Introducción a la Computación (Matemática)

Primer Cuatrimestre de 2018

Algoritmos de Ordenamiento

Complejidad temporal

$T(n)$: Tiempo de ejecución (o complejidad temporal) de un programa; medido en cantidad de operaciones en función del tamaño de la entrada.

- **Peor caso**: $T(n)$ es una cota superior del tiempo de ejecución para entradas arbitrarias de tamaño n.

Complejidad temporal

$T(n)$: Tiempo de ejecución (o complejidad temporal) de un programa; medido en cantidad de operaciones en función del tamaño de la entrada.

- **Peor caso**: $T(n)$ es una cota superior del tiempo de ejecución para entradas arbitrarias de tamaño n.

Orden de complejidad temporal: $T(n) \in O(f(n))$ si existen constantes enteras positivas c y n_0 tales que para $n \geq n_0$, $T(n) \leq c \cdot f(n)$.

- **Ejemplo:** $T(n) = 3n^3 + 2n^2 \in O(n^3)$, dado que con $n_0 = 0$ y $c = 5$, vale que para $n \geq 0$, $T(n) \leq 5 \cdot n^3$.

En general, nos interesa buscar algoritmos con el menor orden posible, para poder procesar entradas arbitrariamente grandes.

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 $f_1 \cdot f_2 \in O(n)$.

Algoritmo de búsqueda lineal

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

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

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

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

Algoritmo de búsqueda lineal

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

$\mathit{está} \leftarrow \mathit{false}$ $\quad O(1)$

$\mathit{pos} \leftarrow -1$ $\quad O(1)$

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

while $(j < |A|)$ { $\quad O(1)$ $\qquad$ while: $O(|A|)$ iteraciones

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

    $\mathit{está} \leftarrow \mathit{true}$ $\quad O(1)$

    $\mathit{pos} \leftarrow j$ $\quad O(1)$

  }

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

}

Complejidad temporal: $O(1+1+1+1+|A| \cdot (1+1+1+1+1))$

$= O(1 + |A| \cdot 1) = O(|A|)$

Algoritmo de búsqueda binaria

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: $\{A = A_0 \wedge x = x_0 \ \wedge \text{Creciente}(A_0)\}$

donde $\text{Creciente}(X) \equiv \forall i \cdot (0 \leq i < |X| - 1 \Rightarrow X[i] \leq X[i+1])$

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

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

Algoritmo de búsqueda binaria

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: $\{A = A_0 \wedge x = x_0 \ \wedge \text{Creciente}(A_0)\}$

donde $\text{Creciente}(X) \equiv \forall i \cdot (0 \leq i < |X| - 1 \Rightarrow X[i] \leq X[i+1])$

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

$(\mathit{está}, \mathit{pos}) \leftarrow (\mathit{false}, -1)$

$(\mathit{izq}, \mathit{der}) \leftarrow (0, |A| - 1)$

while $(\mathit{izq} < \mathit{der})$ { $O(1)$ while: $O(\log_2 |A|)$ iteraciones

$\mathit{med} \leftarrow (\mathit{izq} + \mathit{der})$ div 2 $O(1)$

if $(A[\mathit{med}] < x)$ { $O(1)$

$\mathit{izq} \leftarrow \mathit{med} + 1$ $O(1)$

} else {

$\mathit{der} \leftarrow \mathit{med}$ $O(1)$

}

}

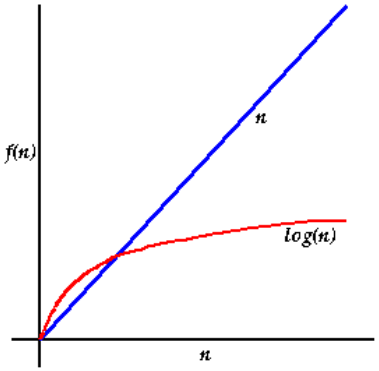
if $(x = A[\mathit{izq}])$ { $O(1)$

$(\mathit{está}, \mathit{pos}) \leftarrow (\mathit{true}, \mathit{izq})$ $O(1)$

Problema de ordenamiento

Problema de búsqueda:

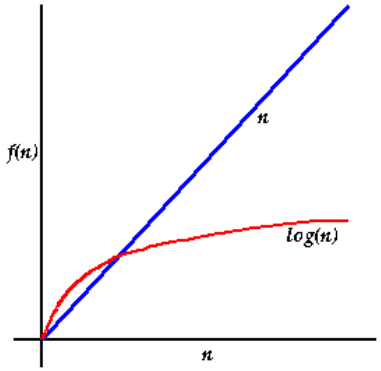
- $O(n)$ para listas arbitrarias.
- $O(\log n)$ para listas ordenadas.



