Introducción a la Computación (Matemática)

Primer Cuatrimestre de 2018

Algoritmos de Búsqueda

Tiempo de ejecución

El tiempo de ejecución de un programa se mide en función del tamaño de la entrada.

- Ejemplo: longitud de la lista de entrada.

Notación: $T(n)$: tiempo de ejecución de un programa con una entrada de tamaño n.

- Unidad: cantidad de instrucciones.
- Ejemplo: $T(n) = c \cdot n^2$, donde c es una constante.

Consideramos el peor caso: $T(n)$ es una cota superior del tiempo de ejecución para entradas arbitrarias de tamaño n.

Cálculo del tiempo de ejecución

Instrucciones minimales: acceso a una variable (asignación o consulta) y operaciones simples de tipos básicos. $T_S(n) = 1$

Secuencialización: $T_{S_1;S_2}(n) = T_{S_1}(n) + T_{S_2}(n)$

Condicional:
$T_{\text{if } (B)\ S_1 \text{ else } S_2}(n) = T_B(n) + \max(T_{S_1}(n), T_{S_2}(n))$

Ciclo: $T_{\text{while } (B)\ S}(n) = T_B(n) + \Sigma_{i \in \text{Iteraciones}} \left(T_{S_i}(n) + T_B(n)\right)$

Orden del tiempo de ejecución

En general, decimos que $T(n) \in O(f(n))$ si existen constantes enteras positivas c y n_0 tales que para $n \geq n_0$, $T(n) \leq c \cdot f(n)$.

Ejemplo: $T(n) = 3n^3 + 2n^2$.

$T(n) \in O(n^3)$, dado que si tomamos $n_0 = 1$ y $c = 5$, vale que para $n \geq 1$, $T(n) \leq 5 \cdot n^3$.

Ejemplo 1.1: $T(|A|) = 5 + \frac{25}{2}|A| + \frac{11}{2}|A|^2 \in O(|A|^2)$ (orden cuadrático)

Ejemplo 1.2: $T(|A|) = 6 + 18|A| \in O(|A|)$ (orden lineal)

Nota: Si $T(n) = cte.$ entonces $T(n) \in O(1)$ (orden constante)

Complejidad temporal

Propiedades de O:

- Regla de la suma:
  Si $f_1 \in O(g)$ y $f_2 \in O(h) \Rightarrow f_1 + f_2 \in O(\text{máx}(g, h))$.

Complejidad temporal

Propiedades de O:

- Regla de la suma:
  Si $f_1 \in O(g)$ y $f_2 \in O(h) \Rightarrow f_1 + f_2 \in O(\max(g, h))$.
  - Ej: $f_1 \in O(n^2)$ y $f_2 \in O(n)$, luego $f_1 + f_2 \in O(n^2)$.
  - Ej: $f_1 \in O(1)$ y $f_2 \in O(1)$, luego $f_1 + f_2 \in O(1)$.

Complejidad temporal

Propiedades de O:

- Regla de la suma:
  Si $f_1 \in O(g)$ y $f_2 \in O(h) \Rightarrow f_1 + f_2 \in O(\text{máx}(g, h))$.
  - Ej: $f_1 \in O(n^2)$ y $f_2 \in O(n)$, luego $f_1 + f_2 \in O(n^2)$.
  - Ej: $f_1 \in O(1)$ y $f_2 \in O(1)$, luego $f_1 + f_2 \in O(1)$.

- Regla del producto:
  Si $f_1 \in O(g)$ y $f_2 \in O(h) \Rightarrow f_1 \cdot f_2 \in O(g \cdot h)$.

Complejidad temporal

Propiedades de O:

- Regla de la suma:
  Si $f_1 \in O(g)$ y $f_2 \in O(h) \Rightarrow f_1 + f_2 \in O(\text{máx}(g, h))$.
  - Ej: $f_1 \in O(n^2)$ y $f_2 \in O(n)$, luego $f_1 + f_2 \in O(n^2)$.
  - Ej: $f_1 \in O(1)$ y $f_2 \in O(1)$, luego $f_1 + f_2 \in O(1)$.

