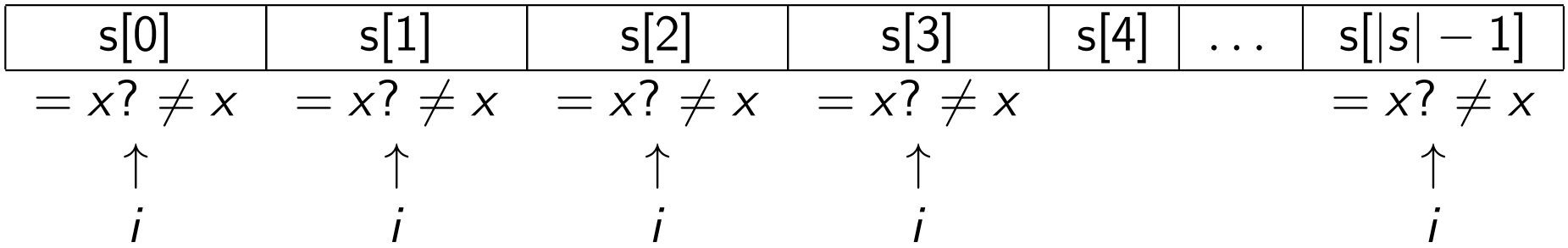
Algoritmos de búsqueda sobre secuencias

Algoritmos y Estructuras de Datos I

Búsqueda lineal

- Recordemos el problema de búsqueda por valor de un elemento en una secuencia.
- proc *contiene*(in $s : seq\langle\mathbb{Z}\rangle$, in $x : \mathbb{Z}$, out *result* : Bool){
  Pre $\{True\}$
  Post $\{result = true \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L s[i] = x)\}$
  }
- ¿Cómo podemos buscar un elemento en una secuencia?

Búsqueda lineal



- ¿Qué invariante de ciclo podemos proponer?

  $I \equiv 0 \leq i \leq |s| \wedge_L$
  $(\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$

- ¿Qué función variante podemos usar?

  $fv = |s| - i$

Búsqueda lineal

- Invariante de ciclo:

$$I \equiv 0 \leq i \leq |s| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$$

- Función variante:

$$fv = |s| - i$$

- ¿Cómo lo podemos implementar en C++?

```
bool contiene(vector<int> &s, int x) {
  int i = 0;
  while( i < s.size() && s[i] != x ) {
    i=i+1;
  }
  return i < s.size();
}
```

- ¿Es la implementación correcto con respecto a la especificación?

Recap: Teorema de corrección de un ciclo

- **Teorema.** Sean un predicado I y una función $fv : \mathbb{V} \to \mathbb{Z}$ (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:

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

Búsqueda lineal

- Para este programa, tenemos:
  - $P_C \equiv i = 0$,
  - $Q_C \equiv (i < |s|) \leftrightarrow (\exists j : \mathbb{Z})(0 \leq j < |s| \wedge_L s[j] = x)$.
  - $B \equiv i < |s| \wedge_L s[i] \neq x$
  - $I \equiv 0 \leq i \leq |s| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$
  - $fv = |s| - i$
- Ahora tenemos que probar que:
  1. $P_C \Rightarrow I$,
  2. $\{I \wedge B\}\ S\ \{I\}$,
  3. $I \wedge \neg B \Rightarrow Q_C$,
  4. $\{I \wedge B \wedge v_0 = fv\}\ \mathbf{S}\ \{fv < v_0\}$,
  5. $I \wedge fv \leq 0 \Rightarrow \neg B$,

Recap: Teorema de corrección de un ciclo

1. $P_C \Rightarrow I$,
2. $\{I \wedge B\}\, S\, \{I\}$,
3. $I \wedge \neg B \Rightarrow Q_C$,
4. $\{I \wedge B \wedge v_0 = fv\}\ \mathbf{S}\ \{fv < v_0\}$,
5. $I \wedge fv \leq 0 \Rightarrow \neg B$,

En otras palabras, hay que mostrar que:

- I es un invariante del ciclo (punto 1. y 2.)
- Se cumple la postcondición del ciclo a la salida del ciclo (punto 3.)
- La función variante es estrictamente decreciente (punto 4.)
- Si la función variante alcanza la cota inferior la guarda se deja de cumplir (punto 5.)

