ESTRUCTURAS DE DATOS

TRABAJO PRÁCTICO N ° 2

Tiempo de Ejecución

Departamento de Ciencias e Ingeniería de la Computación - U.N.S.
Primer cuatrimestre de 2019

Bibliografía:
[AU] Aho, A. V., Hopcroft, J. E. and Ullman, J. D. Data Estructures and Algorithms. Addison Wesley, 1983.
[GT] Michael Goodrich & Roberto Tamassia. Data Structures and Algorithms in Java. Fourth Edition. John Wiley and Sons. 2006.
[W12] Mark A. Weiss. Data Structures and Algorithm Analysis in Java. Third Edition. Addison-Wesley Pearson Education, Inc. 2012.

Ejercicio 1:
Ordene las siguientes funciones por tasa de crecimiento asintótico (asumir que la base de logaritmo es k):

$4n \log n + 2n$
2^{10}
$2^{\log n}$
$3n + 100 \log n$
$4n$
2^n
$n^2 + 10n$
n^3
$n \log n$

Ejercicio 2:
Si el número de operaciones ejecutado por los algoritmos A y B es $8n \log n$ y $2n^2$ respectivamente. Determine n_0 tal que A sea mejor que B para $n \geq n_0$.

Ejercicio 3:
Si el número de operaciones ejecutado por los algoritmos *A* y *B* es $40n^2$ y $2n^3$ respectivamente. Determine n_0 tal que *A* sea mejor que *B* para $n \geq n_0$.

Ejercicio 4:
Muestre que si $d(n)$ es $O(f(n))$, entonces $ad(n)$ es $O(f(n))$ para cualquier constante $a > 0$.

Ejercicio 5:
Muestre que 2^{n+1} es $O(2^n)$.

Ejercicio 6:
Muestre que si $d(n)$ es $O(f(n))$ y $e(n)$ es $O(g(n))$, entonces el producto $d(n)e(n)$ es $O(f(n)g(n))$.

Ejercicio 7:
Muestre que si $d(n)$ es $O(f(n))$ y $f(n)$ es $O(g(n))$, entonces $d(n)$ es $O(g(n))$.

Ejercicio 8:
Muestre que si $p(n)$ tiene complejidad polinomial en n, entonces $\log(p(n))$ es $O(\log(n))$.

Ejercicio 9:
Analice el tiempo de ejecución del algoritmo BinarySum del fragmento de código 3.34 [GT].

Ejercicio 10:
De una caracterización usando la notación Big-Oh del tiempo de ejecución de los algoritmos Ex1, Ex2, Ex3, Ex4 y Ex5 presentados en el fragmento de código 4.5[GT].

Ejercicio 11:
Sean P_1 y P_2 dos programas cuyos tiempos de ejecución son $T_1(n)$ y $T_2(n)$ respectivamente, donde n es el tamaño de la entrada. Determine para los siguientes casos en qué condiciones P_2 se ejecuta más rápido que P_1.

a) $T_1(n) = 2.n^2$ $T_2(n) = 1000.n$
b) $T_1(n) = 3.n^4$ $T_2(n) = 3.n^3$
c) $T_1(n) = 126.n^2$ $T_2(n) = 12.n^4$

Ejercicio 12:
Para cada función f(n) y tiempo t en la siguiente tabla, determine el tamaño máximo *n* de un problema P que puede ser resuelto en el tiempo t si la complejidad temporal del algoritmo que resuelve P es f(n) y está expresado en microsegundos.

Tiempo de ejecución f(n) expresado en microsegundos	Tamaño máximo de problema (*n*)			
	1 segundo	1 minuto	1 hora	1 día
200 n	5.000			
3 n^2				
2 n^3				
6 $\log_2 n$				
2^n				

Ejercicio 13:
Determine el caso más desfavorable de los tiempos de ejecución de las siguientes rutinas como una función de n:

a) Calculando el tiempo de ejecución *T(n)*.
b) Calculando el orden del tiempo de ejecución. Asuma que el tiempo de ejecución de una instrucción de asignación, comparación, return, lectura, o una de escritura, es una constante C.

