

Algoritmos y Estructuras de Datos I

Segundo cuatrimestre de 2019

Departamento de Computación - FCEyN - UBA

Algoritmos de búsqueda sobre secuencias

1

Búsqueda lineal

- Sea el problema de búsqueda por valor de un elemento en una secuencia.
- `proc contiene(in s : seq<ℤ>, in x : ℤ, out result : Bool){
 Pre { True }
 Post { result = true ↔ (∃j : ℤ)(0 ≤ j < |s| ∧ s[j] = x) }
}`
- ¿Cómo podemos buscar un elemento en una secuencia?

2

Búsqueda lineal

s[0]	s[1]	s[2]	s[3]	s[4]	...	s[s - 1]
= x?	= x?	= x?	= x?			= x?
≠ x	≠ x	≠ x	≠ x			≠ x
↑	↑	↑	↑			↑
j	j	j	j			j

- ¿Qué invariante de ciclo podemos proponer?

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

- ¿Qué función variante podemos usar?

$$fv = |s| - j$$

3

Búsqueda lineal

- Invariante de ciclo:

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

- Función variante:

$$fv = |s| - j$$

- ¿Cómo lo podemos implementar en C++?

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

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

4

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:

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

5

Búsqueda lineal

- Para el ciclo de este programa, tenemos:

- $P_C \equiv j = 0$,
- $Q_C \equiv (j < |s|) \leftrightarrow (\exists k : \mathbb{Z})(0 \leq k < |s| \wedge_L s[k] = x)$.
- $B \equiv j < |s| \wedge_L s[j] \neq x$
- $I \equiv 0 \leq j \leq |s| \wedge_L (\forall k : \mathbb{Z})(0 \leq k < j \rightarrow_L s[k] \neq x)$
- $fv = |s| - j$

- 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. $\{I \wedge B \wedge V_0 = fv\} S \{fv < V_0\}$,
5. $I \wedge fv \leq 0 \Rightarrow \neg B$,

6

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. $\{I \wedge B \wedge V_0 = fv\} S \{fv < V_0\}$,
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.)

7

Corrección de búsqueda lineal

¿ I es un invariante del ciclo?

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

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

8

Corrección de búsqueda lineal

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

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

$$\mathbb{Q}_C \equiv (j < |s|) \leftrightarrow (\exists k : \mathbb{Z})(0 \leq k < |s| \wedge_L s[k] = x)$$

$$\mathbb{B} \equiv j < |s| \wedge_L s[j] \neq x$$

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

9

Corrección de búsqueda lineal

¿Es la función variante estrictamente decreciente?

$$fv = |s| - j$$

- ▶ En cada iteración, se incrementa en 1 el valor de j
- ▶ Por lo tanto, en cada iteración se reduce en 1 la función variante.

10

Corrección de búsqueda lineal

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

$$fv = |s| - j$$

$$\mathbb{B} \equiv j < |s| \wedge_L s[j] \neq x$$

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

11

Corrección de búsqueda lineal

▶ Finalmente, ahora que probamos que:

1. $\mathbb{P}_C \Rightarrow \mathbb{I}$,
2. $\{\mathbb{I} \wedge \mathbb{B}\} \mathbf{S} \{\mathbb{I}\}$,
3. $\mathbb{I} \wedge \neg \mathbb{B} \Rightarrow \mathbb{Q}_C$,
4. $\{\mathbb{I} \wedge \mathbb{B} \wedge V_0 = fv\} \mathbf{S} \{fv < V_0\}$,
5. $\mathbb{I} \wedge fv \leq 0 \Rightarrow \neg \mathbb{B}$,

- ▶ ...podemos por el teorema concluir que el ciclo termina y es correcto.

12

Búsqueda lineal

► Implementación:

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

13

Búsqueda lineal

- ¿De qué depende cuántas veces se ejecuta el ciclo?
 - Del tamaño de la secuencia
 - De 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.

14

Búsqueda lineal

- Dada una secuencia cualquiera, ¿cuál es el tiempo máximo (i.e. el peor caso) que puede tardar en ejecutar el programa?

Función contiene	T_{exec}	máx.# veces
int j = 0;	c_1	1
while(j < s.size() && s[j] != x) {	c_2	$1 + s $
j=j+1;	c_3	$ s $
}		
return j < s.size();	c_4	1

- ¿Cuál es el tiempo máximo de ejecución para una secuencia s ?
Si definimos a n como la longitud de s , entonces:

$$T_{contiene}(n) = 1 * c_1 + (1 + n) * c_2 + n * c_3 + 1 * c_4$$

15

Tiempo de ejecución de peor caso

Definición. El **tiempo de ejecución** de un programa es una función $f : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ tal que $f(n)$ es el costo de ejecutar el programa en el peor caso para una entrada de tamaño n .

