Complejidad computacional

Programación I - UNGS

Complejidad computacional

Complejidad computacional

- ¿Qué es un buen algoritmo?
- Dados dos algoritmos para resolver un mismo problema ¿cuál es mejor?

Repaso de funciones

- Existen muchos "tipos" de funciones:

  Constantes: $f(x) = k$

Repaso de funciones

- Existen muchos “tipos” de funciones:

Constantes:	$f(x) = k$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$

Universidad Nacional de General Sarmiento

Repaso de funciones

- Existen muchos "tipos" de funciones:

Constantes:	$f(x) = k$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$
Lineales:	$f(x) = ax + b$

Repaso de funciones

- Existen muchos “tipos” de funciones:

Constantes:	$f(x) = k$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$
Lineales:	$f(x) = ax + b$
Cuadráticas:	$f(x) = ax^2 + bx + c$

Repaso de funciones

- Existen muchos “tipos” de funciones:

Constantes:	$f(x) = k$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$
Lineales:	$f(x) = ax + b$
Cuadráticas:	$f(x) = ax^2 + bx + c$
Cúbicas:	$f(x) = ax^3 + bx^2 + cx + d$

Repaso de funciones

- Existen muchos “tipos” de funciones:

Constantes:	$f(x) = k$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$
Lineales:	$f(x) = ax + b$
Cuadráticas:	$f(x) = ax^2 + bx + c$
Cúbicas:	$f(x) = ax^3 + bx^2 + cx + d$
Exponenciales:	$f(x) = a \cdot b^x + d$

Repaso de funciones

- Existen muchos “tipos” de funciones:

Constantes:	$f(x) = k$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$
Lineales:	$f(x) = ax + b$
Cuadráticas:	$f(x) = ax^2 + bx + c$
Cúbicas:	$f(x) = ax^3 + bx^2 + cx + d$
Exponenciales:	$f(x) = a \cdot b^x + d$
Factorial:	$f(n) = a \cdot n! + b$

Repaso de funciones

- Existen muchos “tipos” de funciones:

Constantes:	$f(x) = k$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$
Lineales:	$f(x) = ax + b$
Cuadráticas:	$f(x) = ax^2 + bx + c$
Cúbicas:	$f(x) = ax^3 + bx^2 + cx + d$
Exponenciales:	$f(x) = a \cdot b^x + d$
Factorial:	$f(n) = a \cdot n! + b$
etc. . .	

Repaso de funciones

- Existen muchos "tipos" de funciones:

Constantes:	$f(x) = k$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$
Lineales:	$f(x) = ax + b$
Cuadráticas:	$f(x) = ax^2 + bx + c$
Cúbicas:	$f(x) = ax^3 + bx^2 + cx + d$
Exponenciales:	$f(x) = a \cdot b^x + d$
Factorial:	$f(n) = a \cdot n! + b$
etc. . .	

- Vamos a usar funciones para "medir" la complejidad de los algoritmos. . . pero antes vamos a catalogar las funciones según su tipo.

Notación "O grande"

- Esta notación nos permite agrupar las funciones que sean "parecidas" (del mismo tipo):

Constantes: $f(x) = k \quad \Rightarrow f \in O(1)$

Notación "O grande"

- Esta notación nos permite agrupar las funciones que sean "parecidas" (del mismo tipo):

Constantes:	$f(x) = k$	$\Rightarrow f \in O(1)$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$	$\Rightarrow f \in O(\log(x))$

Notación "O grande"

- Esta notación nos permite agrupar las funciones que sean "parecidas" (del mismo tipo):

Constantes:	$f(x) = k$	$\Rightarrow f \in O(1)$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$	$\Rightarrow f \in O(\log(x))$
Lineales:	$f(x) = ax + b$	$\Rightarrow f \in O(x)$

Notación "O grande"

- Esta notación nos permite agrupar las funciones que sean "parecidas" (del mismo tipo):

Constantes:	$f(x) = k$	$\Rightarrow f \in O(1)$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$	$\Rightarrow f \in O(\log(x))$
Lineales:	$f(x) = ax + b$	$\Rightarrow f \in O(x)$
Cuadráticas:	$f(x) = ax^2 + bx + c$	$\Rightarrow f \in O(x^2)$