Nota: Fórmulas útiles para el cálculo de T(n) ☺

$$\sum_{i=1}^{n} 1 = n \quad \sum_{i=1}^{n} k = kn \quad \sum_{j=i}^{n} 1 = n - i + 1 \quad \sum_{i=a}^{n} i = (a+n)(n-a+1)\frac{1}{2}$$

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

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

Método Ejemplo1:

```
static void ejemplo1(int n)
{
int a, b, i, j;

i=1;  j=2;
for( a=1; a<=n; a++ )
             if( NumeroLindo(a) ) {
                   for( b=a+1; b<=n; b++ ) i *= 3;
                   for( b=1; b<=a; b++ ) j *= 2;
             } else {
                   i *= 2;
             }
}
```

Método búsqueda binaria: Analice además qué ocurre si el arreglo está vacío.
Entrada: Un arreglo a ordenado en forma ascendente de n elementos, con n > 0.
Salida: true si elem pertenece al arreglo a y false en caso contrario.

```
static bool busqueda_binaria( float elem, float [ ] a, int n )
{
      int pri = 1, ultimo = n, miro;
      bool encontre = false;
      do {

             miro = (pri + ultimo) / 2;
             if( elem == a[miro] )
                   encontre = true;
             else if( elem < a[miro] ) ultimo = miro - 1;
             else pri = miro + 1;
      } while ( pri <= ultimo && !encontre );
      return encontre;

}
```

Método factorial:
Entrada: un entero n
Salida: n! = 1 x 2 x 3 x 4 x ... x (n-1) x n

```
static int factorial( int n )
{
      if ( n > 0 ) return n * factorial( n - 1 );
      else return 1;
}
```

Método Pot1:
Entrada: un entero n
Salida: 2^n

```
static int Pot1( int n )
{
      if( n <= 0 ) return 1;
      else return Pot1( n - 1 ) + Pot1( n - 1);
}
```

Método Pot2:
Entrada: un entero n
Salida: 2^n

```
static int Pot2( int n )
{
      if( n <= 0 ) return 1;
      else return 2 * Pot2( n - 1 );
}
```

Método Pot3:
Entrada: un entero n
Salida: 2^n

```
static int Pot3( int n )
{
       if( n <= 0 ) return 1;
       else return (int) Math.round(Math.exp( n * Math.log( 2.0 ) ));
}
```

Ejercicio 14:

a) Describa un algoritmo eficiente para encontrar el décimo elemento mayor de un arreglo de tamaño *n*. ¿Cuál es el tiempo de ejecución de su algoritmo?

b) Muestre que $\sum_{i=1}^{n} i^2$ es $O(n^3)$

c) Muestre que $5n^2$ es $\Omega(n^2)$

d) Muestre que nlog(n) es $\Omega(n)$

e) Muestre que $5n^2+3$ es $\theta(n^2)$

Ejercicio 15:

Determine el caso más desfavorable de los tiempos de ejecución de las siguientes rutinas de ordenamiento como una función de n:

a) Calculando el tiempo de ejecución *T(n)*.

b) Calculando el orden del tiempo de ejecución. Asuma que el tiempo de ejecución de una instrucción de asignación, comparación, return, lectura, o una de escritura, es una constante C.

Selectionsort
Entrada: un entero n y un arreglo de enteros a
Explicación: Haremos n-1 pasadas sobre el arreglo a.
En la pasada i-ésima, encontramos el i-ésimo menor elemento del arreglo y lo intercambiamos con a[i].

Pasada 1: 3 5 2 1 4 intercambiar a[0] con a[3]

Pasada 2: 1 5 2 3 4 intercambiar a[1] con a[2]

Pasada 3: 1 2 5 3 4 intercambiar a[2] con a[3]

Pasada 4: 1 2 3 5 4 intercambiar a[3] con a[4]

1 2 3 4 5

```
public static void selectionsort(int []a, int n)
 {    for( int i=0; i<n-1; i++ ) {
          // Hallar el mínimo a[p] de a[i], …, a[n-1]
          p = i;
          for( int j=i+1; j<n; j++ )
              if( a[j] < a[p] ) p = j;
          //Intercambiar a[i] y a[p]
          int item = a[p];
          a[p] = a[i];
          a[i] = item;
      }
}
```

