Algoritmos de ordenamiento sobre secuencias

Algoritmos y Estructuras de Datos I

Motivación

4	5	1	2	3	6	7

$$\downarrow$$

1	2	3	4	5	6	7

Ordenamiento de secuencias

- proc $ordenar(\text{inout } s : seq\langle\mathbb{Z}\rangle)\{$
  Pre $\{s = S_0\}$
  Post $\{mismos(s, S_0) \ \wedge \ ordenado(s)\}$
  $\}$
- pred $mismos(s, t : seq\langle\mathbb{Z}\rangle)\{$
  $(\forall e : \mathbb{Z})(\#apariciones(s, e) = \#apariciones(t, e))$
  $\}$
- aux $\#apariciones(s : seq\langle T\rangle, e : T) : \mathbb{Z} =$
  $\sum_{i=0}^{|s|-1}(\text{if } s[i] = e \text{ then } 1 \text{ else } 0 \text{ fi})$
- pred $ordenado(s : seq\langle\mathbb{Z}\rangle)\{$
  $(\forall i : \mathbb{Z})(0 \leq i < |s| - 1 \rightarrow_L s[i] \leq s[i+1])$
  $\}$

Ordenamiento de secuencias

- Modificamos la secuencia solamente a través de intercambios de elementos.

proc $swap(\text{inout } s : seq\langle\mathbb{Z}\rangle, \text{in } i, j : \mathbb{Z})\{$
Pre $\{0 \leq i, j < |s| \wedge s = S_0\}$
Post $\{s[i] = S_0[j] \wedge s[j] = S_0[i] \wedge$
$(\forall k : \mathbb{Z})(0 \leq k < |s| \wedge i \neq k \wedge j \neq k \rightarrow_L s[k] = S_0[k])\}$
$\}$

- **Propiedad 1:**

$$s = S_0 \rightarrow mismos(s, S_0)$$

- **Propiedad 2:**

$$\{mismos(s, S_0)\}$$
swap(s,i,j)
$$\{mismos(s, S_0)\}$$

- De esta forma, nos aseguramos que $mismos(s, S_0)$ a lo largo de la ejecución del algoritmo.

Ordenamiento por selección (Selection Sort)

- **Idea:** Seleccionar el mínimo elemento e intercambiarlo con la primera posición de la secuencia. Repetir con el segundo, etc.

4	5	1	2	3	6	7

Ordenamiento por selección (Selection Sort)

- **Idea:** Seleccionar el mínimo elemento e intercambiarlo con la primera posición de la secuencia. Repetir con el segundo, etc.

```
void selectionSort(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    // indice del minimo elemento de s entre i y s.size()
    int minPos = ...
    swap(s, i, minPos);
  }
}
```


Ordenamiento por selección (Selection Sort)

- Podemos refinar un poco el código:

```
void selectionSort(vector<int> &s) {
  for(int i=0; i<s.size()-1; i++) {
    int minPos= findMinPosition(s, i, s.size());
    swap(s, i, minPos);
  }
}
```

- Entonces surge la necesidad de especificar el problema auxiliar de buscar el mínimo entre i y $s.size()$:

proc *findMinPosition*(in $s : seq\langle\mathbb{Z}\rangle$, in $d, h : \mathbb{Z}$, out $min : \mathbb{Z}$){
  Pre $\{0 \leq d < h \leq |s|\}$
  Post $\{d \leq min < h$
    $\wedge_L (\forall i : \mathbb{Z})(d \leq i < h \rightarrow_L s[min] \leq s[i])\}$
}

Buscar el Mínimo Elemento

$d\,min$		min					h
$\downarrow$		$\downarrow$					$\downarrow$
4	5	1	2	3	6	7	
	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	
	i	i	i	i	i	i	

- ¿Qué invariante de ciclo podemos proponer?

$$d \leq min < i \leq h \wedge_L (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[min] \leq s[j])$$

- ¿Qué función variante podemos usar?

$$fv = h - i$$

Buscar el Mínimo Elemento

- Invariante:

$$d \leq min < i \leq h \ \wedge_L (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[min] \leq s[j])$$

- Función variante

$$fv = h - i$$

- ¿Cómo lo implementamos?

```
int findMinPosition(vector<int> &s, int d, int h) {
  int min = d;
  for(int i = d + 1; i < h; i++) {
    if (s[i] < s[min]) {
      min = i;
    }
  }
  return min;
}
```