Problema de ordenamiento

Problema de búsqueda:

- $O(n)$ para listas arbitrarias.
- $O(\log n)$ para listas ordenadas.



Ordenar una lista es un problema de mucha importancia en la práctica.

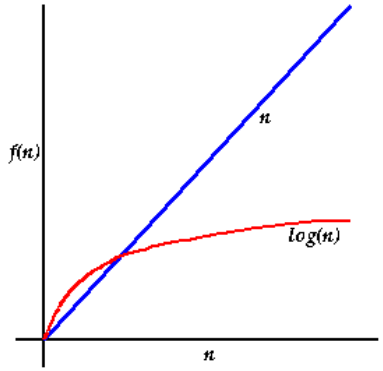
- Ej: ¿De qué sirve una guía de teléfonos desordenada?

Encabezado: *Ordenar* : $A \in \mathbb{Z}[\,] \to \emptyset$
Precondición: $\{A = A_0\}$
Poscondición: $\{\text{Permutación}(A, A_0) \wedge \text{Creciente}(A)\}$

Problema de ordenamiento

Problema de búsqueda:

- $O(n)$ para listas arbitrarias.
- $O(\log n)$ para listas ordenadas.



Ordenar una lista es un problema de mucha importancia en la práctica.

- Ej: ¿De qué sirve una guía de teléfonos desordenada?

Encabezado: *Ordenar* : $A \in \mathbb{Z}[\,] \to \emptyset$
Precondición: $\{A = A_0\}$
Poscondición: $\{\text{Permutación}(A, A_0) \wedge \text{Creciente}(A)\}$

¿Ideas para resolver este problema?

Problema de ordenamiento

59	7	388	41	2	280	50	123

Selection sort

Para cada i entre 0 y $|A| - 1$, buscar el menor elemento en $A[i \,..\, |A| - 1]$ e intercambiarlo con $A[i]$.

Selection sort

Para cada i entre 0 y $|A| - 1$, buscar el menor elemento en $A[i \,..\, |A| - 1]$ e intercambiarlo con $A[i]$.

```
i ← 0
while (i < |A|) {
        posmin ← i
        j ← i + 1
        while (j < |A|) {
                if (A[j] < A[posmin]) {
                   posmin ← j
                }
                j ← j + 1
        }
        swap(A, i, posmin)
        i ← i + 1
}
```

Selection sort

Para cada i entre 0 y $|A| - 1$, buscar el menor elemento en $A[i \,..\, |A| - 1]$ e intercambiarlo con $A[i]$.

$i \leftarrow 0$ $O(1)$
while $(i < |A|)$ { O(1)
  $posmin \leftarrow i$ $O(1)$
  $j \leftarrow i + 1$ $O(1)$
  while $(j < |A|)$ { O(1)
    if $(A[j] < A[posmin])$ { $O(1)$
      $posmin \leftarrow j$ $O(1)$
    }
    $j \leftarrow j + 1$ $O(1)$
  }
  swap$(A, i, posmin)$ $O(1)$
  $i \leftarrow i + 1$ $O(1)$
}

Selection sort

Para cada i entre 0 y $|A| - 1$, buscar el menor elemento en $A[i \,..\, |A| - 1]$ e intercambiarlo con $A[i]$.

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

while $(i < |A|)$ { O(1) while: $O(|A|)$ iteraciones

  $posmin \leftarrow i$ $O(1)$

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

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

    if $(A[j] < A[posmin])$ { $O(1)$

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

    }

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

  }

  swap$(A, i, posmin)$ $O(1)$

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

}

Selection sort

Para cada i entre 0 y $|A| - 1$, buscar el menor elemento en $A[i \,..\, |A| - 1]$ e intercambiarlo con $A[i]$.