Notación "O grande"

- Esta notación nos permite agrupar las funciones que sean "parecidas" (del mismo tipo):

Constantes:	$f(x) = k$	$\Rightarrow f \in O(1)$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$	$\Rightarrow f \in O(\log(x))$
Lineales:	$f(x) = ax + b$	$\Rightarrow f \in O(x)$
Cuadráticas:	$f(x) = ax^2 + bx + c$	$\Rightarrow f \in O(x^2)$
Cúbicas:	$f(x) = ax^3 + bx^2 + cx + d$	$\Rightarrow f \in O(x^3)$

Notación "O grande"

- Esta notación nos permite agrupar las funciones que sean "parecidas" (del mismo tipo):

Constantes:	$f(x) = k$	$\Rightarrow f \in O(1)$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$	$\Rightarrow f \in O(\log(x))$
Lineales:	$f(x) = ax + b$	$\Rightarrow f \in O(x)$
Cuadráticas:	$f(x) = ax^2 + bx + c$	$\Rightarrow f \in O(x^2)$
Cúbicas:	$f(x) = ax^3 + bx^2 + cx + d$	$\Rightarrow f \in O(x^3)$
Exponenciales:	$f(x) = a \cdot b^x + d$	$\Rightarrow f \in O(b^x)$

Notación “O grande”

- Esta notación nos permite agrupar las funciones que sean “parecidas” (del mismo tipo):

Constantes:	$f(x) = k$	$\Rightarrow f \in O(1)$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$	$\Rightarrow f \in O(\log(x))$
Lineales:	$f(x) = ax + b$	$\Rightarrow f \in O(x)$
Cuadráticas:	$f(x) = ax^2 + bx + c$	$\Rightarrow f \in O(x^2)$
Cúbicas:	$f(x) = ax^3 + bx^2 + cx + d$	$\Rightarrow f \in O(x^3)$
Exponenciales:	$f(x) = a \cdot b^x + d$	$\Rightarrow f \in O(b^x)$
Factorial:	$f(n) = a \cdot n! + b$	$\Rightarrow f \in O(n!)$

Notación "O grande"

- Esta notación nos permite agrupar las funciones que sean "parecidas" (del mismo tipo):

Constantes:	$f(x) = k$	$\Rightarrow f \in O(1)$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$	$\Rightarrow f \in O(\log(x))$
Lineales:	$f(x) = ax + b$	$\Rightarrow f \in O(x)$
Cuadráticas:	$f(x) = ax^2 + bx + c$	$\Rightarrow f \in O(x^2)$
Cúbicas:	$f(x) = ax^3 + bx^2 + cx + d$	$\Rightarrow f \in O(x^3)$
Exponenciales:	$f(x) = a \cdot b^x + d$	$\Rightarrow f \in O(b^x)$
Factorial:	$f(n) = a \cdot n! + b$	$\Rightarrow f \in O(n!)$
etc...		

Notación "O grande"

- Esta notación nos permite agrupar las funciones que sean "parecidas" (del mismo tipo):

Constantes:	$f(x) = k$	$\Rightarrow f \in O(1)$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$	$\Rightarrow f \in O(\log(x))$
Lineales:	$f(x) = ax + b$	$\Rightarrow f \in O(x)$
Cuadráticas:	$f(x) = ax^2 + bx + c$	$\Rightarrow f \in O(x^2)$
Cúbicas:	$f(x) = ax^3 + bx^2 + cx + d$	$\Rightarrow f \in O(x^3)$
Exponenciales:	$f(x) = a \cdot b^x + d$	$\Rightarrow f \in O(b^x)$
Factorial:	$f(n) = a \cdot n! + b$	$\Rightarrow f \in O(n!)$
etc...		

