo de ejecución de procesarMovimientos y procesarMovimientosOptimizado del ejercicio 1. Compare el tiempo con los valores obtenidos empíricamente.