Algunas observaciones:

1. Cada línea de programa cuesta un tiempo constante c_j (excepto llamados a otros programas)
2. Nos interesa el peor caso (i.e. el que tarda más) del programa.
3. Podemos medir la cantidad de operaciones en lugar del tiempo total.
4. El tiempo de ejecución se mide en función del tamaño de la entrada y no de la entrada particular.

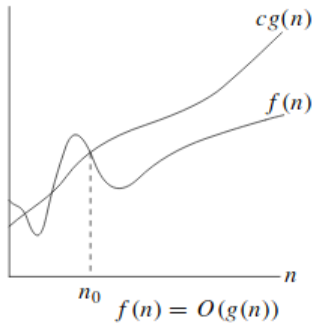
16

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

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

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



17

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

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

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).
- ▶ Si $T_{\text{contiene}}(n) = 1 * c_1 + (1 + n) * c_2 + n * c_3 + 1 * c_4$,
 - ▶ entonces $T_{\text{contiene}}(n) \in O(n)$

18

Tiempo de Ejecución de peor caso

Utilizamos la notación "O grande" para expresar el tiempo de ejecución de peor caso f de los programas.

- ▶ 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 $O(n)$ con n el tamaño de la entrada. Decimos también "el algoritmo es $O(n)$ ". ¿Se puede dar un algoritmo de búsqueda más eficiente?

19

Búsqueda sobre secuencias ordenadas

- ▶ Supongamos ahora que la secuencia está **ordenada**.
- ▶

```
proc contieneOrdenada(in s : seq<Z>, in x : Z, out result : Bool){  
    Pre {ordenado(s)}  
    Post {result = true ↔ (∃j : Z)(0 ≤ j < |s| ∧ s[j] = x)}  
}
```
- ▶ ¿Podemos aprovechar que la secuencia está ordenada para crear un programa más **eficiente**?

20

Búsqueda sobre secuencias ordenadas

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

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

¿Cuál es el tiempo de ejecución de peor caso?

21

Búsqueda sobre secuencias ordenadas

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

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

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

22

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

23

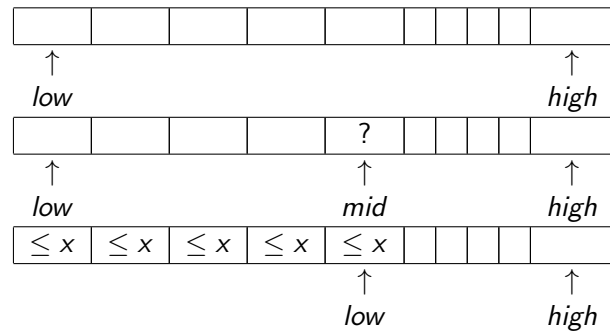
Búsqueda sobre secuencias ordenadas

- ¿Podemos aprovechar el ordenamiento de la secuencia para mejorar el tiempo de ejecución 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, $s[|s| - 1] == x \leftrightarrow x \in s$

24

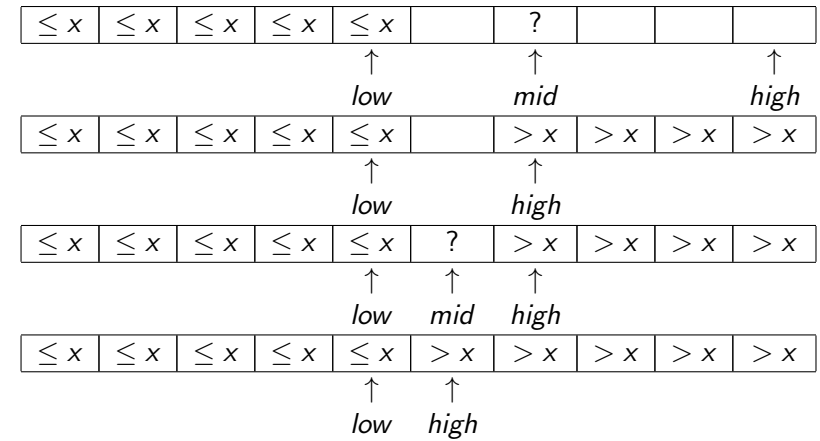
Búsqueda sobre secuencias ordenadas

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



25

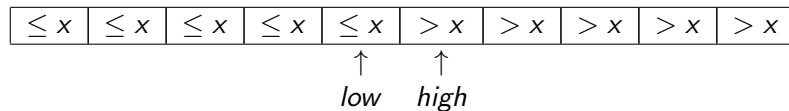
Búsqueda sobre secuencias ordenadas



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

26

Búsqueda sobre secuencias ordenadas



► ¿Qué invariante de ciclo podemos escribir?