$i \leftarrow 0$ $O(1)$
while $(i < |A|)$ { O(1) while: $O(|A|)$ iteraciones
  $posmin \leftarrow i$ $O(1)$
  $j \leftarrow i + 1$ $O(1)$
  while $(j < |A|)$ { O(1) while: $O(|A|)$ iteraciones
    if $(A[j] < A[posmin])$ { $O(1)$
      $posmin \leftarrow j$ $O(1)$
    }
    $j \leftarrow j + 1$ $O(1)$
  }
  swap$(A, i, posmin)$ $O(1)$
  $i \leftarrow i + 1$ $O(1)$
} Complejidad temporal:

$$O\big(1 + 1 + |A| \cdot (1 + 1 + 1 + 1 + |A| \cdot (1 + 1 + \text{máx}(1, 0) + 1) + 1 + 1)\big) =$$

Selection sort

Para cada i entre 0 y $|A| - 1$, buscar el menor elemento en $A[i \,..\, |A| - 1]$ e intercambiarlo con $A[i]$.

$i \leftarrow 0$
while $(i < |A|)$ {
    $posmin \leftarrow i$
    $j \leftarrow i + 1$
    while $(j < |A|)$ {
        if $(A[j] < A[posmin])$ {
            $posmin \leftarrow j$
        }
        $j \leftarrow j + 1$
    }
    swap$(A, i, posmin)$
    $i \leftarrow i + 1$
}

Selection sort

Para cada i entre 0 y $|A| - 1$, buscar el menor elemento en $A[i \,..\, |A| - 1]$ e intercambiarlo con $A[i]$.

$i \leftarrow 0$
while $(i < |A|)$ {
    $posmin \leftarrow i$
    $j \leftarrow i + 1$
    while $(j < |A|)$ {
        if $(A[j] < A[posmin])$ {
            $posmin \leftarrow j$
        }
        $j \leftarrow j + 1$
    }
    swap$(A, i, posmin)$
    $i \leftarrow i + 1$
}

Inv $\equiv$ Permutación$(A, A_0) \wedge 0 \leq i \leq |A| \wedge$
Creciente$(A, 0, i - 1) \wedge \big(0 < i < |A| \Rightarrow$
Máx$(A, 0, i-1) \leq$ Mín$(A, i, |A|-1)\big)$

Insertion sort

Para cada i entre 0 y $|A| - 1$, mover el elemento $A[i]$ a su posición correcta en $A[0\,..\,i]$ (así, $A[0\,..\,i]$ queda ordenado).

Insertion sort

Para cada i entre 0 y $|A| - 1$, mover el elemento $A[i]$ a su posición correcta en $A[0\,..\,i]$ (así, $A[0\,..\,i]$ queda ordenado).

```
i ← 0
while (i < |A|) {
        j ← i
        while (j > 0 ∧ A[j − 1] > A[j]) {
                swap(A, j − 1, j)
                j ← j − 1
        }
        i ← i + 1
}
```

Insertion sort

Para cada i entre 0 y $|A| - 1$, mover el elemento $A[i]$ a su posición correcta en $A[0\,..\,i]$ (así, $A[0\,..\,i]$ queda ordenado).

$i \leftarrow 0$ $O(1)$
while $(i < |A|)$ { O(1)
  $j \leftarrow i$ $O(1)$
  while $(j > 0 \wedge A[j-1] > A[j])$ { O(1)
    swap$(A, j-1, j)$ $O(1)$
    $j \leftarrow j - 1$ $O(1)$
  }
  $i \leftarrow i + 1$ $O(1)$
}

Insertion sort

Para cada i entre 0 y $|A| - 1$, mover el elemento $A[i]$ a su posición correcta en $A[0\,..\,i]$ (así, $A[0\,..\,i]$ queda ordenado).

