Algoritmos

INDICE

- Complejidad y eficiencia.
- Algoritmos de ordenación.
- Algoritmos de búsqueda.

COMPLEJIDAD Y EFICIENCIA

- ¿Cómo de bueno es cada algoritmo?
- ¿Cuánto tarda en comparación con otros algoritmos?
- Nuestros algoritmos son más **eficientes** a menor complejidad.
- Por complejidad **O()**, entendemos, el **coste (temporal y espacial)** de un algoritmo en realizar una determinada tarea.
- Por ejemplo, la complejidad de un algoritmo que calcule la suma de **N** elementos (**tamaño del problema**) introducidos por el usuario será O(N) = en función de f (N)

Complejidad: Cálculo

- Operaciones **primitivas (asignación, incremento, ...)**: O(1)
- **Secuencia** de instrucciones: máximo de la complejidad de cada instrucción.
- **Condiciones simples**: Tiempo necesario para evaluar la condición más el requerido para ejecutar la consecuencia (peor caso).
- **Condiciones alternativas**: Tiempo necesario para evaluar la condición más el requerido para ejecutar el mayor de los tiempos de las consecuencias (peor caso).
- **Bucle con iteraciones fijas**: multiplicar el número de iteraciones por la complejidad del bloque.
- **Bucle con iteraciones variables**: Igual que el caso anterior, pero poniéndonos en el peor escenario, es decir, mayor número de iteraciones posible.
- **Llamadas a subprogramas**(funciones o métodos): Evaluación de cada parámetro más el del cuerpo del módulo.

Complejidad: Cálculo. Ejemplo:

```java
int suma = 10;
for (int i = 10; i <= 50; ++i)
{
    if (i % 2 == 0)
    {
        suma += i;
    }
}
```

```java
public static boolean esBisiesto(int _year)
{
    boolean esBisiesto = (_year%4 == 0 && _year % 100 != 0) || _year % 400 == 0;
    return esBisiesto;
}
```

```java
int[] a = new int[N];
for (int i = 0; i < a.length; i++)
{
    System.out.println(a[i]);
}
```

```java
public static double sumasSucesivas()
{
    int producto = 0;
    int a,b;
    Scanner input = new Scanner(System.in);
    a = input.nextInt();
    b = input.nextInt();
    do
    {
        producto += a;
        --b;
    } while (b > 0);
    input.close();
    return producto;
}
/*
```

```java
int[][] matriz = new int[N][M];
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < M; j++)
    {
        System.out.println(matriz[i][j]);
    }
}
```

Algoritmos de ordenación

- Ordenación por selección.

```
Definir i, j, aux como Entero
para i=0 hasta n-1 hacer
        para j=i+1 hasta n hacer
                si array[j] < array[i] entonces
                        aux <- array[j];
                        array[j] <- array[i]
                        array[i] <- aux;
                finsi
        finpara
finpara
```

¿Complejidad?

- Ordenación por inserción.

```
para i=1 hasta n hacer
        aux <- array[i];
        j <- i-1;
        mientras ((j>=0) Y (aux<array[j])) hacer
                array[j+1] <- array[j];
                j--;
        finmientras
        array[j+1] <- aux;
finpara
```

¿Complejidad?

Algoritmos de ordenación

- Ordenación por burbuja.

```
Definir i,aux,j como Entero
para i=1 hasta n-1
        para j=0 hasta n-i-1
                si array[j] > array[j+1] entonces
                        aux <- array[j];
                        array[j] <- array[j+1];
                        array[j+1] <- aux;
                 finsi
        finpara
finpara
```

¿Complejidad?

Algoritmos de ordenación

- Ordenación rápida o Quicksort.

```
Algoritmo Quicksort(array,inf,sup)
i<-inf
j<-sup
x<-array [(inf+sup)div 2]
mientras i=<j hacer
        mientras array[i]< x hacer
                i<-i+1
        finmientras
        mientras array[j]>x hacer
              j<- j-1
        finmientras
        si i=<j entonces
              tam<-array[i]
               array[i]<-array[j]
               array[j]<-tam
              i=i+1
               j=j-1
        finsi
finmientras
si inf<j entonces
        Quicksort(array,inf,j)
finsi
si i<sup entonces
        Quicksort(array,i,sup)
finsi
finAlgoritmo
```

¿Complejidad?

Algoritmos de búsqueda

- Búsqueda lineal.

```
Algoritmo bLineal(array,elemento)
pos<- -1
N<-tamaño array
para i <- 0 hasta N hacer
        si array[i] == elemento entonces
                pos <- i;
        finsi
finpara
Devolver pos
finAlgoritmo
```

¿Complejidad?

- Búsqueda binaria.

```
Algoritmo bBinaria(array, elemento)
pos<- -1
inf <- 0
sup <- tamaño array -1
mientras inf=<sup hacer
        centro <- (sup+inf)/2;
        si array[centro]== elemento entonces
                pos <- centro
        sino
                si elmento < array[centro] entonces
                        sup <- centro - 1
                sino
                        inf <- centro +1
                finsi
        finsi
devolver pos
finAlgoritmo
```

Algoritmos: Comparación

- Métrica 1:
  - Para cada uno de lo algoritmos anteriores, añade un contador de comparaciones y otro de asignaciones.
  - Al finalizar la ejecución deberán mostrarse los valores de ambos contadores.
- Métrica 2:
  - Vamos a calcular el tiempo de ejecución de cada algoritmo mediante la directiva: System.currentTimeMillis().
  - Tiempo final menos inicial.