

## 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⟩){  
    |s| = |t| ∧  
    (∀ i : ℤ)(0 ≤ i < |s|  
        →L #apariciones(s, s[i]) = #apariciones(t, s[i]))  
}  
  
► fun #apariciones(s : seq⟨T⟩, e : T) : ℤ =  
    ∑i=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:**

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

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

## Ordenamiento por selección (Selection Sort)

- **Notación:**  $s[i, j] = \text{subseq}(s, i, j + 1)$ .  
 $s[i, j) = \text{subseq}(s, i, j)$ .
- **Observación:**  $\text{subseq}$  nunca se indefine (devuelve la secuencia vacía  $\langle \rangle$ )
- **Idea:** Seleccionar el mínimo elemento e **intercambiarlo** con la primera posición del vector. Repetir con el segundo, etc.

---

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

---

## Ordenamiento por selección (Selection Sort)

- Podemos refinar un poco el código:

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

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

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

## Buscar el Mínimo Elemento

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

- Necesitamos implementar la especificación de **minimo**:

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

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

- $B \equiv i < h$

- $I \equiv ?$

- En cada iteración del ciclo, la variable **result** contiene el índice del menor elemento encontrado hasta ahora. Es decir,  $s[\text{result}]$  el menor elemento de  $s[d, i)$ .

- ¡Esto nos proporciona el invariante del ciclo!

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

- $fv = h - i$

## Buscar el Mínimo Elemento

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

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

## Correctitud: Buscar el Mínimo Elemento

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

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

```
1 int minimo(vector<int> &s, int d, int h) {  
2   int result = s[d];  
3   for(int i=d+1; i<h; i++) {  
4     if (s[result] < s[i]) {  
5       result = i;  
6     }  
7   }  
8   return result;  
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 seleccion(vector<int> &s) {  
2   for(int i=0; i<s.size(); i++) {  
3     int pos = minimo(s, i, s.size());  
4     swap(s, i, pos);  
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,  $s[0, i)$  contiene los  $i$  primeros elementos ordenados! ¿Tenemos entonces el **invariante** del ciclo?
  - ▶  $I \equiv \text{mismos}(s, S_0) \wedge \text{ordenado}(s[0, i)) \wedge (0 \leq i \leq |s|)$
- ▶  $fv = |s| - i$

## Ordenamiento por selección (Selection Sort)

- ▶  $I \equiv 0 \leq i \leq |s| \wedge \text{mismos}(s, S_0) \wedge \text{ordenado}(s[0, i])$
- ▶  $fv = |s| - i$

---

```

1 void seleccion(vector<int> &s) {
2     for(int i=0; i<s.size(); i++) {
3         int pos = minimo(s, i, s.size());
4         swap(s, i, pos);
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:

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

## Correctitud: Ordenamiento por selección (Selection Sort)

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

Gráficamente:

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

## Correctitud: Ordenamiento por selección (Selection Sort)

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

- ▶ ¿ $I$  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.)? **✓**

## Correctitud: Ordenamiento por selección (Selection Sort)

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

---

```

1 void seleccion(vector<int> &s) {
2     for(int i=0; i<s.size(); i++) {
3         int pos = minimo(s, i, s.size());
4         swap(s, i, pos);
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 minimo(vector<int> &s, int d, int h) {  
2     int result = s[d];  
3     for(int i=d+1; i<h; i++) {  
4         if (s[result] < s[i]) {  
5             result = i;  
6         }  
7     }  
8     return result;  
9 }  
10 void seleccion(vector<int> &s) {  
11     for(int i=0; i<s.size(); i++) {  
12         int pos = minimo(s,i,s.size());  
13         swap(s, i, pos);  
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 **minimo**

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

- ▶ 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).

## Intervalo

Break!

## Ordenamiento por inserción (Insertion Sort)

- Veamos un segundo algoritmo, en el que usaremos como invariante únicamente que:

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

- Esto implica que (a diferencia del algoritmo de selection sort) en cada iteración los primeros  $i$  elementos están ordenados, sin ser necesariamente los  $i$  elementos más pequeños del vector.

## Ordenamiento por inserción (Insertion Sort)

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

---

```

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

---

- ¿ $I$  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?
    - ¡Insertando  $s[i]$  en la posición temporaria que le corresponda!