- Regla del producto:
  Si $f_1 \in O(g)$ y $f_2 \in O(h) \Rightarrow f_1 \cdot f_2 \in O(g \cdot h)$.
  - Ej: $f_1 \in O(n^2)$ y $f_2 \in O(n)$, luego $f_1 \cdot f_2 \in O(n^3)$.
  - Ej: $f_1 \in O(n)$ y $f_2 \in O(1)$, luego $f_1 \cdot f_2 \in O(n)$.

Problema de búsqueda

Encabezado: $\mathit{Buscar} : x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \rightarrow está \in \mathbb{B} \times pos \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0\}$

Poscondición: $\{(está = true \wedge 0 \leq pos < |A_0| \wedge A_0[pos] = x_0) \vee$
$(está = false \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

Problema de búsqueda

Encabezado: *Buscar* : $x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \rightarrow está \in \mathbb{B} \times pos \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0\}$

Poscondición: $\{(está = true \wedge 0 \leq pos < |A_0| \wedge A_0[pos] = x_0) \vee$
$(está = false \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

$está \leftarrow false$
$pos \leftarrow -1$
$j \leftarrow 0$
while $(j < |A|)$ {
    if $(A[j] = x)$ {
        $está \leftarrow$ true
        $pos \leftarrow j$
    }
    $j \leftarrow j + 1$
}

Problema de búsqueda

Encabezado: *Buscar* : $x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \rightarrow está \in \mathbb{B} \times pos \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0\}$

Poscondición: $\{(está = true \wedge 0 \leq pos < |A_0| \wedge A_0[pos] = x_0) \vee$
$(está = false \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

```
está ← false
pos ← −1
j ← 0
while (j < |A|) {
        if (A[j] = x) {
            está ← true
            pos ← j
        }
        j ← j + 1
}
```

¿Cuál es el orden de complejidad?

Problema de búsqueda

Encabezado: *Buscar* : $x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \rightarrow está \in \mathbb{B} \times pos \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0\}$

Poscondición: $\{(está = true \wedge 0 \leq pos < |A_0| \wedge A_0[pos] = x_0) \vee$
$(está = false \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

$está \leftarrow false$ $O(1)$

$pos \leftarrow -1$ $O(1)$

$j \leftarrow 0$ $O(1)$

while $(j < |A|)$ { $O(1)$

  if $(A[j] = x)$ { $O(1)$

    $está \leftarrow$ true $O(1)$

    $pos \leftarrow j$ $O(1)$

  }

  $j \leftarrow j + 1$ $O(1)$

} while: $O(|A|)$ iteraciones

¿Cuál es el orden de complejidad?

Problema de búsqueda

Encabezado: *Buscar* : $x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \rightarrow está \in \mathbb{B} \times pos \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0\}$

Poscondición: $\{(está = true \wedge 0 \leq pos < |A_0| \wedge A_0[pos] = x_0) \vee$
$(está = false \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

$está \leftarrow false$ $O(1)$

$pos \leftarrow -1$ $O(1)$

$j \leftarrow 0$ $O(1)$

while $(j < |A|)$ { $O(1)$

  if $(A[j] = x)$ { $O(1)$

    $está \leftarrow$ true $O(1)$

    $pos \leftarrow j$ $O(1)$

  }

  $j \leftarrow j + 1$ $O(1)$

} while: $O(|A|)$ iteraciones

¿Cuál es el orden de complejidad?

$T(|A|) \in O(|A|)$ **Búsqueda lineal**

Problema de búsqueda

Encabezado: *Buscar* : $x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \to está \in \mathbb{B} \times pos \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0\}$

Poscondición: $\{(está = true \wedge 0 \leq pos < |A_0| \wedge A_0[pos] = x_0) \vee$
$(está = false \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

$está \leftarrow false$ $O(1)$

$pos \leftarrow -1$ $O(1)$

$j \leftarrow 0$ $O(1)$

while $(j < |A|)$ { $O(1)$

  if $(A[j] = x)$ { $O(1)$

    $está \leftarrow$ true $O(1)$

    $pos \leftarrow j$ $O(1)$

  }

  $j \leftarrow j + 1$ $O(1)$

} while: $O(|A|)$ iteraciones

¿Cuál es el orden de complejidad?