Corrección de búsqueda lineal

¿I es un invariante del ciclo?

$$I \equiv 0 \leq i \leq |s| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$$

- La variable i toma el primer valor 0 y se incrementa por cada iteración hasta llegar a $|s|$.
- $\Rightarrow 0 \leq i \leq |s|$
- En cada iteración, todos los elementos a izquierda de i son distintos de x
- $\Rightarrow (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$

Corrección de búsqueda lineal

¿Se cumple la postcondición del ciclo a la salida del ciclo?

$$I \equiv 0 \leq i \leq |s| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$$

$$Q_C \equiv (i < |s|) \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L s[i] = x)$$

- Al salir del ciclo, no se cumple la guarda. Entonces no se cumple $i < |s|$ o no se cumple $s[i] \neq x$
  - Si no se cumple $i < |s|$, no existe ninguna posición que contenga x
  - Si no se cumple $s[i] \neq x$, existe al menos una posición que contiene a x

Corrección de búsqueda lineal

¿Es la función variante estrictamente decreciente?

$$fv = |s| - i$$

- En cada iteración, se incremente en 1 el valor de i
- Por lo tanto, en cada iteración se reduce en 1 la función variante.

Corrección de búsqueda lineal

¿Si la función variante alcanza la cota inferior la guarda se deja de cumplir?

$$fv = |s| - i$$

$$B \equiv i < |s| \wedge_L s[i] \neq x$$

- Si $fv = |s| - i \leq 0$,entonces $i \geq |s|$
- Como siempre pasa que $i \leq |s|$, entonces es cierto que $i = |s|$
- Por lo tanto $i < |s|$ es falso.

Corrección de búsqueda lineal

- Finalmente,ahora que probamos que:
  1. $P_C \Rightarrow I$,
  2. $\{I \wedge B\}\ S\ \{I\}$,
  3. $I \wedge \neg B \Rightarrow Q_C$,
  4. $\{I \wedge B \wedge v_0 = fv\}\ \mathbf{S}\ \{fv < v_0\}$,
  5. $I \wedge fv \leq 0 \Rightarrow \neg B$,
- ...podemos por el teorema concluir que el ciclo termina y es correcto.

Búsqueda lineal

- Implementación:

```
bool contiene(vector<int> &s, int x) {
  int i = 0;
  while( i < s.size() && s[i] != x ) {
    i=i+1;
  }
  return i < s.size();
}
```

- Analicemos cuántas veces va a iterar este programa:

s	x	# iteraciones
$\langle\rangle$	1	0
$\langle 1\rangle$	1	0
$\langle 1,2\rangle$	2	1
$\langle 1,2,3\rangle$	4	3
$\langle 1,2,3,4\rangle$	4	3
$\langle 1,2,3,4,5\rangle$	-1	5

Búsqueda lineal

- ¿De qué depende cuántas veces se ejecuta el ciclo? Esto depende de
  - El tamaño de la secuencia
  - Si el valor buscado está o no contenido en la secuencia
- ¿Qué tiene que pasar para que el tiempo de ejecución sea el máximo posible?
  - El elemento no debe estar contenido.
- Esto representa el **peor caso** en tiempo de ejecución.

Complejidad computacional

Definición. La función de complejidad de un algoritmo es una función $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ tal que $f(n)$ es la cantidad de operaciones elementales que realiza el algoritmo en el peor caso para una entrada de tamaño n.