## Ordenamiento por inserción (Insertion Sort)

---

```

1 void insercion(vector<int> &s) {
2   for(int i=0; i<s.size(); i++) {
3     for(int j=i; j>0 && s[j] < s[j-1]; j--) {
4       swap(s, j, j-1);
5     }
6   }
7 }
```

---

$$I_{ext} \equiv \text{mismos}(s, S_0) \wedge \text{ordenado}(s[0, i]) \wedge (0 \leq i \leq |s|)$$

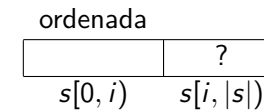
- ¿Cuál es el invariante del for interno?

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

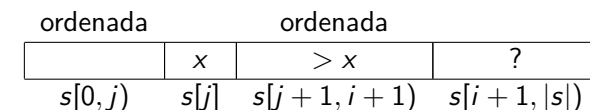
## Ordenamiento por inserción (Insertion Sort)

Gráficamente:

- Invariante del ciclo exterior



- Invariante del ciclo interior



## Ordenamiento por inserción (Insertion Sort)

```
1 void insercion(vector<int> &s) {  
2     for(int i=0; i<s.size(); i++) {  
3         for( int j=i; j>0 && s[j] < s[j-1] ; j--) {  
4             swap(s, j, j-1);  
5         }  
6     }  
7 }
```

- ▶ ¿Cómo son las funciones variantes de cada ciclo?

$$f_{v_{ext}} = |s| - i$$

$$f_{v_{int}} = j$$

## Ordenamiento por inserción (Insertion Sort)

```
1 void insercion(vector<int> &s) {  
2     for(int i=0; i<s.size(); i++) {  
3         for( int j = i; j>0 && s[j] < s[j-1] ; j--) {  
4             swap(s, j, j-1);  
5         }  
6     }  
7 }
```

- ▶ ¿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 el while interior 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 : seq<ℤ>){`  
 $\text{Pre } \{s = S_0 \wedge (\forall e : \mathbb{Z})(e \in s \leftrightarrow (e = 0 \vee e = 1 \vee e = 2))\}$   
 $\text{Post } \{ \text{mismos}(s, S_0) \wedge \text{ordenado}(s) \}$   
`}`
- ▶ ¿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?

## Dutch National Flag Problem

1. En una pasada contar la cantidad de apariciones de cada color: RED (0), WHITE (1) y BLUE (2), almacenándolas en una secuencia de tres elementos.
2. En una segunda pasada repoblar la secuencia usando las cantidades leídas.
3. ¿Qué cantidad de iteraciones se ejecutaría en peor caso?
  - ▶ **Rta:**  $2 * |s| \leq (|s|)^2$

## Dutch National Flag Problem

---

```

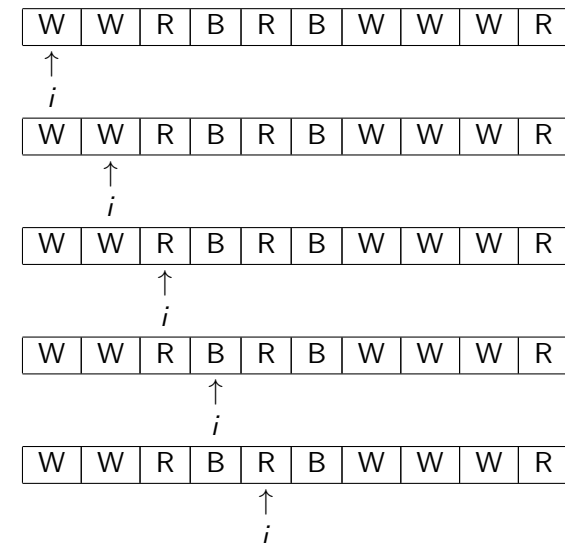
1  #define RED  0
2  #define WHITE 1
3  #define BLUE 2
4
5  void dutchNationalFlag(vector<int> &s) {
6      // contamos la cantidad de apariciones de cada color
7      vector<int> colorCount = fillColorCount(s);
8
9      // usamos la cantidad de apariciones para repoblar la secuencia
10     populate(s,colorCount);
11 }

```

---

## Dutch National Flag Problem

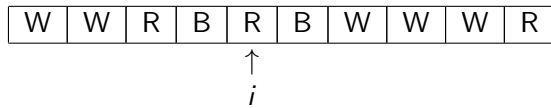
¿Qué invariante de ciclo podemos usar para contar la cantidad de apariciones de cada color?