$T(|A|) \in O(|A|)$ **Búsqueda lineal**

¿Y si agregamos "$\wedge \neg está$"
a la guarda del while?

Problema de búsqueda

Encabezado: *Buscar* : $x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \to está \in \mathbb{B} \times pos \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0\}$

Poscondición: $\{(está = true \wedge 0 \leq pos < |A_0| \wedge A_0[pos] = x_0) \vee$
$(está = false \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

$está \leftarrow false$ $O(1)$
$pos \leftarrow -1$ $O(1)$
$j \leftarrow 0$ $O(1)$
while $(j < |A|)$ { $O(1)$
  if $(A[j] = x)$ { $O(1)$
    $está \leftarrow$ true $O(1)$
    $pos \leftarrow j$ $O(1)$
  }
  $j \leftarrow j + 1$ $O(1)$
} while: $O(|A|)$ iteraciones

¿Cuál es el orden de complejidad?

$T(|A|) \in O(|A|)$ **Búsqueda lineal**

¿Y si agregamos "$\wedge \neg está$"
a la guarda del while?

En este algoritmo, cortar antes
la ejecución puede ahorrar tiempo,
pero no cambia el
orden en el peor caso.

Problema de búsqueda

Encabezado: $\mathit{Buscar} : x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \to \mathit{está} \in \mathbb{B} \times \mathit{pos} \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0\}$

Poscondición: $\{(\mathit{está} = \mathit{true} \wedge 0 \leq \mathit{pos} < |A_0| \wedge A_0[\mathit{pos}] = x_0) \vee$
$(\mathit{está} = \mathit{false} \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

$\mathit{está} \leftarrow \mathit{false}$ $O(1)$
$\mathit{pos} \leftarrow -1$ $O(1)$
$j \leftarrow 0$ $O(1)$
while $(j < |A|)$ { $O(1)$
  if $(A[j] = x)$ { $O(1)$
    $\mathit{está} \leftarrow$ true $O(1)$
    $\mathit{pos} \leftarrow j$ $O(1)$
  }
  $j \leftarrow j + 1$ $O(1)$
} while: $O(|A|)$ iteraciones

¿Cuál es el orden de complejidad?

$T(|A|) \in O(|A|)$ **Búsqueda lineal**

¿Y si agregamos "$\wedge \neg \mathit{está}$" a la guarda del while?

En este algoritmo, cortar antes la ejecución puede ahorrar tiempo, pero no cambia el **orden en el peor caso**.

¿Cuán eficientes son estos algoritmos si A está ordenado?

Veamos un algoritmo de búsqueda para listas ordenadas.

4	7	23	41	44	59	97	134	165	187	210	212	249	280	314
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Buscamos el número **97**...

Veamos un algoritmo de búsqueda para listas ordenadas.

4	7	23	41	44	59	97	134	165	187	210	212	249	280	314
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Buscamos el número **97**...

4	7	23	41	44	59	97	134
0	1	2	3	4	5	6	7

165	187	210	212	249	280	314
8	9	10	11	12	13	14

Veamos un algoritmo de búsqueda para listas ordenadas.

4	7	23	41	44	59	97	134	165	187	210	212	249	280	314
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Buscamos el número **97**...

4	7	23	41	44	59	97	134
0	1	2	3	4	5	6	7

165	187	210	212	249	280	314
8	9	10	11	12	13	14

4	7	23	41
0	1	2	3

44	59	97	134
4	5	6	7

Veamos un algoritmo de búsqueda para listas ordenadas.

4	7	23	41	44	59	97	134	165	187	210	212	249	280	314
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Buscamos el número **97**...