- Ojo! Estos conjuntos no son disjuntos! Por ejemplo:

$$O(x) \subset O(x^2) \subset O(x^3) \subset O(b^x).$$

Universidad Nacional de General Sarmiento

Notación "O grande"

- Esta notación nos permite agrupar las funciones que sean "parecidas" (del mismo tipo):

Constantes:	$f(x) = k$	$\Rightarrow f \in O(1)$
Logarítmicas:	$f(x) = a \cdot \log(x) + b$	$\Rightarrow f \in O(\log(x))$
Lineales:	$f(x) = ax + b$	$\Rightarrow f \in O(x)$
Cuadráticas:	$f(x) = ax^2 + bx + c$	$\Rightarrow f \in O(x^2)$
Cúbicas:	$f(x) = ax^3 + bx^2 + cx + d$	$\Rightarrow f \in O(x^3)$
Exponenciales:	$f(x) = a \cdot b^x + d$	$\Rightarrow f \in O(b^x)$
Factorial:	$f(n) = a \cdot n! + b$	$\Rightarrow f \in O(n!)$
etc...		

- Ojo! Estos conjuntos no son disjuntos! Por ejemplo:

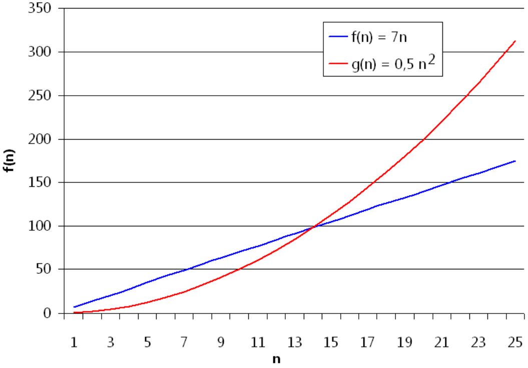
$$O(x) \subset O(x^2) \subset O(x^3) \subset O(b^x).$$

Veamos un poco esto en detalle...

Notación "O grande" - Definición formal

- **Definición:** Si f y g son dos funciones, decimos que $f \in O(g)$ si existen $\alpha \in \mathbb{R}$ y $n_0 \in \mathbb{N}$ tales que

$$f(n) \leq \alpha \cdot g(n) \qquad \text{para todo } n \geq n_0.$$



Notación "O grande"

- Intuitivamente, $f \in O(g)$ si $\alpha \cdot g(n)$ "le gana" a $f(n)$ para valores grandes de n, para algún α cualquiera fijo.

Notación “O grande”

- Intuitivamente, $f \in O(g)$ si $\alpha \cdot g(n)$ “le gana” a $f(n)$ para valores grandes de n, para algún α cualquiera fijo.
- Ejemplos:
  - Si $f(n) = n$ y $g(n) = n^2$, entonces...

Universidad Nacional de General Sarmiento

Notación "O grande"

- Intuitivamente, $f \in O(g)$ si $\alpha \cdot g(n)$ "le gana" a $f(n)$ para valores grandes de n, para algún α cualquiera fijo.
- Ejemplos:
  - Si $f(n) = n$ y $g(n) = n^2$, entonces... $f \in O(g)$.

Notación "O grande"

- Intuitivamente, $f \in O(g)$ si $\alpha \cdot g(n)$ "le gana" a $f(n)$ para valores grandes de n, para algún α cualquiera fijo.
- Ejemplos:
  - Si $f(n) = n$ y $g(n) = n^2$, entonces... $f \in O(g)$.
  - Si $f(n) = n^2$ y $g(n) = n$, entonces...

Notación "O grande"

- Intuitivamente, $f \in O(g)$ si $\alpha \cdot g(n)$ "le gana" a $f(n)$ para valores grandes de n, para algún α cualquiera fijo.
- Ejemplos:
  - Si $f(n) = n$ y $g(n) = n^2$, entonces... $f \in O(g)$.
  - Si $f(n) = n^2$ y $g(n) = n$, entonces... $f \notin O(g)$.

Notación "O grande"

- Intuitivamente, $f \in O(g)$ si $\alpha \cdot g(n)$ "le gana" a $f(n)$ para valores grandes de n, para algún α cualquiera fijo.

- Ejemplos:
  - Si $f(n) = n$ y $g(n) = n^2$, entonces... $f \in O(g)$.
  - Si $f(n) = n^2$ y $g(n) = n$, entonces... $f \notin O(g)$.
  - Si $f(n) = 100n$ y $g(n) = n^2$, entonces...