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

► ¿Qué función variante podemos definir?

$$fv = high - low - 1$$

27

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
        ◦ ..
    }
}
```

28

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**

29

Búsqueda binaria

- Veamos ahora que este algoritmo es correcto.

$$\mathbb{P}_C \equiv \text{ordenada}(s) \wedge (|s| > 1 \wedge_L s[0] \leq x \leq s[|s| - 1]) \\ \wedge \text{low} = 0 \wedge \text{high} = |s| - 1$$

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

$$\mathbb{B} \equiv \text{low} + 1 < \text{high}$$

$$\mathbb{I} \equiv 0 \leq \text{low} < \text{high} < |s| \wedge_L s[\text{low}] \leq x < s[\text{high}]$$

$$\text{fv} = \text{high} - \text{low} - 1$$

30

Corrección de la búsqueda binaria

- $\mathbb{I} \equiv 0 \leq \text{low} < \text{high} < |s| \wedge_L s[\text{low}] \leq x < s[\text{high}]$ ¿Es 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 $s[\text{low}] \leq x$ y que $x < s[\text{high}]$
- $\mathbb{Q}_C \equiv (s[\text{low}] = x) \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L s[i] = x)$ ¿A la salida del ciclo se cumple la postcondición \mathbb{Q}_C ?
 - Al salir, se cumple que $\text{low} + 1 = \text{high}$
 - Sabemos que $s[\text{high}] > x$ y $s[\text{low}] \leq x$
 - Como *s* está ordenada, si $x \in s$, entonces $s[\text{low}] = x$

31

Corrección de la búsqueda binaria

- ¿Es la función variante estrictamente decreciente?
 - Nunca ocurre que $\text{low} = \text{high}$
 - Por lo tanto, siempre ocurre que $\text{low} < \text{mid} < \text{high}$
 - De este modo, en cada iteración, o bien *high* es estrictamente menor, o bien *low* es estrictamente mayor.
 - Por lo tanto, la expresión $\text{high} - \text{low} - 1$ siempre es estrictamente menor.
- ¿Si la función variante alcanza la cota inferior la guarda se deja de cumplir?
 - Si $\text{high} - \text{low} - 1 \leq 0$, entonces $\text{high} \leq \text{low} + 1$.
 - Por lo tanto, no se cumple $(\text{high} > \text{low} + 1)$, que es la guarda del ciclo

32

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;  
}
```

33

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;
}
```

34

## 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.

35

Búsqueda binaria

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

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

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

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

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

36

Búsqueda binaria

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

s	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
- **Peeero**, requiere que la secuencia esté ya ordenada.

37

Bonus Track: Nearly all binary searches are broken!



38

Nearly all binary searches are broken!

- En 2006 comenzaron a reportarse **accesos fuera de rango** a vectores dentro de la función `binarySearch` implementada en las bibliotecas estándar de Java.
- En la implementación en Java, los enteros tienen precisión finita, con rango $[-2^{31}, 2^{31} - 1]$.
- Si `low` y `high` son valores muy grandes, al calcular `k` se produce **overflow**.
- La falla estuvo *dormida* muchos años y se manifestó sólo cuando el tamaño de los vectores creció a la par de la capacidad de memoria de las computadoras.
- Bugfix: Computar `k` evitando el overflow:

```
int mid = low + (high-low)/2;
```

39

Conclusiones

- La búsqueda binaria implementada en Java estaba formalmente demostrada ...
- ... pero la demostración suponía enteros de precisión infinita (en la mayoría de los lenguajes imperativos son de precisión finita).
 - En AED1 no nos preocupan los problemas de aritmética de precisión finita (+Info: Orga1).
 - Es importante validar que las hipótesis sobre las que se realizó la demostración valgan en la implementación (aritmética finita, existencia de acceso concurrente, multi-threading, etc.)

40

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