4	7	23	41	44	59	97	134
0	1	2	3	4	5	6	7

165	187	210	212	249	280	314
8	9	10	11	12	13	14

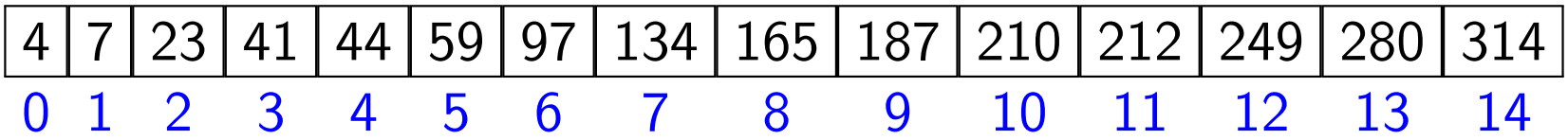
4	7	23	41
0	1	2	3

44	59	97	134
4	5	6	7

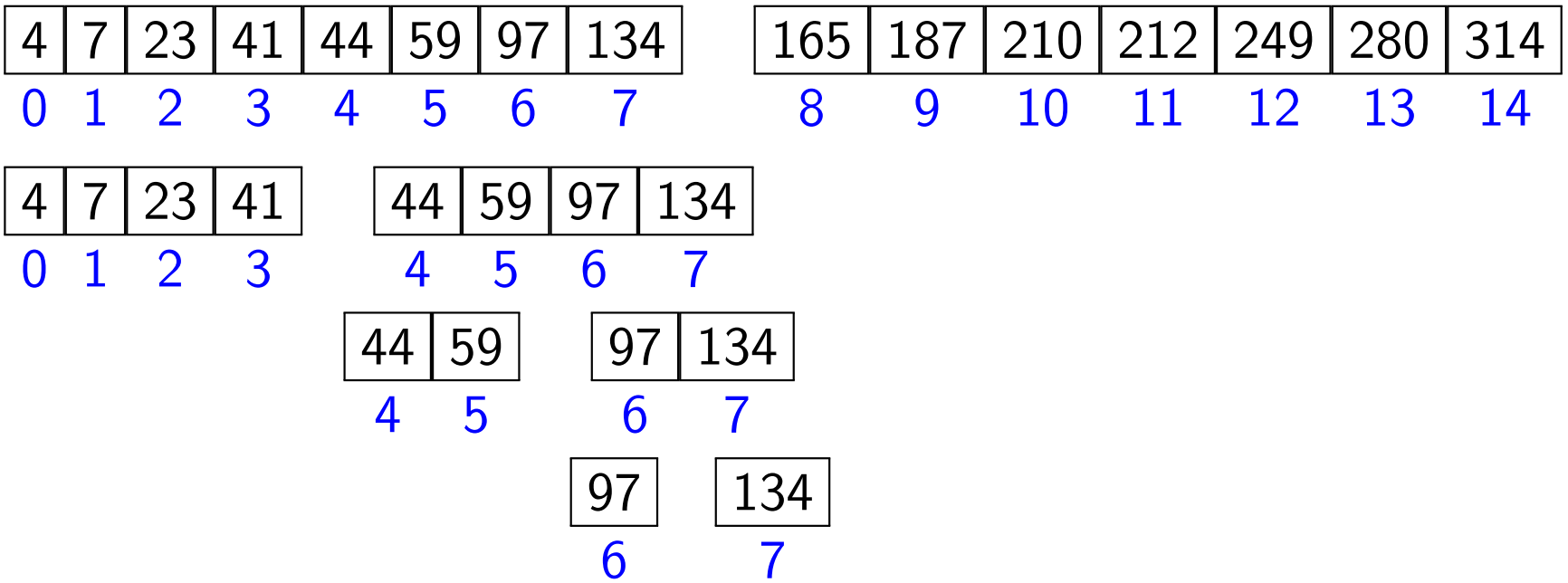
44	59
4	5

97	134
6	7

Veamos un algoritmo de búsqueda para listas ordenadas.



Buscamos el número **97**...



Veamos un algoritmo de búsqueda para listas ordenadas.

4	7	23	41	44	59	97	134	165	187	210	212	249	280	314
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Buscamos el número **97**...

4	7	23	41	44	59	97	134
0	1	2	3	4	5	6	7

165	187	210	212	249	280	314
8	9	10	11	12	13	14

4	7	23	41
0	1	2	3

44	59	97	134
4	5	6	7

44	59
4	5

97	134
6	7

97	134
6	7

97	✓
6	

Búsqueda binaria

¿Cuál es el comportamiento detrás de este algoritmo?