Algunas observaciones:

1. Medimos la cantidad de operaciones elementales en lugar del tiempo total.
2. Nos interesa el peor caso (el que genera la mayor cantidad de operaciones elementales) del programa.
3. El tiempo de ejecución se mide en función del tamaño de la entrada y no de la entrada particular.

Notación "O grande"

Definición. Si f y g son dos funciones, decimos que $f \in O(g)$ si existen $c \in \mathbb{R}$ y $n_0 \in \mathbb{N}$ tales que

$$f(n) \leq c\, g(n) \qquad \text{para todo } n \geq n_0.$$

Intuitivamente, $f \in O(g)$ si $g(n)$ "le gana" a $f(n)$ para valores grandes de n.

Ejemplos:

- Si $f(n) = n$ y $g(n) = n^2$, entonces $f \in O(g)$.
- Si $f(n) = n^2$ y $g(n) = n$, entonces $f \notin O(g)$.
- Si $f(n) = 100n$ y $g(n) = n^2$, entonces $f \in O(g)$.
- Si $f(n) = 4n^2$ y $g(n) = 2n^2$, entonces $f \in O(g)$ (y a la inversa).

Complejidad computacional

Utilizamos la notación "O grande" para expresar la función de complejidad computacional f de un algoritmo.

- Si $f \in O(n)$ (y $f \notin O(1)$) decimos que el programa es lineal.
- Si $f \in O(n^2)$ (y $f \notin O(n)$) decimos que el programa es cuadrático.
- Si $f \in O(n^3)$ (y $f \notin O(n^2)$) decimos que el programa es cúbico.
- En general, si $f \in O(n^k)$, decimos que el programa es polinomial.
- Si $f \in O(2^n)$ o similar, decimos que el programa es exponencial.

La búsqueda lineal tiene un tiempo de ejecución (de peor caso) perteneciente $O(n)$. Decimos también "el algoritmo es $O(n)$". ¿Se puede dar un algoritmos de búsqueda más eficiente?

Búsqueda sobre secuencias ordenadas

- Supongamos ahora que la secuencia está ordenada.
- proc *contieneOrdenada*(in $s : seq\langle\mathbb{Z}\rangle$, in $x : \mathbb{Z}$, out *result* : Bool){

  Pre {ordenado(s)}

  Post $\{result = true \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L s[i] = x)\}$

  }
- ¿Podemos aprovechar que la secuencia está ordenada para crear un programa más eficiente ?

Búsqueda sobre secuencias ordenadas

Podemos interrumpir la búsqueda tan pronto como verificamos que $s[i] \geq x$.

```
bool contieneOrdenada(vector<int> &s, int x) {
  int i = 0;
  while( i < s.size() && s[i] < x ) {
    i=i+1;
  }
  return (i < s.size() && s[i] == x);
}
```

¿Cuál es el tiempo de ejecución de peor caso?

Búsqueda sobre secuencias ordenadas

- Podemos interrumpir la búsqueda tan pronto como verificamos que $s[i] \geq x$.

Función `contieneOrdenado`	T_{exec}	máx.# veces
`int i = 0;`	c'_1	1
`while( i < s.size() && s[i] < x ) {`	c'_2	$1 + \|s\|$
`  i=i+1;`	c'_3	$\|s\|$
`}`		
`return (i < s.size() && s[i] == x);`	c'_4	1

- Sea n la longitud de s, ¿cuál es el tiempo de ejecución en el peor caso?

$$T_{contieneOrdenado}(n) = 1 * c'_1 + (1 + n) * c'_2 + n * c'_3 + 1 * c'_4$$

- ¿A qué O grande pertenece la función $T_{contieneOrdenado}(n)$?