Notación “O grande”

- Intuitivamente, $f \in O(g)$ si $\alpha \cdot g(n)$ “le gana” a $f(n)$ para valores grandes de n, para algún α cualquiera fijo.
- Ejemplos:
    - Si $f(n) = n$ y $g(n) = n^2$, entonces... $f \in O(g)$.
    - Si $f(n) = n^2$ y $g(n) = n$, entonces... $f \notin O(g)$.
    - Si $f(n) = 100n$ y $g(n) = n^2$, entonces... $f \in O(g)$.

Notación "O grande"

- Intuitivamente, $f \in O(g)$ si $\alpha \cdot g(n)$ "le gana" a $f(n)$ para valores grandes de n, para algún α cualquiera fijo.
- Ejemplos:
    - Si $f(n) = n$ y $g(n) = n^2$, entonces... $f \in O(g)$.
    - Si $f(n) = n^2$ y $g(n) = n$, entonces... $f \notin O(g)$.
    - Si $f(n) = 100n$ y $g(n) = n^2$, entonces... $f \in O(g)$.
    - Si $f(n) = 4n^2$ y $g(n) = 2n^2$, entonces...

Notación "O grande"

- Intuitivamente, $f \in O(g)$ si $\alpha \cdot g(n)$ "le gana" a $f(n)$ para valores grandes de n, para algún α cualquiera fijo.
- Ejemplos:
  - Si $f(n) = n$ y $g(n) = n^2$, entonces... $f \in O(g)$.
  - Si $f(n) = n^2$ y $g(n) = n$, entonces... $f \notin O(g)$.
  - Si $f(n) = 100n$ y $g(n) = n^2$, entonces... $f \in O(g)$.
  - Si $f(n) = 4n^2$ y $g(n) = 2n^2$, entonces... $f \in O(g)$ (y a la inversa).

Universidad Nacional de General Sarmiento

Notación "O grande"

- Intuitivamente, $f \in O(g)$ si $\alpha \cdot g(n)$ "le gana" a $f(n)$ para valores grandes de n, para algún α cualquiera fijo.
- Ejemplos:
  - Si $f(n) = n$ y $g(n) = n^2$, entonces... $f \in O(g)$.
  - Si $f(n) = n^2$ y $g(n) = n$, entonces... $f \notin O(g)$.
  - Si $f(n) = 100n$ y $g(n) = n^2$, entonces... $f \in O(g)$.
  - Si $f(n) = 4n^2$ y $g(n) = 2n^2$, entonces... $f \in O(g)$ (y a la inversa).
- Como dijimos, la idea es agrupar funciones "parecidas"...

Complejidad computacional

Complejidad computacional

- Una forma habitual de medir el tiempo de ejecución de un algoritmo es evaluar el peor caso en función del tamaño de la entrada del algoritmo.

Universidad Nacional de General Sarmiento

Complejidad computacional

- Una forma habitual de medir el tiempo de ejecución de un algoritmo es evaluar el peor caso en función del tamaño de la entrada del algoritmo.
- **Definición:** La función de complejidad de un algoritmo es una función $f : \mathbb{N} \to \mathbb{N}$ tal que $f(n)$ es la cantidad de operaciones que realiza el algoritmo en el peor caso cuando toma una entrada de tamaño n.

Complejidad computacional

- Una forma habitual de medir el tiempo de ejecución de un algoritmo es evaluar el peor caso en función del tamaño de la entrada del algoritmo.
- **Definición:** La función de complejidad de un algoritmo es una función $f : \mathbb{N} \to \mathbb{N}$ tal que $f(n)$ es la cantidad de operaciones que realiza el algoritmo en el peor caso cuando toma una entrada de tamaño n.
- Algunas observaciones:
  1. Medimos la cantidad de operaciones en lugar del tiempo (¿por qué?).
  2. Nos interesa el peor caso del algoritmo.
  3. La complejidad se mide en función del tamaño de la entrada y no de la entrada particular.

Complejidad computacional

- Utilizamos la notación "O grande" para especificar la función de complejidad f de los algoritmos.

Complejidad computacional

- Utilizamos la notación "O grande" para especificar la función de complejidad f de los algoritmos.
    - Si $f \in O(1)$, decimos que el algoritmo es constante.

