

Algoritmos de ordenamiento sobre secuencias

Algoritmos y Estructuras de Datos I

Ordenamiento de vectores

```
► proc ordenar(inout s : seq(Z)) {  
    Pre {s = S0}  
    Post {mismos(s, S0) ∧ ordenado(s)}  
}  
  
► pred mismos(s, t : seq(Z)) {  
    (∀ e : Z)(#apariciones(s, e) = #apariciones(t, e))  
}  
  
► fun #apariciones(s : seq(T), e : T) : Z =  
    Σi=0|s|-1 (if s[i] = e then 1 else 0 fi)  
  
► pred ordenado(s : seq(Z)) {  
    (∀ i : Z)(0 ≤ i < |s| - 1 →L s[i] ≤ s[i + 1])  
}
```

Ordenamiento de vectores

- Modificamos el vector solamente a través de **intercambios** de elementos.

```
proc swap(inout s : seq(Z), in i, j : Z) {  
    Pre {0 ≤ i, j < |s| ∧ s = S0}  
    Post {s[i] = S0[j] ∧ s[j] = S0[i] ∧  
        (∀ k : Z)(0 ≤ k < |s| ∧ i ≠ k ∧ j ≠ k →L s[k] = S0[k])}  
}
```

- **Propiedad 1:**

$$s = S_0 \rightarrow \text{mismos}(s, S_0)$$

- **Propiedad 2:**

$$\begin{aligned} &\{\text{mismos}(s, S_0)\} \\ &\quad \text{swap}(s, i, j) \\ &\{\text{mismos}(s, S_0)\} \end{aligned}$$

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

Ordenamiento por selección (Selection Sort)

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

```
1 void selectionSort(vector<int> &s) {  
2     for(int i=0; i<s.size(); i++) {  
3         int minPos = // posicion del minimo elemento de s entre i y s.size()  
4         swap(s, i, minPos);  
5     }  
6 }
```

Ordenamiento por selección (Selection Sort)

- Podemos refinar un poco el código:

```
1 void selectionSort(vector<int> &s) {  
2   for(int i=0; i<s.size()-1; i++) {  
3     int minPos= findMinPosition(s, i, s.size());  
4     swap(s, i, minPos);  
5   }  
6 }
```

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

```
proc findMinPosition(in s : seq(Z), in d, h : Z, out min : Z){  
  Pre {0 ≤ d < h ≤ |s|}  
  Post {d ≤ min < h  
        ∧L (∀ i : Z)(d ≤ i < h →L s[min] ≤ s[i])}  
}
```

Buscar el Mínimo Elemento

7	5	10	1	12	5	15
↑	↑	↑	↑	↑	↑	↑
i	i	i	i	i	i	i
min	min		min			

Buscar el Mínimo Elemento

- ¿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?

$$fv = h - i$$

- ¿Cómo lo implementamos?

```
1 int findMinPosition(vector<int> &s, int d, int h) {  
2   int min = d;  
3   for(int i=d+1; i<h; i++) {  
4     if (s[i]<s[min]) {  
5       min = i;  
6     }  
7   }  
8   return min;  
9 }
```

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:

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

Buscar el Mínimo Elemento

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

```

1  int findMinPosition(vector<int> &s, int d, int h) {
2      int min = d;
3      for(int i=d+1; i<h; i++) {
4          if (s[i] < s[min]) {
5              min = i;
6          }
7      }
8      return min;
9  }
```

Correctitud: Buscar el Mínimo Elemento

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

Correctitud: Buscar el Mínimo Elemento

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

```

1  int findMinPosition(vector<int> &s, int d, int h) {
2      int min = d;
3      for(int i=d+1; i<h; i++) {
4          if (s[i] < s[min]) {
5              min = i;
6          }
7      }
8      return min;
9  }
```

- ▶ ¿ 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:

```

1  void selectionSort(vector<int> &s) {
2      for(int i=0; i<s.size(); i++) {
3          int minPos = findMinPosition(s, i, s.size());
4          swap(s, i, minPos);
5      }
6  }
```

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

Ordenamiento por selección (Selection Sort)