$$T_{contieneOrdenado}(n) \in O(n)$$

Búsqueda sobre secuencias

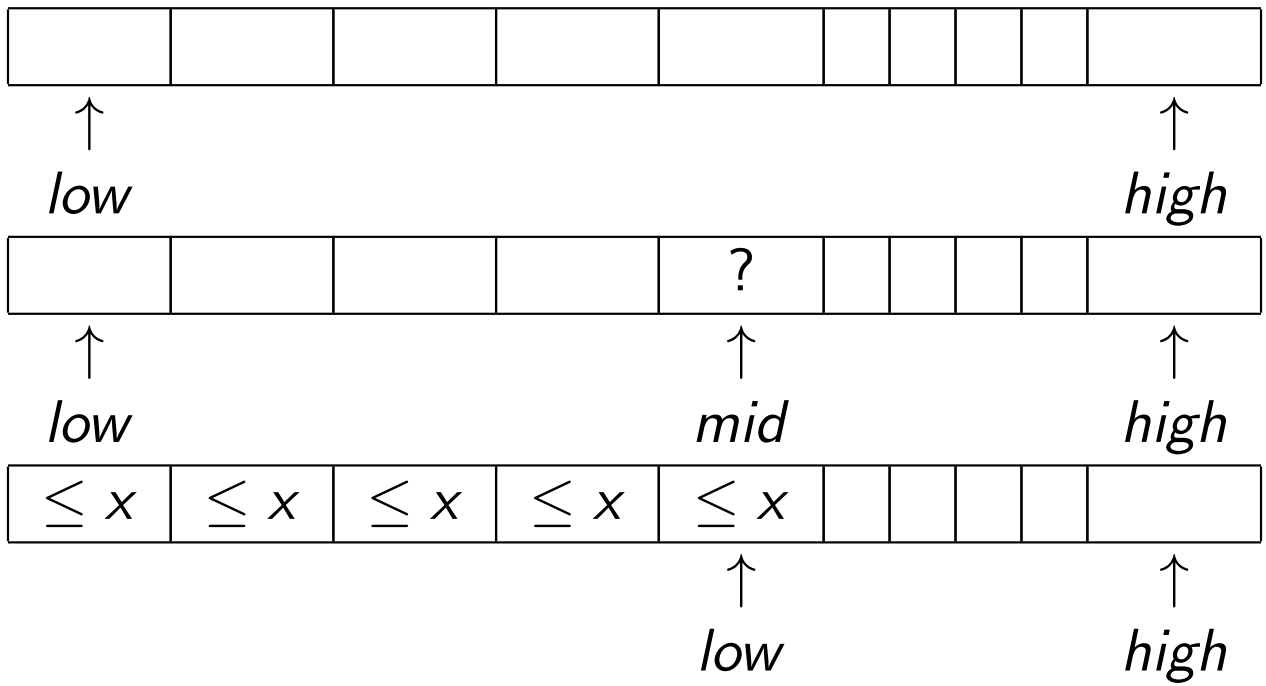
- $T_{contiene}(n) \in O(n)$
- $T_{contieneOrdenado}(n) \in O(n)$
- El tiempo de ejecución de peor caso de `contiene` y `contieneOrdenado` está acotado por la misma función $c * n$.
- Entonces ambas funciones crecen a la *misma* velocidad
  - Abuso de notación: podemos decir que ambos programas tienen el "mismo" tiempo de ejecución de peor caso