Complejidad computacional

- Utilizamos la notación "O grande" para especificar la función de complejidad f de los algoritmos.
  - Si $f \in O(1)$, decimos que el algoritmo es constante.
  - Si $f \in O(\log(n))$, decimos que el algoritmo es logarítmico.

Complejidad computacional

- Utilizamos la notación “O grande” para especificar la función de complejidad f de los algoritmos.
  - Si $f \in O(1)$, decimos que el algoritmo es constante.
  - Si $f \in O(\log(n))$, decimos que el algoritmo es logarítmico.
  - Si $f \in O(n)$, decimos que el algoritmo es lineal.

Complejidad computacional

- Utilizamos la notación "O grande" para especificar la función de complejidad f de los algoritmos.
  - Si $f \in O(1)$, decimos que el algoritmo es constante.
  - Si $f \in O(\log(n))$, decimos que el algoritmo es logarítmico.
  - Si $f \in O(n)$, decimos que el algoritmo es lineal.
  - Si $f \in O(n^2)$, decimos que el algoritmo es cuadrático.

Complejidad computacional

- Utilizamos la notación "O grande" para especificar la función de complejidad f de los algoritmos.
  - Si $f \in O(1)$, decimos que el algoritmo es constante.
  - Si $f \in O(\log(n))$, decimos que el algoritmo es logarítmico.
  - Si $f \in O(n)$, decimos que el algoritmo es lineal.
  - Si $f \in O(n^2)$, decimos que el algoritmo es cuadrático.
  - Si $f \in O(n^3)$, decimos que el algoritmo es cúbico.

Complejidad computacional

- Utilizamos la notación "O grande" para especificar la función de complejidad f de los algoritmos.
  - Si $f \in O(1)$, decimos que el algoritmo es constante.
  - Si $f \in O(\log(n))$, decimos que el algoritmo es logarítmico.
  - Si $f \in O(n)$, decimos que el algoritmo es lineal.
  - Si $f \in O(n^2)$, decimos que el algoritmo es cuadrático.
  - Si $f \in O(n^3)$, decimos que el algoritmo es cúbico.
  - En general, si $f \in O(n^k)$, decimos que el algoritmo es polinomial.

Complejidad computacional

- Utilizamos la notación “O grande” para especificar la función de complejidad f de los algoritmos.
  - Si $f \in O(1)$, decimos que el algoritmo es constante.
  - Si $f \in O(\log(n))$, decimos que el algoritmo es logarítmico.
  - Si $f \in O(n)$, decimos que el algoritmo es lineal.
  - Si $f \in O(n^2)$, decimos que el algoritmo es cuadrático.
  - Si $f \in O(n^3)$, decimos que el algoritmo es cúbico.
  - En general, si $f \in O(n^k)$, decimos que el algoritmo es polinomial.
  - Si $f \in O(2^n)$, decimos que el algoritmo es exponencial.

Universidad Nacional de General Sarmiento

Complejidad computacional

- Utilizamos la notación "O grande" para especificar la función de complejidad f de los algoritmos.
  - Si $f \in O(1)$, decimos que el algoritmo es constante.
  - Si $f \in O(\log(n))$, decimos que el algoritmo es logarítmico.
  - Si $f \in O(n)$, decimos que el algoritmo es lineal.
  - Si $f \in O(n^2)$, decimos que el algoritmo es cuadrático.
  - Si $f \in O(n^3)$, decimos que el algoritmo es cúbico.
  - En general, si $f \in O(n^k)$, decimos que el algoritmo es polinomial.
  - Si $f \in O(2^n)$, decimos que el algoritmo es exponencial.
  - Si $f \in O(n!)$, decimos que el algoritmo es factorial.

Complejidad computacional

- Utilizamos la notación "O grande" para especificar la función de complejidad f de los algoritmos.
  - Si $f \in O(1)$, decimos que el algoritmo es constante.
  - Si $f \in O(\log(n))$, decimos que el algoritmo es logarítmico.
  - Si $f \in O(n)$, decimos que el algoritmo es lineal.
  - Si $f \in O(n^2)$, decimos que el algoritmo es cuadrático.
  - Si $f \in O(n^3)$, decimos que el algoritmo es cúbico.
  - En general, si $f \in O(n^k)$, decimos que el algoritmo es polinomial.
  - Si $f \in O(2^n)$, decimos que el algoritmo es exponencial.
  - Si $f \in O(n!)$, decimos que el algoritmo es factorial.
  - etc...