BubbleSort

Entrada: un entero n y un arreglo de enteros a

Explicación: En cada pasada del método burbuja mira dos elementos adyacentes y los intercambia si están fuera de orden.
En cada pasada, el mayor elemento del arreglo queda al final del subarreglo que se está ordenando.

```
public static void bubblesort( int [] a, int n ) {
for( int i=n-1; i>=1; i-- ) {
            // Burbujear el item más grande en a[0], …, a[i] a a[i]
            for( int j=0; j<i-1; j++ )
               if( a[j] > a[j+1] ) {
                     // Intercambiar items
                     int item = a[j];
                     a[j] = a[j+1];
                     a[j+1] = item;
               }
            }
}
```

InsertionSort

Entrada: un entero n y un arreglo de enteros a

Explicación: Es equivalente a la forma de ordenar un mazo de cartas, se comienza con un mazo vacío y uno desordenado. Cada carta se trata de insertar en la posición correcta.
En un momento dado, una parte del mazo está ordenada y se trata de insertar la siguiente carta de la porción desordenada del mazo en la posición correcta.
Terminamos cuando la porción desordenada está vacía.

```
public static void insertionsort( int [] a, int n ) {
       for( int i = 1; i<n; i++ ) {
                 // Insertar a[i] en la secuencia ordenada a[0], …, a[i-1]
                 int item = a[i]; // item a insertar
                 int j = i; // puntero de inserción
                 boolean found = false;
                 while( j>0 && !found )
                           if( a[j-1] <= item ) // El item debería ser a[j]
                                  found = true;
                          else {
                                 a[j] = a[j-1]; // Mover a[j-1] para arriba
                                  j--;
                           }
                 a[j] = item; // Insertar item
        }
}
```

MergeSort

Entrada: un entero n y un arreglo de enteros a

Explicación: Ordenamiento por mezcla
Caso recursivo: Partir el arreglo en dos, ordenar recursivamente cada mitad y luego hacer la mezcla de cada mitad ordenada en un gran arreglo ordenado.
Caso base: El arreglo tiene 0 o 1 componentes entonces está ordenado.

```
public static void mergesort( int [] a, int n){
       msort( a, 0, n-1);
}
```

```
private static void msort( int [] a, int ini, int fin) {
       if( ini < fin) {
           int medio = (ini + fin) / 2;
           msort(a, ini, medio );
           msort(a, medio + 1,fin);
           merge( a, ini, medio, fin ); // merge hace la mezcla de los sub-arreglos
       }
 }
```

QuickSort

Entrada: un entero ini, un enterro fin y un arreglo de enteros a

Explicación: El quick sort trabaja como merge sort pero evita hacer la mezcla

Algoritmo: Si la lista es no vacía entonces

1) dividir la lista en dos de tal manera que los ítems en la primera mitad vengan antes que los items en la segunda mitad (acomodar pivot)
2) ordenar con quick sort la primera mitad
3) ordenar con quick sort la segunda mitad

```
public static void QuickSort (int [] a, int ini, int fin) {
   int pospivot;
   if (ini < fin){
      pospivot =AcomodarPivot (a,ini,fin);
      QuickSort (a, ini,pospivot-1);
      QuickSort (a, pospivot+1,fin);
   }
}

private static int AcomodaPivot (int [] a, int ini, int fin) {
   pos =Avanzar (a,ini,fin);
}

private static int Avanzar(int [] a, int ini,int fin){
   int posp;
   if(ini>=fin)
         posp=ini;
   else{
         if(a[ini]>a[ini+1]){
             intercambiar(a,ini,ini+1);
             posp=Avanzar(a,ini+1,fin);
         }
         else
             posp=Retroceder(a,ini,fin);
        }
   return posp;
}

private static int Retroceder(int [] a, int ini, int fin){
   int posp;
   if(ini>=fin)
             posp=ini;
   else{
             if(a[fin]>a[ini+1])
                    posp=Retroceder(a,ini,fin-1);
             else{
                    intercambiar(a,ini+1,fin);
                    posp=Avanzar(a,ini,fin-1);
             }
  }
  return posp;
}
```