- ▶ $I \equiv \text{mismos}(s, S_0) \wedge ((0 \leq i \leq |s|) \wedge_L \text{ordenado}(\text{subseq}(s, 0, i)))$
- ▶ $f_v = |s| - i$

```

1 void selectionSort(vector<int> &s) {
2     for(int i=0; i<s.size(); i++) {
3         int minPos = findMinPosition(s, i, s.size());
4         swap(s, i, minPos);
5     }
6 }
```

- ▶ ¿ I se preserva en cada iteración (punto 2.)? **X**
- ▶ 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:

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

Correctitud: Ordenamiento por selección (Selection Sort)

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

Gráficamente:

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

Correctitud: Ordenamiento por selección (Selection Sort)

- ▶ $P_C \equiv i = 0 \wedge s = S_0$
- ▶ $Q_C \equiv \text{mismos}(s, S_0) \wedge \text{ordenado}(s)$
- ▶ $B \equiv i < |s|$
- ▶ $I \equiv \text{mismos}(s, S_0) \wedge ((0 \leq i \leq |s|) \wedge_L (\text{ordenado}(\text{subseq}(s, 0, i))) \wedge (\forall j, k : \mathbb{Z})((0 \leq j < i \wedge i \leq k < |s|) \rightarrow_L s[j] \leq s[k]))$
- ▶ $f_v = |s| - i$
- ▶ ¿ 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.)? **✓**

Correctitud: Ordenamiento por selección (Selection Sort)

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

```

1 void selectionSort(vector<int> &s) {
2     for(int i=0; i<s.size(); i++) {
3         int minPos = findMinPosition(s, i, s.size());
4         swap(s, i, minPos);
5     }
6 }
```

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

Ordenamiento por selección (Selection Sort)

```
1 int findMinPosition(vector<int> &s, int d, int h) {  
2     int min = d;  
3     for(int i=d+1; i<h; i++) {  
4         if (s[i] < s[min]) {  
5             min = i;  
6         }  
7     }  
8     return min;  
9 }  
10 void selectionSort(vector<int> &s) {  
11     for(int i=0; i<s.size(); i++) {  
12         int minPos = findMinPosition(s,i,s.size());  
13         swap(s, i, minPos);  
14     }  
15 }
```

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

Ordenamiento por selección (Selection Sort)

- ▶ ¿Cuántas iteraciones ejecuta este programa en peor caso?
 - ▶ Para ello contamos la cantidad de veces que se ejecuta el `if` de `min`

$$\text{ejecuciones}_{if} = \sum_{i=0}^{|s|-1} |s| - i = |s| \times |s| - \frac{(|s| - 1) \times |s|}{2} \leq (|s|)^2.$$

- ▶ Decimos que el algoritmo de ordenamiento por selección es un algoritmo **cuadrático** (¡más información en **algo2!**).
- ▶ ¿Puede ejecutarse una cantidad menor de veces?
 - ▶ Siempre se ejecuta la misma cantidad de veces. El peor caso es igual al mejor caso.

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).
- ▶ Ambas variantes también son algoritmos cuadráticos (iteran una cantidad cuadrática de veces).

Ordenamiento por inserción (Insertion Sort)

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

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

- ▶ 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)

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

```

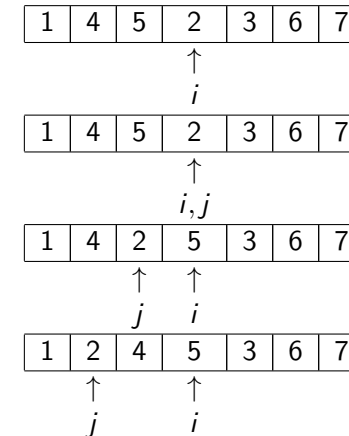
1 void insertionSort(vector<int> &s) {
2   for(int i=0; i<s.size(); i++) {
3     // Tenemos que preservar el invariante ...
4   }
5 }

```

- ▶ ¿ 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?