Recap: Teorema de corrección de un ciclo

- **Teorema.** Sean un predicado I y una función $fv : \mathbb{V} \rightarrow \mathbb{Z}$ (donde $\mathbb{V}$ es el producto cartesiano de los dominios de las variables del programa), y supongamos que $I \Rightarrow \text{def}(B)$. Si

  1. $P_C \Rightarrow I$,
  2. $\{I \wedge B\}\ S\ \{I\}$,
  3. $I \wedge \neg B \Rightarrow Q_C$,
  4. $\{I \wedge B \wedge V_0 = fv\}$ **S** $\{fv < V_0\}$,
  5. $I \wedge fv \leq 0 \Rightarrow \neg B$,

  ... entonces la siguiente tripla de Hoare es válida:

$$\{P_C\} \textbf{ while B do S endwhile } \{Q_C\}$$

Buscar el Mínimo Elemento

- $P_C \equiv 0 \leq d < h \leq |s| \wedge min = d \wedge i = d + 1$
- $Q_C \equiv d \leq min < h$
  $\wedge_L(\forall i : \mathbb{Z})(d \leq i < h \rightarrow_L s[min] \leq s[i])$
- $B \equiv i < h$
- $I \equiv d \leq min < i \leq h$
  $\wedge_L (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[min] \leq s[j])$
- $fv = h - i$

```
int findMinPosition(vector<int> &s, int d, int h) {
  int min = d;
  for(int i=d+1; i<h; i++) {
    if (s[i] < s[min]) {
      min = i;
    }
  }
  return min;
}
```


Corrección: Buscar el Mínimo Elemento

- $P_C \equiv 0 \leq d < h \leq |s| \wedge min = d \wedge i = d + 1$
- $Q_C \equiv d \leq min < h$
  $\wedge_L(\forall i : \mathbb{Z})(d \leq i < h \rightarrow_L s[min] \leq s[i])$
- $B \equiv i < h$
- $I \equiv d \leq min < i \leq h$
  $\wedge_L (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[min] \leq s[j])$
- $fv = h - i$

- ¿I es se cumple al principio del ciclo (punto 1.)? ✓
- ¿Se cumple la postcondición del ciclo a la salida del ciclo (punto 3.)? ✓
- ¿Si la función variante alcanza la cota inferior la guarda se deja de cumplir (punto 5.)? ✓

Corrección: Buscar el Mínimo Elemento

- $I \equiv d \leq min < i \leq h$
  $\wedge_L\ (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[min] \leq s[j])$
- $fv = h - i$

```
int findMinPosition(vector<int> &s, int d, int h) {
  int min = d;
  for(int i=d+1; i<h; i++) {
    if (s[i] < s[min]) {
      min = i;
    }
  }
  return min;
}
```

- ¿I se preserva en cada iteración (punto 2.)? ✓
- ¿La función variante es estrictamente decreciente (punto 4.)?✓

Ordenamiento por selección (Selection Sort)

- Volvamos ahora al programa de ordenamiento por selección:

```
void selectionSort(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    int minPos = findMinPosition(s, i, s.size());
    swap(s, i, minPos);
  }
}
```

- $P_C \equiv i = 0 \wedge s = S_0$
- $Q_C \equiv mismos(s, S_0) \wedge\ ordenado(s)$
- $B \equiv i < |s|$
- $I \equiv ?$
  - ¡Luego de la i-ésima iteración, $subseq(s, 0, i)$ contiene los i primeros elementos ordenados! ¿Tenemos entonces el invariante del ciclo?
  - $I \equiv mismos(s, S_0) \wedge ((0 \leq i \leq |s|) \wedge_L ordenado(subseq(s, 0, i)))$
- $fv = |s| - i$

Ordenamiento por selección (Selection Sort)

- $I \equiv mismos(s, S_0) \wedge ((0 \leq i \leq |s|) \wedge_L ordenado(subseq(s, 0, i)))$
- $fv = |s| - i$

```
void selectionSort(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    int minPos = findMinPosition(s, i, s.size());
    swap(s, i, minPos);
  }
}
```

- ¿I se preserva en cada iteración (punto 2.)? ✗
- Contraejemplo:
  - Si arrancamos la iteración con $i = 1$ y $s = \langle 100, 2, 1 \rangle$
  - Terminamos con $i = 2$ y $s = \langle 100, 1, 2 \rangle$ que no satisface I

Debemos reforzar el invariante para probar la corrección:

$$I \equiv mismos(s, S_0) \wedge ((0 \leq i \leq |s|) \wedge_L (ordenado(subseq(s, 0, i))) \wedge (\forall j, k : \mathbb{Z})((0 \leq j < i \wedge i \leq k < |s|) \rightarrow_L s[j] \leq s[k]))$$

Corrección: Ordenamiento por selección (Selection Sort)

$$I \equiv mismos(s, S_0) \wedge ((0 \leq i \leq |s|) \wedge_L (ordenado(subseq(s, 0, i))) \wedge (\forall j, k : \mathbb{Z})((0 \leq j < i \wedge i \leq k < |s|) \rightarrow_L s[j] \leq s[k]))$$

Gráficamente:

$x \in subseq(s, 0, i)$	$y \in subseq(s, i+1, \|s\|)$
$\leq y$	$\geq x$
ordenado	?