$i \leftarrow 0$ $O(1)$
while $(i < |A|)$ { O(1) while: $O(|A|)$ iteraciones
  $j \leftarrow i$ $O(1)$
  while $(j > 0 \wedge A[j-1] > A[j])$ { O(1) while: $O(|A|)$ iters
    swap$(A, j-1, j)$ $O(1)$
    $j \leftarrow j - 1$ $O(1)$
  }
  $i \leftarrow i + 1$ $O(1)$
}

Insertion sort

Para cada i entre 0 y $|A| - 1$, mover el elemento $A[i]$ a su posición correcta en $A[0\,..\,i]$ (así, $A[0\,..\,i]$ queda ordenado).

$i \leftarrow 0$ $O(1)$
while $(i < |A|)$ { O(1) while: $O(|A|)$ iteraciones
  $j \leftarrow i$ $O(1)$
  while $(j > 0 \wedge A[j-1] > A[j])$ { O(1) while: $O(|A|)$ iters
    swap$(A, j-1, j)$ $O(1)$
    $j \leftarrow j - 1$ $O(1)$
  }
  $i \leftarrow i + 1$ $O(1)$
}

Complejidad temporal: $O(|A|^2)$

Bubble sort

Comparar cada par de elementos adyacentes en A, e invertirlos si están en orden incorrecto. Repetir $|A|$ veces.

Bubble sort

Comparar cada par de elementos adyacentes en A, e invertirlos si están en orden incorrecto. Repetir $|A|$ veces.

```
i ← 0
while (i < |A|) {
        j ← 0
        while (j < |A| − 1) {
                if (A[j] > A[j + 1]) {
                   swap(A, j, j + 1)
                }
                j ← j + 1
        }
        i ← i + 1
}
```

Bubble sort

Comparar cada par de elementos adyacentes en A, e invertirlos si están en orden incorrecto. Repetir $|A|$ veces.

$i \leftarrow 0$ $O(1)$
while $(i < |A|)$ { O(1)
  $j \leftarrow 0$ $O(1)$
  while $(j < |A| - 1)$ { O(1)
    if $(A[j] > A[j+1])$ { $O(1)$
      swap$(A, j, j+1)$ $O(1)$
    }
    $j \leftarrow j + 1$ $O(1)$
  }
  $i \leftarrow i + 1$ $O(1)$
}

Bubble sort

Comparar cada par de elementos adyacentes en A, e invertirlos si están en orden incorrecto. Repetir $|A|$ veces.

$i \leftarrow 0$ $O(1)$
while $(i < |A|)$ { O(1) while: $O(|A|)$ iteraciones
  $j \leftarrow 0$ $O(1)$
  while $(j < |A| - 1)$ { O(1) while: $O(|A|)$ iteraciones
    if $(A[j] > A[j+1])$ { $O(1)$
      swap$(A, j, j+1)$ $O(1)$
    }
    $j \leftarrow j + 1$ $O(1)$
  }
  $i \leftarrow i + 1$ $O(1)$
}

Bubble sort

Comparar cada par de elementos adyacentes en A, e invertirlos si están en orden incorrecto. Repetir $|A|$ veces.

```
i ← 0                                   O(1)
while (i < |A|) {                       O(1)        while: O(|A|) iteraciones
    j ← 0                               O(1)
    while (j < |A| − 1) {               O(1)        while: O(|A|) iteraciones
        if (A[j] > A[j + 1]) {          O(1)
            swap(A, j, j + 1)           O(1)
        }
        j ← j + 1                       O(1)
    }
    i ← i + 1                           O(1)
}
```

Complejidad temporal: $O(|A|^2)$

Complejidad y Ordenamiento

Bibliografía:

- Aho, Hopcroft & Ullman, "Estructuras de Datos y Algoritmos", Addison-Wesley, 1988.
- Balcazar, "Programación metódica", McGraw-Hill, 1993.

Demos y otras yerbas:

- http://www.sorting-algorithms.com/
- http://www.youtube.com/watch?v=MtcrEhrt_K0
- http://www.youtube.com/watch?v=INHF_5RIxTE

Repaso de la clase de hoy

- Algoritmos de ordenamiento de listas:
  - Selection sort. $O(n^2)$
  - Insertion sort. $O(n^2)$
  - Bubble sort. $O(n^2)$

Próximos temas

- Recursión algorítmica.
- Divide and Conquer. Torres de Hanoi.
- Merge sort. $O(n \log n)$
- Cálculo de complejidad algorítmica para funciones recursivas.