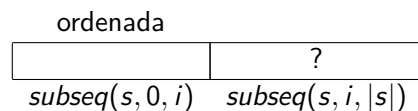
Ordenamiento por inserción (Insertion Sort)

Necesitamos desplazar $s[i]$ hasta una posición donde $\text{subseq}(s, 0, i)$ esté ordenada de vuelta.

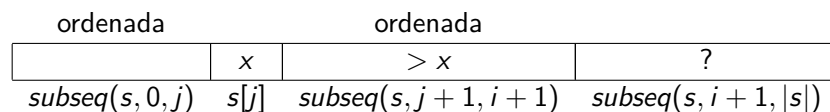


Ordenamiento por inserción (Insertion Sort)

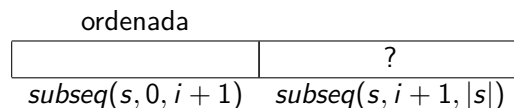
- ▶ Antes de comenzar el desplazamiento, tenemos que:



- ▶ Durante el desplazamiento, se cumple que que:



- ▶ Al finalizar el desplazamiento, nuevamente tenemos que:



Ordenamiento por inserción (Insertion Sort)

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

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

- ▶ ¿Cuál es la función variante de insert ?

$$fv = j$$

Ordenamiento por inserción (Insertion Sort)

- ¿Cuál es una posible implementación de insert?

```
1 void insert(vector<int> &s, int i) {  
2     for(int j=i; j>0 && s[j] < s[j-1]; j--) {  
3         swap(s, j, j-1);  
4     }  
5 }
```

- ¿Cuál es una posible implementación de insertSort?

```
1 void insertionSort(vector<int> &s) {  
2     for(int i=0; i<s.size(); i++) {  
3         insert(s,i);  
4     }  
5 }
```

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

Ordenamiento por inserción (Insertion Sort)

- ¿Cuántas veces se ejecuta el **swap** del ciclo interior?
 - ¡Depende de los datos!
- Analizamos el **peor caso** (es decir, que insert realice $i + 1$ iteraciones).

$$\text{ejecuciones}_{if} = \sum_{i=0}^{|s|} i + 1 = \frac{|s| \times (|s| + 1)}{2} + |s| \leq |s|^2$$

- El algoritmo de ordenamiento por inserción también es un algoritmo cuadrático (itera una cantidad cuadrática de veces)
- **Observación:** Selection sort e insertion sort se pueden generalizar a secuencias de tipo T con un predicado de orden \leq (no solamente funcionan con secuencias de \mathbb{Z})

Dutch National Flag Problem

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:

$\langle \text{White}, \text{Red}, \text{Blue}, \text{Blue}, \text{Red} \rangle$

El programa debe modificar la secuencia para que quede:

$\langle \text{Red}, \text{Red}, \text{White}, \text{Blue}, \text{Blue} \rangle$

Dutch National Flag Problem

- Si Red=0, White=1 y Blue=2, ¿Cuál sería la especificación del problema?
- **proc** *dutchNationalFlag*(inout $s : \text{seq}(\mathbb{Z})$) {
 Pre $\{s = S_0 \wedge (\forall e : \mathbb{Z})(e \in s \leftrightarrow (e = 0 \vee e = 1 \vee e = 2))\}$
 Post $\{ \text{mismos}(s, S_0) \wedge \text{ordenado}(s) \}$
}
- ¿Cómo podemos implementar una solución a este problema?
 - ¿Podemos usar algún algoritmo de ordenamiento que conozcamos? **Rta:** podemos usar insertionSort o selectionSort.
 - ¿Cuál es el peor caso? **Rta:** $(|s|)^2$
 - ¿Podemos hacer que tenga un peor caso mas eficiente?

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?
 - ▶ Quicksort y BubbleSort: Peor caso cuadrático $(|s|)^2$
 - ▶ Mergesort y Heapsort: Peor caso: $|s| \times \log_2(|s|)$
 - ▶ Counting sort (para secuencias de enteros). Peor caso: $|s|$
 - ▶ Radix sort (para secuencias de enteros). Peor caso: 2^{32}
- ▶ Bubble sort está en la práctica 8. El resto los van a ver en **algo2**.

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>