Corrección: Ordenamiento por selección (Selection Sort)

- $P_C \equiv i = 0 \land s = S_0$
- $Q_C \equiv mismos(s, S_0) \land \ ordenado(s)$
- $B \equiv i < |s|$
- $I \equiv mismos(s, S_0) \land ((0 \leq i \leq |s|) \land_L (ordenado(subseq(s, 0, i))) \land (\forall j, k : \mathbb{Z})((0 \leq j < i \land i \leq k < |s|) \rightarrow_L s[j] \leq s[k]))$
- $fv = |s| - i$

- ¿I es se cumple al principio del ciclo (punto 1.)? ✓
- ¿Se cumple la postcondición del ciclo a la salida del ciclo (punto 3.)?✓
- ¿Si la función variante alcanza la cota inferior la guarda se deja de cumplir (punto 5.)?✓

Corrección: Ordenamiento por selección (Selection Sort)

- $I \equiv mismos(s, S_0) \land ((0 \leq i \leq |s|) \land_L (ordenado(subseq(s, 0, i))) \land (\forall j, k : \mathbb{Z})((0 \leq j < i \land i \leq k < |s|) \rightarrow_L s[j] \leq s[k]))$
- $fv = |s| - i$

```
void selectionSort(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    int minPos = findMinPosition(s, i, s.size());
    swap(s, i, minPos);
  }
}
```

- ¿I se preserva en cada iteración (punto 2.)? ✓
- ¿La función variante es estrictamente decreciente (punto 4.)?✓

Ordenamiento por selección (Selection Sort)

```
int findMinPosition(vector<int> &s, int d, int h) {
  int min = d;
  for(int i=d+1; i<h; i++) {
    if (s[i] < s[min]) {
      min = i;
    }
  }
  return min;
}
void selectionSort(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    int minPos = findMinPosition(s,i,s.size());
    swap(s, i, minPos);
  }
}
```

- ¿Cómo se comporta este algoritmo?
- Veámoslo en `https://visualgo.net/es/sorting`.

Tiempo de ejecución de peor caso

findMinPosition

- Sea $n = |s|$ ¿cuál es el tiempo de ejecución de peor caso de `findMinPosition`?

`int min = d;`	c_1	1
`for(int i=d+1; i<h; i++) {`	c_2	n
`if (s[i] < s[min]) {`	c_3	$n-1$
`min = i;`	c_4	$n-1$
`}`		
`}`		
`return min;`	c_5	1

- $T_{findMinPosition}(n) = 1*c_1 + n*c_2 + (n-1)*c_3 + (n-1)*c_4 + 1*c_5$
- $T_{findMinPosition}(n) \in O(n)$
- Decimos que `findMinPosition` tiene un tiempo de ejecución de peor caso lineal en función de la longitud de la secuencia.

Tiempo de ejecución de peor caso
selectionSort

- Sea $n = |s|$ ¿cuál es el tiempo de ejecución de peor caso para el programa `selectionSort`?

`for(int i=0; i<s.size(); i++) {`	c_1'	$n+1$
`int minPos=findMinPosition(s,i,s.size());`	$c_2' * n$	n
`swap(s, i, minPos);`	c_3'	n
`}`		

- $T_{selectionSort}(n) = (n+1) * c_1' + n * n * c_2' + n * c_3'$
- $T_{selectionSort}(n) \in O((n+1) * n) = O(n^2 + n) = O(n^2)$
- Decimos que `selectionSort` tiene un tiempo de ejecución de peor caso cuadrático en función de la longitud de la secuencia.

