Análisis de Algoritmos

Objetivos

- Introducir nociones de Análisis de Algoritmos, 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.
- Analizar algoritmos.
- 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.

Ejercicio 1

Calcule el tiempo de ejecución en el peor caso para cada uno de los siguientes algoritmos. Determine el orden de crecimiento O y Ω del tiempo de ejecución de cada algoritmo.

```
sum = 0;
for (int i=1; i<=n; i++)
  sum++;
```

Algoritmo 1

```
sum = 0;
for (int i=1;i<=n;i++)
 for (int j=1;j<=i;j++)
   sum++;
```

Algoritmo 2

```
sum = 0;
for (int i=1; i<=n; i++)
 for (int j=1;j<=n*n;j++)
   sum++;
```

Algoritmo 3

Ejercicio 2

Considere el siguiente algoritmo:

```
bool F (int *A, int *B, int n) {
  bool b=false;
  for (int i = 0; i < n; i++)
    for (int j = n - 1; j >- 1; j--)
      b = b || (A[i] < B[j]);
  return b;
}
```

(a) ¿Qué calcula la función `F`?

(b) Calcule el tiempo de ejecución para el peor caso de la función `F` y el orden de crecimiento Θ.

(c) Escriba una función `G` que resuelva el mismo problema que la función `F` con orden de crecimiento $\Theta(n)$. Justifique.

Ejercicio 3 Trasponer

El siguiente algoritmo, en el que `A` es un arreglo bidimensional de n filas y n columnas, traspone el arreglo. Se asume $n \geq 1$.

```c
void trasponer(int ** A, int n) {
  for (int i = 1; i < n; i++)
    for (int j = 0; j < i; j++) {
      int swap = A[i][j];
      A[i][j] = A[j][i];
      A[j][i] = swap;
    }
}
```

En este ejercicio se consideran que todas las operaciones elementales tienen tiempo de ejecución 1.

En la siguiene expresión identificamos el tiempo de ejecución de las operaciones mediante los siguientes subíndices:

- 1_a: asignaciones que involucran elementos del arreglo
- 1_o: otras asignaciones
- 1_c: comparaciones
- 1_i: incrementos

Sea $T(n)$ el tiempo de ejecución del algoritmo.

(a) Considere la siguiente expresión

$$T(n) = 1_o + \sum_{i=1}^{n-1}\left(1_c + 1_o + \sum_{j=0}^{i-1}(1_c + 3\cdot 1_a + 1_i) + 1_c + 1_i\right) + 1_c.$$

Justifique la procedencia de cada término.

Desarrolle la expresión haciendo abstracción de la procedencia de cada tiempo (o sea, sin considerar los subíndices).

(b) ¿Cuáles de las siguientes afirmaciones son verdaderas? Justifique.

1. $T(n) = O(n^3)$
2. $T(n) = \Omega(n^2)$
3. $T(n) = O(n)$
4. $T(n) = O(3\,n^2)$
5. $T(n) = \Theta(n^2)$

Ejercicio 4 Insertion Sort

(a) Determine el orden de crecimiento Θ del tiempo de ejecución para el mejor y para el peor caso del siguiente algoritmo de inserción en un arreglo ordenado.

```c
/* Inserta 'elem' en 'A' y deja ordenado A[0 .. pos].
  Precondición: 'A' es un arreglo de tamaño mayor a 'pos' y
  A[0 .. pos - 1] está ordenado.
 */
void insertar (float *A, int pos, float elem) {
  int i = pos - 1;
  while ((i >= 0) && (A[i] > elem)) {
    A[i + 1] = A[i];
    i--;
  }
  A[i+1] = elem;
}
```

(b) Determine el orden de crecimiento Θ del tiempo de ejecución para el mejor y para el peor caso del siguiente algoritmo de ordenamiento.

```
/* Ordena A[0 .. n - 1]. */
void insSort(float *A, int n) {
  for (int i = 1; i < n; i++)
    insertar(A, i - 1, A[i]);
}
```

Ejercicio 5 Sort1_k

Desarrolle un algoritmo, `Sort1_k`, con orden de crecimiento *O(n)* que permita ordenar de menor a mayor un arreglo de tamaño n, cuyos valores están en el rango $[1..k]$, siendo k una constante conocida.

Ejercicio 6 Permutación

(a) Determine el orden de crecimiento Θ del tiempo de ejecución para el mejor y para el peor caso del siguiente algoritmo que evalúa si el arreglo A almacena en el rango 0..n-1 una permutación de los enteros en $\{1, \dots, n\}$.

```
bool esPermutacion(int *A, int n) {
  bool  result = true;
  for (int num = 1; num <= n; num++) {
    bool pertenece = false;
    for (int pos = 0; pos < n; pos++)
      pertenece = pertenece || (A[pos] == num);
    result = result && pertenece;
  };
  return result;
}
```

(b) ¿El algoritmo hace lo que se pide? (¿qué pasa con los repetidos? ¿qué pasa si algún elemento no está?)

(c) Diseñe un algoritmo que mejore el tiempo de ejecución. Describa las instancias en las que se dan los mejores casos y los peores casos. ¿Se modifica el orden de crecimiento del mejor o del peor caso?

Ejercicio 7 Espacio Fibonacci

La noción de orden de crecimiento puede usarse también como medida del espacio de almacenamiento requerido por un algoritmo.

(a) Considere el siguiente algoritmo que calcula `Fibonacci(n)`.

```
int fibonacci(int n) {
  int * fibs = new int[n+1];
  fibs[0] = fibs[1] = 1;
  for (int i = 2; i <= n; i++)
    fibs[i] = fibs[i-1] + fibs[i-2];
  int result = fibs[n];
  delete [] fibs;
  return result;
}
```

Determine el orden de crecimiento Θ del tiempo de ejecución del algoritmo y el orden de crecimiento Θ del espacio requerido.

(b) Diseñe y analice un algoritmo que calcule lo mismo que el de la parte anterior y tenga el mismo orden de tiempo de ejecucion y cuyo orden de espacio requerido sea $O(1)$.

Ejercicio 8

En el siguiente algoritmo `col` es una colección de números de cardinalidad (cantidad de elementos) mayor a n, y `removerMinimo` es una operación que remueve el elemento más chico de una colección.

```
for(int i = 1; i <= n/2; i++)
  removerMinimo (col)
```

(a) Explique si el algoritmo cumple algunos de los siguientes órdenes. $O(n), \Omega(n), \Theta(n)$.

(b) Asumiendo que remover el mínimo es $O(n)$, explique si el algoritmo cumple algunos de los siguientes órdenes. $O(n^2), \Omega(n^2), \Theta(n^2)$.

(c) Asumiendo ahora que remover el mínimo es $O(\log(n))$ discuta que se puede afirmar de los órdenes O, Ω, Θ.