Complejidad computacional

- Utilizamos la notación “O grande” para especificar la función de complejidad f de los algoritmos.
    - Si $f \in O(1)$, decimos que el algoritmo es constante.
    - Si $f \in O(\log(n))$, decimos que el algoritmo es logarítmico.
    - Si $f \in O(n)$, decimos que el algoritmo es lineal.
    - Si $f \in O(n^2)$, decimos que el algoritmo es cuadrático.
    - Si $f \in O(n^3)$, decimos que el algoritmo es cúbico.
    - En general, si $f \in O(n^k)$, decimos que el algoritmo es polinomial.
    - Si $f \in O(2^n)$, decimos que el algoritmo es exponencial.
    - Si $f \in O(n!)$, decimos que el algoritmo es factorial.
    - etc. . .
- Veamos un ejemplo. . .

Un ejemplo

```
public static int maximo(int[] a) {
    int max = a[0];

    for (int i = 0; i < a.length; i++) {
        if (a[i] < max) {
            max = a[i];
        }
    }
    return max;
}
```

Un ejemplo

```
public static int maximo(int[] a) {
    int max = a[0];

    for (int i = 0; i < a.length; i++) {
        if (a[i] < max) {
            max = a[i];
        }
    }
    return max;
}
```

- ¿Cuál es la entrada? ¿Cuál es el tamaño de entrada?

Un ejemplo

```
public static int maximo(int[] a) {
    int max = a[0];

    for (int i = 0; i < a.length; i++) {
        if (a[i] < max) {
            max = a[i];
        }
    }
    return max;
}
```

- ¿Cuál es la entrada? ¿Cuál es el tamaño de entrada?
- ¿Cuántas operaciones realiza en el peor caso?

Un ejemplo

```
public static int maximo(int[] a) {
    int max = a[0];

    for (int i = 0; i < a.length; i++) {
        if (a[i] < max) {
            max = a[i];
        }
    }
    return max;
}
```

- ¿Cuál es la entrada? ¿Cuál es el tamaño de entrada?
- ¿Cuántas operaciones realiza en el peor caso?
- ¿Qué tipo de complejidad tiene $f(n)$?

Otro ejemplo

```
public static boolean tieneDuplicados(int[] a) {
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a.length; j++) {
            if (i != j && a[i] == a[j]) {
                return true;
            }
        }
    }
    return false;
}
```

Otro ejemplo

```
public static boolean tieneDuplicados(int[] a) {
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a.length; j++) {
            if (i != j && a[i] == a[j]) {
                return true;
            }
        }
    }
    return false;
}
```

- ¿Cuál es la entrada? ¿Cuál es el tamaño de entrada?

Universidad Nacional de General Sarmiento

Otro ejemplo

```
public static boolean tieneDuplicados(int[] a) {
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a.length; j++) {
            if (i != j && a[i] == a[j]) {
                return true;
            }
        }
    }
    return false;
}
```

- ¿Cuál es la entrada? ¿Cuál es el tamaño de entrada?
- ¿Cuántas operaciones realiza en el peor caso?

Otro ejemplo

```
public static boolean tieneDuplicados(int[] a) {
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a.length; j++) {
            if (i != j && a[i] == a[j]) {
                return true;
            }
        }
    }
    return false;
}
```

- ¿Cuál es la entrada? ¿Cuál es el tamaño de entrada?
- ¿Cuántas operaciones realiza en el peor caso?
- ¿Qué tipo de complejidad tiene $f(n)$?

Buscando un elemento en un arreglo

```
public static boolean buscar(int[] a, int b) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == b) {
            return true;
        }
    }
    return false;
}
```

Buscando un elemento en un arreglo

```
public static boolean buscar(int[] a, int b) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == b) {
            return true;
        }
    }
    return false;
}
```

- ¿Cuál es el tamaño de entrada?