Ordenamiento por selección (Selection Sort)

- Variantes del algoritmo básico:
  1. Cocktail sort: consiste en buscar en cada iteración el máximo y el mínimo del vector por ordenar, intercambiando el mínimo con i y el máximo con $|s| - i - 1$.
  2. Bingo sort: consiste en ubicar todas las apariciones del valor mínimo en el vector por ordenar, y mover todos los valores mínimos al mismo tiempo (efectivo si hay muchos valores repetidos).
- El tiempo de ejecución de peor caso de ambas variantes en función de $n = |s|$ es:
  - $T_{cocktailSort}(n) \in O(n^2)$
  - $T_{bingoSort}(n) \in O(n^2)$
- Por lo tanto, ambas variantes de `selectionSort` tienen el *"mismo"* tiempo de ejecución de peor caso (cuadrático)

Ordenamiento por inserción (Insertion Sort)

- Veamos otro algoritmo de ordenamiento, pero donde el invariante (a diferencia de `selectionSort`) es:

$$I \equiv mismos(s, S_0) \wedge (0 \leq i \leq |s| \wedge_L ordenado(subseq(s, 0, i)))$$

- Esto implica que en cada iteración los primeros i elementos están ordenados, sin ser necesariamente los i elementos más pequeños del vector.
- La función variante de este algoritmo de ordenamiento (al igual que `selectionSort`) es:

$$fv = |s| - i$$

Ordenamiento por inserción (Insertion Sort)

$$I \equiv mismos(s, S_0) \wedge (0 \leq i \leq |s| \wedge_L ordenado(subseq(s, 0, i)))$$

```
void insertionSort(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    // Tenemos que preservar el invariante...
  }
}
```

- ¿I es se cumple al principio del ciclo (punto 1.)? ✓
- ¿Se cumple la postcondición del ciclo a la salida del ciclo (punto 3.)? ✓
- ¿I se preserva en cada iteración (punto 2.)?
  - Sabiendo que los primeros i elementos están ordenados, tenemos que hacer que los primeros $i+1$ elementos pasen a estar ordenados!
  - ¿Cómo lo podemos hacer?

Ordenamiento por inserción (Insertion Sort)

Necesitamos desplazar $s[i]$ hasta una posición donde subseq(s,0,i) esté ordenada de vuelta. Ejemplo, ya están ordenadas las primeras 3 posiciones.

1	4	5	2	3	6	7
			$\uparrow$ i			

1	4	5	2	3	6	7
			$\uparrow$ i,j			

1	4	2	5	3	6	7
		$\uparrow$ j	$\uparrow$ i			

1	2	4	5	3	6	7
	$\uparrow$ j		$\uparrow$ i			

Ordenamiento por inserción (Insertion Sort)

- Antes de comenzar el desplazamiento, tenemos que:

ordenada	
	?
$subseq(s,0,i)$	$subseq(s,i,|s|)$

- Durante el desplazamiento, se cumple que que:

ordenada		ordenada	
	x	$> x$	?
$subseq(s,0,j)$	$s[j]$	$subseq(s,j+1,i+1)$	$subseq(s,i+1,|s|)$

- Al finalizar el desplazamiento, nuevamente tenemos que:

ordenada	
	?
$subseq(s,0,i+1)$	$subseq(s,i+1,|s|)$

Ordenamiento por inserción (Insertion Sort)

- Llamemos `insert` a la función auxiliar que desplaza el elemento s[i] ¿cuál es el invariante para esta función?

$$
\begin{aligned}
I \ \equiv\ & 0 \leq j \leq i \\
\wedge\ & mismos(subseq(s,0,i+1), subseq(S_0,0,i+1)) \\
\wedge\ & subseq(s,i+1,|s|) = subseq(S_0,i+1,|s|) \\
\wedge\ & ordenado(subseq(s,0,j)) \ \wedge\ ordenado(subseq(s,j,i+1)) \\
\wedge\ & (\forall k : \mathbb{Z})(j < k \leq i \rightarrow_L s[j] < s[k])
\end{aligned}
$$

- ¿Cuál es la función variante de `insert`?

$$fv = j$$

Ordenamiento por inserción (Insertion Sort)

- ¿Cuál es una posible implementación de `insert`?

```
void insert(vector<int> &s, int i) {
  for(int j=i; j>0 && s[j] < s[j-1]; j--) {
    swap(s, j, j-1);
  }
}
```

- ¿Cuál es una posible implementación de `insertSort`?

```
void insertionSort(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    insert(s,i);
  }
}
```

- ¿Cómo se comporta este algoritmo de ordenamiento?
- Veámoslo en `https://visualgo.net/es/sorting`.

Tiempo de ejecución de peor caso
insert

- Sea $n = |s|$ ¿cuál es el tiempo de ejecución de peor caso de `insert`?