Si la lista está ordenada, entonces en cada paso puedo partir la lista en:

a) la mitad que puede contener el elemento; y
b) la mitad que no puede contenerlo.

Indefectiblemente, se llega a un punto en que la lista ya no puede ser dividida (tiene un solo elemento) y, o bien el elemento es el buscado o no.

Encabezado: $\mathit{Buscar} : x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \rightarrow está \in \mathbb{B} \times pos \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0 \wedge$
$(\forall i)(0 \leq i < |A| - 1 \Rightarrow A[i] \leq A[i+1])\}$

Poscondición: $\{(está = true \wedge 0 \leq pos < |A_0| \wedge A_0[pos] = x_0) \vee$
$(está = false \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

Encabezado: *Buscar* : $x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \rightarrow está \in \mathbb{B} \times pos \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0 \wedge$
$(\forall i)(0 \leq i < |A| - 1 \Rightarrow A[i] \leq A[i+1])\}$

Poscondición: $\{(está = true \wedge 0 \leq pos < |A_0| \wedge A_0[pos] = x_0) \vee$
$(está = false \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

$(está, pos) \leftarrow (false, -1)$
$(izq, der) \leftarrow (0, |A| - 1)$
while $(izq < der)$ {
    $med \leftarrow (izq + der)$ div 2
    if $(A[med] < x)$ {
        $izq \leftarrow med + 1$
    } else {
        $der \leftarrow med$
    }
}
if $(x = A[izq])$ {
    $(está, pos) \leftarrow (true, izq)$

Encabezado: *Buscar* : $x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \rightarrow está \in \mathbb{B} \times pos \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0 \wedge$
$(\forall i)(0 \leq i < |A| - 1 \Rightarrow A[i] \leq A[i+1])\}$

Poscondición: $\{(está = true \wedge 0 \leq pos < |A_0| \wedge A_0[pos] = x_0) \vee$
$(está = false \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

$(está, pos) \leftarrow (false, -1)$
$(izq, der) \leftarrow (0, |A| - 1)$
while $(izq < der)$ {
    $med \leftarrow (izq + der)$ div 2
    if $(A[med] < x)$ {
        $izq \leftarrow med + 1$
    } else {
        $der \leftarrow med$
    }
}
if $(x = A[izq])$ {
    $(está, pos) \leftarrow (true, izq)$

¿Cuál es el orden de complejidad?

Encabezado: *Buscar* : $x \in \mathbb{Z} \times A \in \mathbb{Z}[\,] \rightarrow está \in \mathbb{B} \times pos \in \mathbb{Z}$

Precondición: $\{A = A_0 \wedge x = x_0 \wedge$
$(\forall i)(0 \leq i < |A| - 1 \Rightarrow A[i] \leq A[i+1])\}$

Poscondición: $\{(está = true \wedge 0 \leq pos < |A_0| \wedge A_0[pos] = x_0) \vee$
$(está = false \wedge (\forall i)(0 \leq i < |A_0| \Rightarrow A_0[i] \neq x_0))\}$

$(está, pos) \leftarrow (false, -1)$
$(izq, der) \leftarrow (0, |A| - 1)$
while $(izq < der)$ {
    $med \leftarrow (izq + der)$ div 2
    if $(A[med] < x)$ {
        $izq \leftarrow med + 1$
    } else {
        $der \leftarrow med$
    }
}
if $(x = A[izq])$ {
    $(está, pos) \leftarrow (true, izq)$

¿Cuál es el orden de complejidad?

$T(|A|) \in O(\log |A|)$

orden logarítmico

Búsqueda binaria

Para ver que el orden es logarítmico, basta observar que la función variante $fv = der - izq$ decrece aproximadamente a la mitad en cada iteración:

Sea $fv = der - izq$ al comienzo de una iteración.
Al final de la misma, pueden ocurrir dos cosas:

- $fv' = der - \lfloor \frac{izq+der}{2} \rfloor - 1 \approx \lfloor \frac{fv}{2} \rfloor$
- $fv' = \lfloor \frac{izq+der}{2} \rfloor - izq \approx \lfloor \frac{fv}{2} \rfloor$

En ambos casos, fv termina valiendo aproximadamente la mitad que al principio de la iteración.