Buscando un elemento en un arreglo

```
public static boolean buscar(int[] a, int b) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == b) {
            return true;
        }
    }
    return false;
}
```

- ¿Cuál es el tamaño de entrada?
- ¿Cuántas operaciones realiza en el peor caso?

Buscando un elemento en un arreglo

```
public static boolean buscar(int[] a, int b) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == b) {
            return true;
        }
    }
    return false;
}
```

- ¿Cuál es el tamaño de entrada?
- ¿Cuántas operaciones realiza en el peor caso?
- ¿Qué tipo de complejidad tiene $f(n)$?

Buscando un elemento en un arreglo

```
public static boolean buscar(int[] a, int b) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == b) {
            return true;
        }
    }
    return false;
}
```

- ¿Cuál es el tamaño de entrada?
- ¿Cuántas operaciones realiza en el peor caso?
- ¿Qué tipo de complejidad tiene $f(n)$?
- ¿Se puede hacer con una mejor complejidad?

Búsqueda binaria

Supongamos que el arreglo está ordenado:

2	10	11	12	15	20	30	31	38	50

¿Cómo haríamos para buscar el número 17 en este arreglo?

Búsqueda binaria

```
public static boolean busquedaBinaria(int[] a, int b) {
    int izq = 0, der = a.length - 1;

    while (izq + 1 < der) {
        int m = (izq + der) / 2;
        if (a[m] == b) {
            return true;
        }
        if (a[m] < b) {
            izq = m;
        } else {
            der = m;
        }
    }
    return a[izq] == b || a[der] == b;
}
```

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?

- El algoritmo termina cuando $der - izq \leq 1$ y en cada iteración se reduce a la mitad. ¿Cuánto vale $der - izq$ en cada iteración?

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?
- El algoritmo termina cuando $der - izq \leq 1$ y en cada iteración se reduce a la mitad. ¿Cuánto vale $der - izq$ en cada iteración?

Iteración	$der - izq$
0	$(n-1)$

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?

- El algoritmo termina cuando $der - izq \leq 1$ y en cada iteración se reduce a la mitad. ¿Cuánto vale $der - izq$ en cada iteración?

Iteración	$der - izq$
0	$(n-1)$
1	$(n-1)/2$

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?

- El algoritmo termina cuando $der - izq \leq 1$ y en cada iteración se reduce a la mitad. ¿Cuánto vale $der - izq$ en cada iteración?

Iteración	$der - izq$
0	$(n-1)$
1	$(n-1)/2$
2	$(n-1)/4$

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?

- El algoritmo termina cuando $der - izq \leq 1$ y en cada iteración se reduce a la mitad. ¿Cuánto vale $der - izq$ en cada iteración?

Iteración	$der - izq$
0	$(n-1)$
1	$(n-1)/2$
2	$(n-1)/4$
3	$(n-1)/8$

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?
- El algoritmo termina cuando $der - izq \leq 1$ y en cada iteración se reduce a la mitad. ¿Cuánto vale $der - izq$ en cada iteración?

Iteración	$der - izq$
0	$(n-1)$
1	$(n-1)/2$
2	$(n-1)/4$
3	$(n-1)/8$
$\vdots$	$\vdots$
k	$(n-1)/2^k$

Universidad Nacional de General Sarmiento

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?

- El algoritmo termina cuando $der - izq \leq 1$ y en cada iteración se reduce a la mitad. ¿Cuánto vale $der - izq$ en cada iteración?

Iteración	$der - izq$
0	$(n-1)$
1	$(n-1)/2$
2	$(n-1)/4$
3	$(n-1)/8$
$\vdots$	$\vdots$
k	$(n-1)/2^k$

$$der - izq = 1 \quad \Longrightarrow \quad \frac{n-1}{2^k} = 1$$

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?
- El algoritmo termina cuando $der - izq \leq 1$ y en cada iteración se reduce a la mitad. ¿Cuánto vale $der - izq$ en cada iteración?