Búsqueda sobre secuencias ordenadas

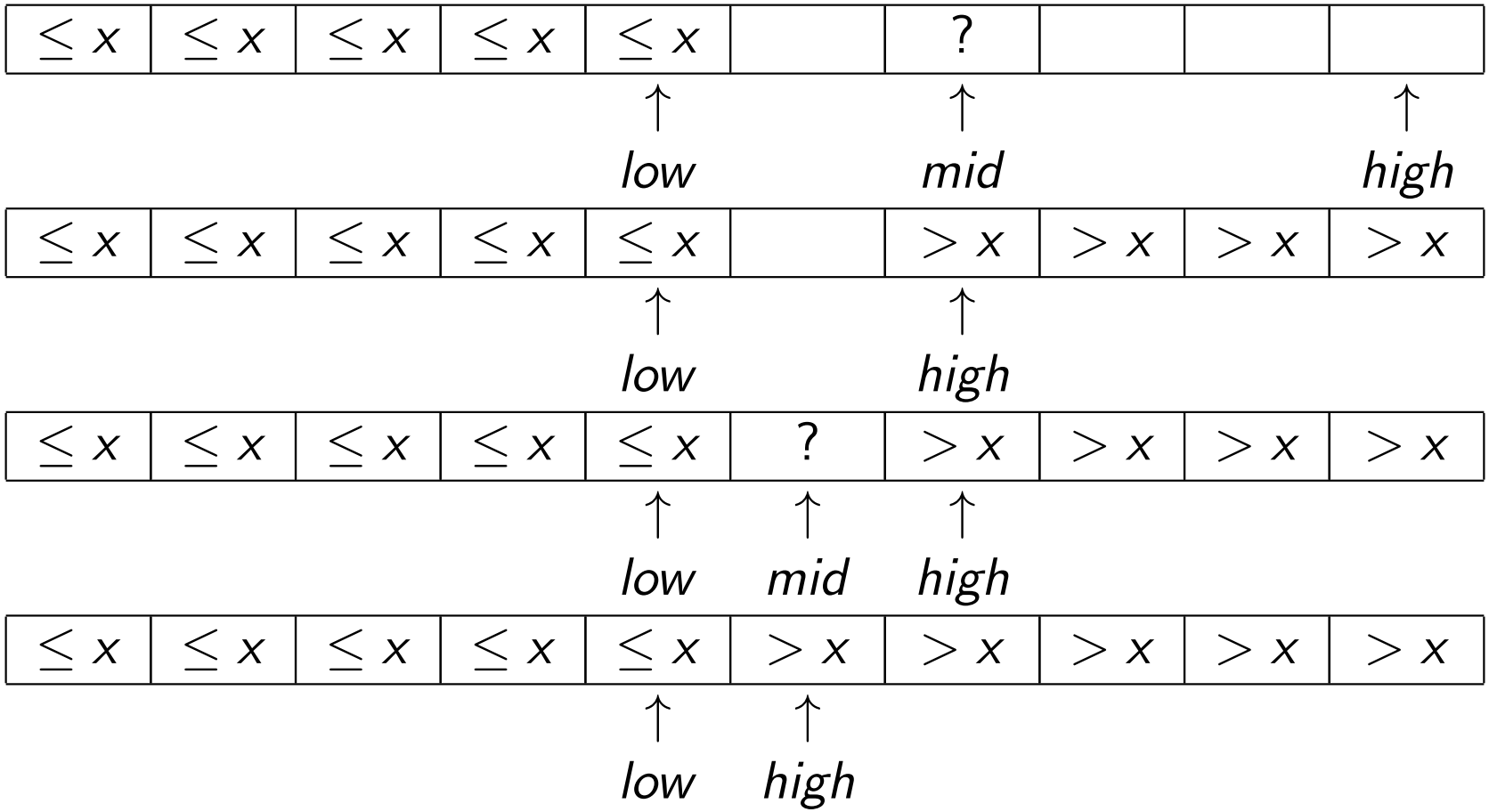
- ¿Podemos aprovechar el ordenamiento de la secuencia para mejorar el tiempo de ejecució de peor caso?
  - ¿Necesitamos iterar si $|s| = 0$? Trivialmente, $x \notin s$
  - ¿Necesitamos iterar si $|s| = 1$?Trivialmente, $s[0] == x \leftrightarrow x \in s$
  - ¿Necesitamos iterar si $x < s[0]$? Trivialmente, $x \notin s$
  - ¿Necesitamos iterar si $x \geq s[|s| - 1]$? Trivialmente, $s[|s| - 1] == x \leftrightarrow x \in s$