`for(int j=i; j>0 && s[j] < s[j-1]; j--) {`	c_1''	$n+1$
`  swap(s, j, j-1);`	c_2''	n
`}`		

- $T_{insert}(n) = c_1'' * (n+1) + c_2'' * n$
- $T_{insert}(n) \in O(n)$
- `insert` tiene tiempo de ejecución de peor caso lineal.

Tiempo de ejecución de peor caso
insertSort

- Sea $n = |s|$ ¿cuál es el tiempo de ejecución de peor caso de `insertSort`?

`for(int i=0; i<s.size(); i++) {`	c_1'''	$n+1$
`  insert(s, i);`	$c_2''' * n$	n
`}`		

- $T_{insertSort}(n) = c_1''' * (n+1) + c_2''' * n * n$
- $T_{insertSort}(n) \in O(n^2)$
- `insertSort` tiene tiempo de ejecución de peor caso cuadrático (igual que `selectionSort`)

El problema de la bandera holandesa

Dado una secuencia que contiene colores (rojo, blanco y azul) ordenarlos de modo que respeten el orden de la bandera holandesa (primero rojo, luego blanco y luego azul)

Por ejemplo, si la secuencia es:

$$\langle White, Red, Blue, Blue, Red \rangle$$

El programa debe modificar la secuencia para que quede:

$$\langle Red, Red, White, Blue, Blue \rangle$$

El problema de la bandera holandesa

- Si Red=1,White=2 y Blue=3, ¿Cuál sería la especificación del problema?
- proc $banderaHolandesa(\text{inout } s : seq\langle\mathbb{Z}\rangle)\{$
  Pre $\{s = S_0 \land (\forall e : \mathbb{Z})(e \in s \leftrightarrow (e = 1 \lor e = 2 \lor e = 3))\}$
  Post $\{mismos(s, S_0) \ \land \ ordenado(s)\}$
  $\}$
- ¿Cómo podemos implementar una solución a este problema?
  - ¿Podemos usar algún algoritmo de ordenamiento que conozcamos? **Rta:** podemos usar `insertionSort` o `selectionSort`.
  - ¿Cuál es tiempo de ejecución de peor caso? **Rta:** $T_{banderaHolandesa}(n) \in O(|s|^2)$
  - ¿Podemos buscar otra solución que tenga un tiempo de ejecución de peor caso lineal?

Idea de solución

2	2	1	2	3	1	2

$$\#1 = 2, \#2 = 4, \#3 = 1$$

$$\downarrow$$

1	1	2	2	2	2	3

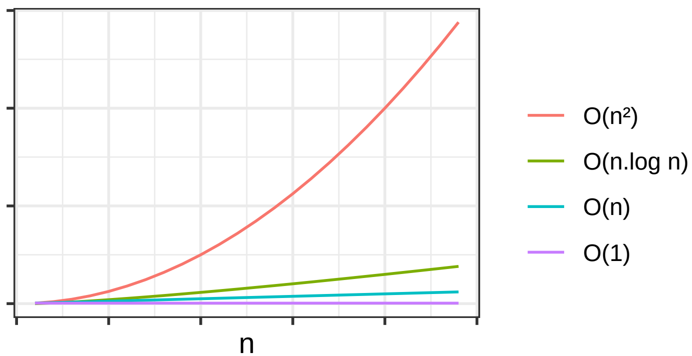
Eficiencia de los Algoritmos de ordenamiento

- Tanto selection sort como insertion sort son algoritmos **cuadráticos** (iteran una cantidad cuadrática de veces)
- ¿Hay algoritmos con comportamiento más eficiente en peor caso?
  - Quicksort y BubbleSort: Peor caso: $O(n^2)$
  - Mergesort y Heapsort: Peor caso: $O(n * log(n))$
  - Counting sort (para secuencias de enteros acotados). Peor caso: $O(n)$
  - Radix sort (para secuencias de enteros). Peor caso: $O(2^{32}) = O(1)$

$$O(1) < O(n) < O(n \times log(n)) < O(n^2)$$

Órden de los órdenes

$$O(1) < O(n) < O(n \times log(n)) < O(n^2)$$



Bibliografía

- Vickers et al. - Reasoned Programming
  - 6.5 - Insertion Sort
- NIST- Dictionary of Algorithms and Data Structures
  - Selection Sort - https://xlinux.nist.gov/dads/HTML/selectionSort.html
  - Bingo Sort - https://xlinux.nist.gov/dads/HTML/bingosort.html
  - Cocktail Sort - https://xlinux.nist.gov/dads/HTML/bidirectionalBubbleSort.html
- Cormen et al. - Introduction to Algorithms
  - Chapter 2.1 - Insertion Sort