Iteración	$der - izq$
0	$(n-1)$
1	$(n-1)/2$
2	$(n-1)/4$
3	$(n-1)/8$
$\vdots$	$\vdots$
k	$(n-1)/2^k$

$$der - izq = 1 \implies \frac{n-1}{2^k} = 1$$
$$\implies n - 1 = 2^k$$

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?
- El algoritmo termina cuando $der - izq \leq 1$ y en cada iteración se reduce a la mitad. ¿Cuánto vale $der - izq$ en cada iteración?

Iteración	$der - izq$
0	$(n-1)$
1	$(n-1)/2$
2	$(n-1)/4$
3	$(n-1)/8$
$\vdots$	$\vdots$
k	$(n-1)/2^k$

$$der - izq = 1 \implies \frac{n-1}{2^k} = 1$$

$$\implies n - 1 = 2^k$$

$$\implies \log_2(n-1) = k$$

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?
- El algoritmo termina cuando $der - izq \leq 1$ y en cada iteración se reduce a la mitad. ¿Cuánto vale $der - izq$ en cada iteración?

Iteración	$der - izq$
0	$(n-1)$
1	$(n-1)/2$
2	$(n-1)/4$
3	$(n-1)/8$
$\vdots$	$\vdots$
k	$(n-1)/2^k$

$$der - izq = 1 \implies \frac{n-1}{2^k} = 1$$
$$\implies n - 1 = 2^k$$
$$\implies \log_2(n-1) = k$$

O sea que el algoritmo realiza $\log_2(n-1)$ iteraciones...

Búsqueda binaria

- ¿Cuántas iteraciones realiza el algoritmo de búsqueda binaria?

- El algoritmo termina cuando $der - izq \leq 1$ y en cada iteración se reduce a la mitad. ¿Cuánto vale $der - izq$ en cada iteración?

Iteración	$der - izq$
0	$(n-1)$
1	$(n-1)/2$
2	$(n-1)/4$
3	$(n-1)/8$
$\vdots$	$\vdots$
k	$(n-1)/2^k$

$$\begin{aligned} der - izq = 1 \quad &\Longrightarrow \quad \frac{n-1}{2^k} = 1 \\ &\Longrightarrow \quad n-1 = 2^k \\ &\Longrightarrow \quad \log_2(n-1) = k \end{aligned}$$

O sea que el algoritmo realiza $\log_2(n-1)$ iteraciones...

- La función de complejidad del algoritmo de búsqueda binaria es $f \in O(\log n)$. Decimos que el algoritmo es logarítmico.

Búsqueda binaria

- ¿Es bueno un algoritmo con complejidad logarítmica?

Búsqueda binaria

- ¿Es bueno un algoritmo con complejidad logarítmica?

n	Búsqueda Lineal	Búsqueda Binaria
10	10	4

Búsqueda binaria

- ¿Es bueno un algoritmo con complejidad logarítmica?

n	Búsqueda Lineal	Búsqueda Binaria
10	10	4
10^2	100	7

Búsqueda binaria

- ¿Es bueno un algoritmo con complejidad logarítmica?

n	Búsqueda Lineal	Búsqueda Binaria
10	10	4
10^2	100	7
10^6	1 000 000	21

Búsqueda binaria

- ¿Es bueno un algoritmo con complejidad logarítmica?

n	Búsqueda Lineal	Búsqueda Binaria
10	10	4
10^2	100	7
10^6	1 000 000	21
2.3×10^7	23 000 000	25

Búsqueda binaria

- ¿Es bueno un algoritmo con complejidad logarítmica?

n	Búsqueda Lineal	Búsqueda Binaria
10	10	4
10^2	100	7
10^6	1 000 000	21
2.3×10^7	23 000 000	25
7×10^9	7 000 000 000	33 (!)

Búsqueda binaria

- ¿Es bueno un algoritmo con complejidad logarítmica?

n	Búsqueda Lineal	Búsqueda Binaria
10	10	4
10^2	100	7
10^6	1 000 000	21
2.3×10^7	23 000 000	25
7×10^9	7 000 000 000	33 (!)

- Sí! Un algoritmo con este orden es muy eficiente.

En el libro. . .

Si quieren leer un poco más sobre el algoritmo de búsqueda binaria, pueden hacerlo en la **Sección 11.8** del libro. Pueden leer sobre eficiencia y complejidad en el **Capítulo 18** del libro.