## Dutch National Flag Problem

¿Qué invariante de ciclo podemos usar para contar la cantidad de apariciones de cada color?



$$I \equiv (0 \leq i \leq |s|) \wedge_L$$

$$\begin{aligned} & (colorCount[RED] = \#apariciones(subseq(s, 0, i), RED)) \wedge \\ & colorCount[WHITE] = \#apariciones(subseq(s, 0, i), WHITE) \wedge \\ & colorCount[BLUE] = \#apariciones(subseq(s, 0, i), BLUE)) \end{aligned}$$

¿Qué función variante podemos usar en la primer pasada?

$$fv \equiv |s| - i$$

## Dutch National Flag Problem

---

```
1 vector<int> fillColorCount(vector<int> &s) {  
2     ...  
3 }
```

---

## Dutch National Flag Problem

---

```
1 vector<int> fillColorCount(vector<int> &s) {  
2     vector<int> colorCount = {0,0,0};  
3     for(int i=0; i<s.size(); i++) {  
4         if (s[i]==RED) {  
5             colorCount[RED]++;  
6         } else if (s[i]==WHITE) {  
7             colorCount[WHITE]++;  
8         } else {  
9             colorCount[BLUE]++;  
10        }  
11    }  
12    return colorCount;  
13 }
```

---

¿Qué cantidad de iteraciones ejecuta `fillColorCount` en **peor caso**? **Rta:** Realiza  $3 + |s|$ -iteraciones en peor caso.

## Dutch National Flag Problem

Ahora nos falta repoblar la secuencia con los valores. ¿Qué invariante de ciclo tendríamos que respetar?

$$\begin{aligned} I & \equiv mismos(s, S_0) \wedge \\ & ((0 \leq i \leq |s|) \wedge_L \\ & (ordenado(subseq(s, 0, i))) \wedge \\ & colorCount[RED] = \#a...(S_0, RED) - \#a...(subseq(s, 0, i), RED)) \wedge \\ & colorCount[WHITE] = \#a...(S_0, WHITE) - \#a...(subseq(s, 0, i), WHITE)) \wedge \\ & colorCount[BLUE] = \#a...(S_0, BLUE) - \#a...(subseq(s, 0, i), BLUE))) \end{aligned}$$

¿Qué función variante podemos usar en la primer pasada?

$$fv \equiv |s| - i$$

## Dutch National Flag Problem

```
1 void populate(vector<FlagColor> &s, vector<int> &colorCount) {  
2  
3     for(int i=0; i<s.size(); i++) {  
4         if (colorCount[RED]>0) {  
5             s[i] = RED;  
6             colorCount[RED]--;  
7         } else if (colorCount[WHITE]>0) {  
8             s[i] = WHITE;  
9             colorCount[WHITE]--;  
10        } else {  
11            s[i] = BLUE;  
12            colorCount[BLUE]--;  
13        }  
14    }  
15 }
```

¿Qué cantidad de iteraciones ejecuta populate en **peor caso**?

**Rta:** Realiza  $|s|$ -iteraciones en peor caso.

## Dutch National Flag Problem

```
1 void dutchNationalFlag(vector<int> &s) {  
2     // realiza (3+|s|)-iteraciones  
3     vector<int> colorCount = fillColorCount(s);  
4  
5     // realiza |s|-iteraciones  
6     populate(s,colorCount);  
7 }
```

¿Qué cantidad de iteraciones ejecuta dutchNationalFlag en **peor caso**?

- ▶ **Rta:** Realiza  $3 + |s| + |s|$ -iteraciones en peor caso.
- ▶ Por lo tanto, a partir de  $|s| > 3$ , se cumple que  $3 + 2 * |s| < |s|^2$
- ▶ En otras palabras, en peor caso la implementación es mas eficiente que selectionSort o insertionSort

## 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 ( $|s|$ )<sup>2</sup>
  - ▶ 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>