Como el ciclo termina cuando $fv \leq 0$, el cuerpo del ciclo se ejecuta $O(\log_2 |A|)$ veces. $\square$

Búsqueda binaria

Para ver que el orden es logarítmico, basta observar que la función variante $fv = der - izq$ decrece aproximadamente a la mitad en cada iteración:

Sea $fv = der - izq$ al comienzo de una iteración.
Al final de la misma, pueden ocurrir dos cosas:

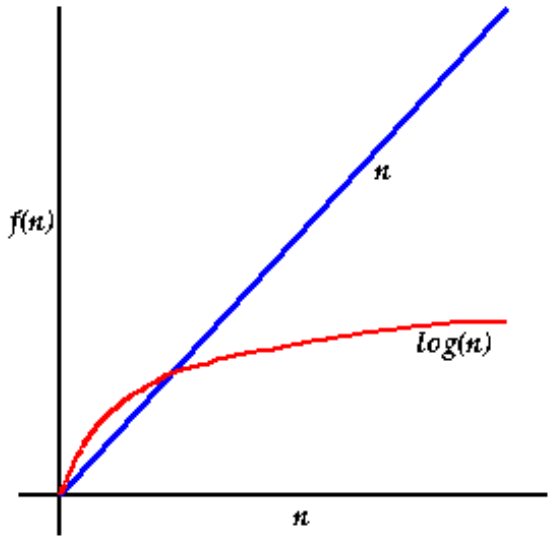
- $fv' = der - \lfloor \frac{izq+der}{2} \rfloor - 1 \approx \lfloor \frac{fv}{2} \rfloor$
- $fv' = \lfloor \frac{izq+der}{2} \rfloor - izq \approx \lfloor \frac{fv}{2} \rfloor$

En ambos casos, fv termina valiendo aproximadamente la mitad que al principio de la iteración.

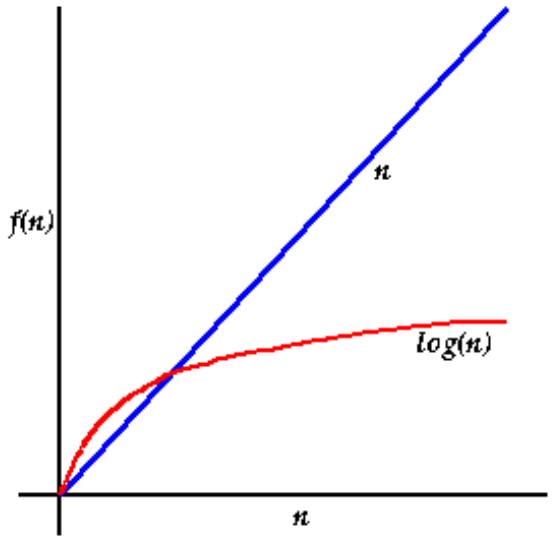
Como el ciclo termina cuando $fv \leq 0$, el cuerpo del ciclo se ejecuta $O(\log_2 |A|)$ veces. $\square$

Obs.: La base del $\log$ es irrelevante para el orden de T.

Búsqueda lineal vs. binaria

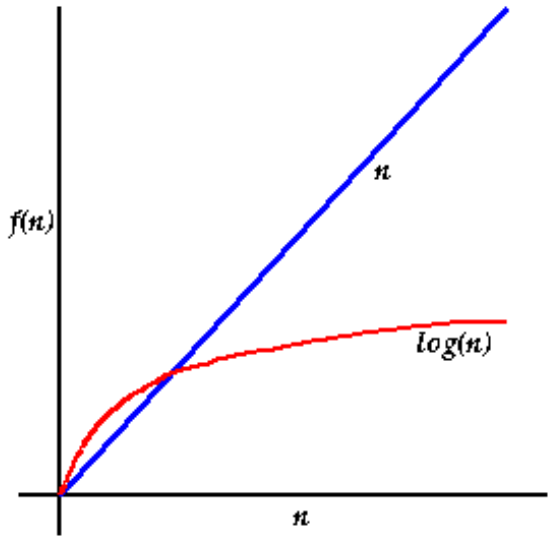


Búsqueda lineal vs. binaria



¿Cuán importante es la diferencia entre $O(\log n)$ y $O(n)$?