Búsqueda sobre secuencias ordenadas

Asumamos por un momento que $|s| > 1 \wedge_L (s[0] \leq x \leq s[|s| - 1])$

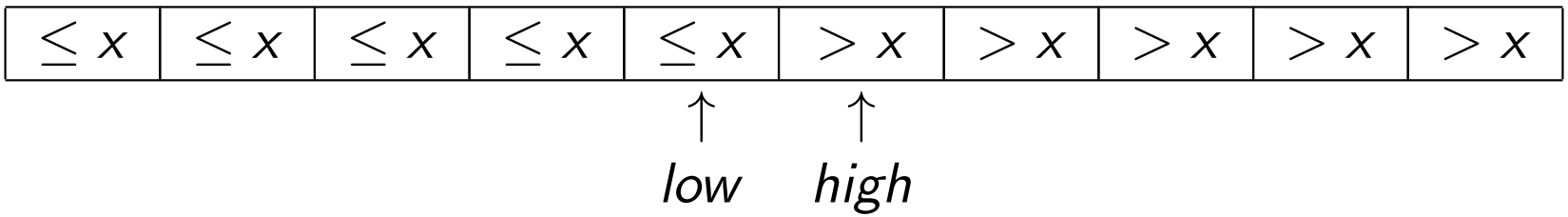


Búsqueda sobre secuencias ordenadas



Si $x \in s$, tiene que estar en la posición *low* de la secuencia.

Búsqueda sobre secuencias ordenadas



- ¿Qué invariante de ciclo podemos escribir?

$$I \equiv 0 \leq low < high < |s| \wedge_L s[low] \leq x < s[high]$$

- ¿Qué función variante podemos definir?

$$fv = high - low - 1$$

Búsqueda sobre secuencias ordenadas

```
bool contieneOrdenada(vector<int> &s, int x) {
  // casos triviales
  if (s.size()==0 ) {
    return false;
  } else if (s.size()==1) {
    return s[0]==x;
  } else if (x<s[0]) {
    return false;
  } else if (x≥s[s.size()-1]) {
    return s[s.size()-1]==x;
  } else {
    // casos no triviales
    ○ ..
  }
}
```

Búsqueda sobre secuencias ordenadas

```
  } else {
    // casos no triviales
    int low = 0;
    int high = s.size() - 1;
    while( low+1 < high ) {
      int mid = (low+high) / 2;
      if( s[mid] ≤ x ) {
        low = mid;
      } else {
        high = mid;
      }
    }
    return s[low] == x;
  }
}
```

A este algoritmo se lo denomina búsqueda binaria

Búsqueda binaria

- Veamos ahora que este algoritmo es correcto.

$$
\begin{array}{rcl}
P_C & \equiv & \mathit{ordenada}(s) \wedge (|s| > 1 \wedge_L s[0] \leq x \leq [|s| - 1]) \\
 & & \wedge \ \ \mathit{low} = 0 \wedge \mathit{high} = |s| - 1 \\
Q_C & \equiv & (s[\mathit{low}] = x) \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L s[i] = x) \\
B & \equiv & \mathit{low} + 1 < \mathit{high} \\
I & \equiv & 0 \leq \mathit{low} < \mathit{high} < |s| \ \wedge_L \ s[\mathit{low}] \leq x < s[\mathit{high}] \\
\mathit{fv} & = & \mathit{high} - \mathit{low} - 1
\end{array}
$$

Corrección de la búsqueda binaria

- ¿Es I un invariante para el ciclo?
  - El valor de *low* es siempre menor estricto que *high*
  - *low* arranca en 0 y sólo se aumenta
  - *high* arranca en $|s| - 1$ y siempre se disminuye
  - Siempre se respecta que $s[low] \leq x$ y que $x < s[high]$

