Análisis de algoritmos recursivos

Objetivos

- Introducir nociones de Análisis de Algoritmos recursivos, que se profundizarán en otros cursos de la carrera (en particular en la unidad curricular Programación 3).
- Aprender a analizar algoritmos, evaluando si usan eficientemente los recursos del sistema. En particular, aprender a analizar el tiempo de ejecución de los algoritmos, utilizando esta medida para compararlos y evaluar si es posible optimizarlos. Entender cómo el tiempo de ejecución de un algoritmo depende de la entrada y del tamaño de la misma.
- Introducir los conceptos de peor caso, mejor caso y caso promedio.
- Comprender qué es el orden del crecimiento de las funciones y cómo se calcula.
- Familiarizarse con los órdenes más significativos.

Se asume que el tiempo de ejecución de las operaciones elementales es 1 a menos que se especifique de manera explícita.

Análisis de algoritmos recursivos

Ejercicio 1 Imprimir

Considere una lista simplemente encadenada, implementada usando memoria dinámica. ¿Es posible imprimir los elementos de dicha lista de atrás hacia delante en *O(n)*, siendo n el largo de la lista? En caso afirmativo, desarrolle el código de un algoritmo que cumpla la restricción establecida y calcule su orden (O). En otro caso, justifique.

Ejercicio 2 Búsqueda binaria

Calcule el orden Θ del tiempo de ejecución para el peor caso de la función recursiva de búsqueda binaria en un arreglo ordenado.

Ejercicio 3 Recorridas de un árbol

¿Cuál es el orden Θ del tiempo de ejecución en el peor caso de los algoritmos que recorren un árbol binario (preorder, inorder, posorder) de n elementos? Asuma que el acceso a un nodo insume *O(1)*.

Ejercicio 4 K-ésimo

Determine el orden de crecimiento del tiempo de ejecución del siguiente algoritmo, sabiendo que el orden de `cantidad` es $O(n)$, siendo n la cantidad de elementos del árbol. Se asume que el árbol es un ABB.

```
struct nodoArbol{int elem; nodoArbol *izq, *der;};
typedef struct nodoArbol * ABB;

/* Devuelve el kesimo menor elemento de 'a'.
  Precondición 1 <= k <= cantidad(a). */
int kesimo(int k, ABB a) {
  int result;
  int c_izq = cantidad(a->izq);
  if (c_izq == k - 1)
    result = a->elem;
  else if (c_izq < k - 1)
    result = kesimo(k - c_izq - 1, a>der);
  else
    result = kesimo(k, a->izq);
```

```
  return result;
}
```

Describa las instancias en las que se da el peor caso.

Ejercicio 5 Merge Sort

Calcule el orden Θ del tiempo de ejecución para el peor caso del algoritmo de ordenamiento de un arreglo por intercalación (`MergeSort`). Se deben ordenar los elementos del arreglo de menor a mayor. Compare con el ordenamiento de un arreglo por inserción (`Insertion Sort`) del Ejercicio `Insertion Sort` del práctico de Análisis de algoritmos iterativos.

```
void MergeSort (int *A, int inicio, int fin) {
   if (fin-inicio > 1){
        int medio =  (inicio + fin) / 2;
        MergeSort (A,inicio, medio) ;
        MergeSort (A, medio+1, fin);
        Merge (A, inicio, medio, fin);
   }
}
```

Figura 1: Algoritmo `MergeSort` de ordenamiento de un arreglo.

El procedimiento `Merge` actúa sobre el arreglo parámetro intercalando de forma ordenanda dos porciones contiguas indicadas por los índices inicio, medio y fin.

```
void Merge (int *A, int ini, int medio, int fin)
```

Figura 2: Procedimiento de intercalación `Merge` que modifica el arreglo parámetro A ordenando la porción `[ini, fin]`, mediante la intercalación ordenanda de las dos porciones ordenadas delimitadas por `[ini : medio]` y `[medio + 1 : fin]`.

Ejercicio 6 Hanoi

Determine el orden de crecimiento del tiempo de ejecucion del siguiente algoritmo. Asuma que la impresión es $O(1)$.

```
void hanoi(int n, char origen, char destino, char intermedio) {
  if (n > 0) {
    hanoi(n-1, origen, intermedio, destino);
    printf("Mueve %d de %c a %c\n", n, origen, destino);
    hanoi(n-1, intermedio, destino, origen);
  }
}
```

Ejercicio 7 Potencia

Considere el siguiente algoritmo que calcula la potencia $i-$ésima, $i \geq 0$, de un número x, x^i.

```
int Potencia (int x, int i) {
  int potencia = 1;
  for (int j = 1; j <= i; j++)
    potencia = potencia * x;
```

```
    }
    return potencia;
}
```

(a) Calcule el orden de crecimiento Θ del tiempo de ejecución de dicho algoritmo.

(b) ¿Se puede desarrollar otro algoritmo que calcule la potencia y que tenga un orden de crecimiento menor que el de la parte anterior?