Búsqueda lineal vs. binaria

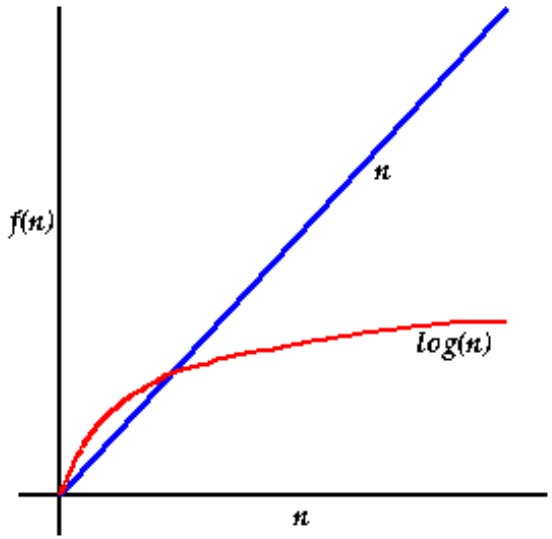


¿Cuán importante es la diferencia entre $O(\log n)$ y $O(n)$?

Depende de nuestro contexto...

- ¿Cuál es el tamaño del listado en el cual haremos la búsqueda? (FCEyN vs. ANSES)

Búsqueda lineal vs. binaria

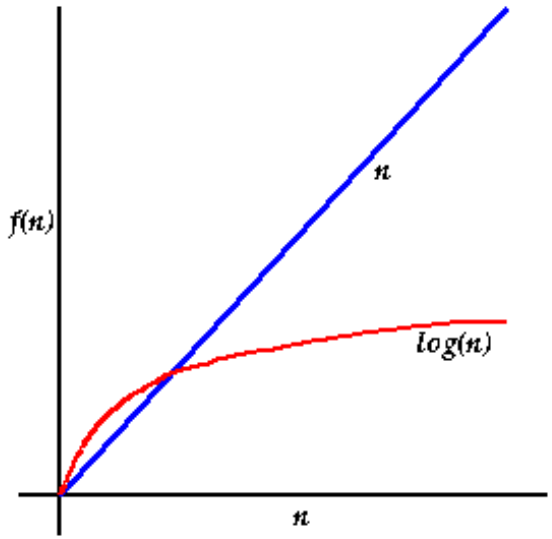


¿Cuán importante es la diferencia entre $O(\log n)$ y $O(n)$?

Depende de nuestro contexto...

- ¿Cuál es el tamaño del listado en el cual haremos la búsqueda? (FCEyN vs. ANSES)
- ¿Cuánto cuesta cada consulta individual? (Lectura en memoria vs. consulta por Internet)

Búsqueda lineal vs. binaria



¿Cuán importante es la diferencia entre $O(\log n)$ y $O(n)$?

Depende de nuestro contexto...

- ¿Cuál es el tamaño del listado en el cual haremos la búsqueda? (FCEyN vs. ANSES)
- ¿Cuánto cuesta cada consulta individual? (Lectura en memoria vs. consulta por Internet)
- ¿Cuántas veces vamos a necesitar hacer esta búsqueda? (una vez por mes vs. millones de veces por día)

¿Cuál programa usamos?

Objetivos contrapuestos

Para resolver un problema, queremos un programa...

1. que sea fácil de programar (que escribirlo nos demande poco tiempo, que sea simple y fácil de entender);
2. que consuma pocos recursos: tiempo y espacio (memoria, disco rígido).

En general priorizamos un objetivo sobre el otro:

- para programas que correrán pocas veces, priorizamos el objetivo 1;
- para programas que correrán muchas veces, priorizamos el objetivo 2.

Repaso de la clase de hoy

- Búsqueda lineal y binaria.

Próximos temas

- Algoritmos de ordenamiento.