- ¿A la salida del ciclo se cumple la postcondicion Q_C?
  - Al salir, se cumple que $low + 1 = high$
  - Sabemos que $s[high] > x$ y $s[low] <= x$
  - Como s está ordenada, si $x \in s$, entonces $s[low] = x$

Corrección de la búsqueda binaria

- ¿Es la función variante estrictamente decreciente?
  - Nunca ocurre que $low = high$
  - Por lo tanto, siempre ocurre que $low < mid < high$
  - De este modo, en cada iteración, o bien high es estrictamente menor, o bien low es estrictamente mayor.
  - Por lo tanto, la expresión $high - low - 1$ siempre es estrictamente menor.

- ¿Si la función variante alcanza la cota inferior la guarda se deja de cumplir?
  - Si $high - low - 1 \leq 0$, entonces $high \leq low + 1$.
  - Por lo tanto, no se cumple $(high > low + 1)$, que es la guarda del ciclo

Búsqueda binaria

- ¿Podemos interrumpir el ciclo si encontramos x antes de finalizar las iteraciones?
- Una posibilidad **no recomendada** (no lo hagan en casa!):
-
  - ..

```
while( low+1 < high) {
  int mid = (low+high) / 2;
  if( s[mid] < x ) {
    low = mid;
  } else if( s[mid] > x ) {
    high = low;
  } else {
    return true; // Argh!
  }
}
return s[low] == x;
}
```

Búsqueda binaria

- Una posibilidad **aún peor** (ni lo intenten!):

- ```
    bool salir = false;
    while( low+1 < high && !salir ) {
      int mid = (low+high) / 2;
      if( s[mid] < x ) {
        low = mid;
      } else if( s[mid] > x ) {
        high = mid;
      } else {
        salir = true; // Puaj!
      }
    }

    return s[low] == x || s[(low+high)/2] == x;
  }
  ```
  ```

Búsqueda binaria

- Si queremos salir del ciclo, el lugar para decirlo es ...
  la guarda!

- ```
  while( low+1 < high && s[low] != x ) {
    int mid = (low+high) / 2;
    if( s[mid] ≤ x ) {
      low = mid;
    } else {
      high = mid;
    }
  }
  return s[low] == x;
}
```

- Usamos fuertemente la condición $s[low] \leq x < s[high]$ del invariante.
```

Búsqueda binaria

- ¿Cuántas iteraciones realiza el ciclo (en peor caso)?

Número de iteración	$high - low$
0	$\lvert s\rvert - 1$
1	$\cong (\lvert s\rvert - 1)/2$
2	$\cong (\lvert s\rvert - 1)/4$
3	$\cong (\lvert s\rvert - 1)/8$
$\vdots$	$\vdots$
t	$\cong (\lvert s\rvert - 1)/2^t$

- Sea t la cantidad de iteraciones necesarias para llegar a $high - low = 1$.

$1 = (|s|-1)/2^t$ entonces $2^t = |s|-1$ entonces $t = \log_2(|s|-1)$.

Luego, el tiempo de ejecución de peor caso de la búsqueda binaria es $O(\log_2 |s| - 1) = O(\log_2 |s|)$.

Búsqueda binaria

- ¿Es mejor un algoritmo que ejecuta una cantidad logarítmica de iteraciones?

$\lvert s\rvert$	Búsqueda Lineal	Búsqueda Binaria
10	10	4
10^2	100	7
10^6	$1,000,000$	21
$2{,}3 \times 10^7$	$23,000,000$	25
7×10^9	$7,000,000,000$	33 (!)

- Sí! Búsqueda binaria es más eficiente que búsqueda lineal
- Pero, requiere que la secuencia esté ya ordenada.

Bibliografía

- David Gries - The Science of Programming
  - Chapter 16 - Developing Invariants (Linear Search, Binary Search)
- Cormen et al. - Introduction to Algorithms
  - Chapter 2.2 -Analyzing algorithms
  - Chapter 3 - Growth of Functions