# Análisis de Algoritmos

*UNAM, Facultad De Ciencias.*

**Profesora.** María de Luz Gasca Soto
**Ayudante.** Enrique Ehecatl Hernández Ferreiro
**Ayudante.** Brenda Margarita Becerra Ruíz

## Tarea 5: Búsquedas

**Alumna:**
Villafán Flores María Fernanda (Ciencias de la Computación).

27 de octubre de 2022
Ciudad de México.

1. Dada un arreglo de $n$ enteros, $A[0..n-1]$, tal que $\forall\, i, 0 \leq i \leq n$, se tiene que $|A[i] - A[i+1]| \leq 1$; si $A[0] = x$ y $A[n-1] = y$, se tiene que $x < y$.
   Diseñar un algoritmo de búsqueda eficiente, de orden logarítmico, que localice al índice $j$ tal que $a[j] = z$, para un valor dado de $z$, $x \leq z \leq y$. **Justificar.**

## Respuesta.

Primero, veamos que si $\big|A[i] - A[i+1]\big| \leq 1$, significa que el arreglo $A$ puede tener elementos repetidos. Entonces, un ejemplo de $A$ podría ser:

$$A = [1, 1, 2, 3, 4, 5, 6, 6]$$

, donde $A[0] = A[1] = 1 \longrightarrow \big|A[0] - A[1]\big| = 0 \quad \text{y} \quad A[3] = 3,\ A[4] = 4 \longrightarrow \big|A[3] - A[4]\big| = 1$.
Así, vamos a diseñar un algoritmo para localizar al índice $j$ tal que:

$$A[j] = z, \text{ con } x \leq z \leq y \text{ (donde } A[0] = x \text{ y } A[n-1] = y)$$

Entonces,

- ***Precondiciones:***
  - $A \neq [\ ]$, es decir, $|A| = n$.
  - $\big|A[i] - A[i+1]\big| \leq 1$, con $0 \leq i \leq n$.
  - Si $A[0] = x$ y $A[n-1] = y$, entonces $x < y$.
  - Los elementos de $A$ son números enteros y comparables.

- ***Algoritmo:***

  Se utilizará el *Algoritmo de Búsqueda Binaria Iterativa* modificado de tal forma que si el elemento a buscar está repetido en el arreglo, el algoritmo regresa el primer índice donde aparece dicho elemento.

  Entonces,

```
// PreC: X[a,b] es un arreglo ordenado, no vacio y finito de enteros
// PostC: X no sufre cambios;
         regresa la posicion i, si el elemento z es encontrado;
         regresa −1 si z no se encuentra.

int BBinaria(int z; data_array X) {
    int max, mitad, min                        // variables temporales
    min = a;      max = b;
    int indice = −1;

    while (min ≤ max) {
        mitad = (max − min) / 2 + min;         // calcula mitad
        if ( z < X[mitad]) {                   // busca a la izquierda
            then max = mitad − 1;
```

```
        } else if (z == X[mitad]) {
            max = mitad - 1;
            indice = mitad;                    // busqueda exitosa
        } else {                               // busca a la derecha
            min = mitad + 1;
        }
    }
    return indice;
} // end BBinaria.
```

- ***Postcondiciones:***

  El índice $j$ tal que $A[j] = z$ para un valor dado de $z$, con $x \leq z \leq y$ (donde $A[0] = x$ y $A[n-1] = y$).

Sabemos que este algoritmo toma tiempo $O(log_2(n))$, puesto que en cada iteración reducimos la longitud del espacio de búsqueda a la mitad. Además, la modificación que se hizo al algoritmo no hace que aumente el desempeño computacional porque únicamente se agregó la variable *indice* y se modificaron los valores que almacenan las variables *min, max* y *mitad* (estos cambios toman tiempo $O(1)$ en realizarse).

∎

2. Suponga que se está usando un programa que manipula textos muy grandes, como un procesador de palabras. El programa toma como entrada un texto, representado como una secuencia de carácteres y produce alguna salida.
   Si en algún momento el programa encuentra un error del cual no puede recuperarse y además no puede indicar qué error es o dónde está, entonces la única acción que el programa toma es escribir *ERROR* y abortar el proceso. Supóngase que el error es local, esto es, se tiene una cadena en particular del texto la cual, por alguna extraña razón, al programa *no le gusta*. El error es independiente del contexto en el cual aparece la cadena "ofensiva". Diseñar una estrategia logarítmica para localizar la fuente del error.

## Respuesta.

Primero, para este ejercicio vamos a suponer que el programa que funciona como un procesador de palabras lee el texto que se le da como entrada en tiempo $O(1)$.
Entonces, la estrategia será:

- Tomamos cada cadena separada por espacios en el texto original y metemos cada una de ellas en una secuencia $S$.

- Creamos dos subsecuencias de $S$ tales que cada una tenga una mitad de los elementos de $S$, es decir, la primera subsecuencia tendrá la primera mitad de los elementos de $S$ (la mitad de la izquierda); y la segunda subsecuencia tendrá la segunda mitad de los elementos de $S$ (la mitad de la derecha).

- Luego decidimos pasarle al programa la primera subsecuencia.

★ Si el programa nos dice que en la primera subsecuencia **NO** se encuentra la cadena "ofensiva", entonces significa que se encuentra en la segunda subsecuencia.

- Si el programa nos dice que en la primera subsecuencia se encuentra la cadena "ofensiva", entonces ya localizamos la fuente del *error*.

Ahora si se da el caso [★], podemos hacer el mismo proceso que hicimos con la secuencia $S$, es decir, a la segunda subsecuencia la dividimos en dos subsecuencias tales que cada una tenga una mitad de los elementos de la segunda subsecuencia.
Por tanto, notamos que este proceso lo podemos realizar así sucesivamente con las mitades de las subsecuencias que vayamos haciendo.
Por lo tanto, este proceso toma tiempo $O(log_2(n))$ en ejecutarse ya que vamos dividiendo una secuencia en mitades cada vez más pequeñas hasta dar con la fuente del *error*.

■

3. Consideremos el siguiente juego, entre dos personas:
El jugador $J_A$ piensa un número entero en un rango. El jugador $J_B$ intenta encontrar tal número haciendo preguntas de la forma:
¿ Es el número menor que $x$? o ¿ Es mayor que $y$ ?
El objetivo es realizar el menor número de preguntas.
Se supone que nadie hace trampa.
**a)** Diseñar una buena estrategia para el juego... cuando el jugador $J_A$ indica un rango específico, digamos de 1 a $N$.
En este caso, ¿Cuál resulta ser la complejidad del algoritmo? **Justificar.**
**b)** Diseñar una buena estrategia para el juego... cuando el jugador $J_A$ **no** indica el rango del número que pensó. ¿Cuál es la complejidad del algoritmo propuesto? **Justificar.**

## Respuesta.

***Inciso (a)*** La estrategia sería:
Cuando el jugador $J_A$ indica el rango $[1..N]$, el jugador $J_B$ calcula el elemento que se encuentra a la mitad de la siguiente forma:

$$mid = \left\lceil \frac{(1+N)}{2} \right\rceil$$

y pregunta si el número que pensó el jugador $J_A$ es igual a $mid$.

★ Si resulta ser verdadero, entonces el jugador $J_B$ adivinó el número y gana.

★ Si resulta ser falso, entonces el jugador $J_B$ pregunta si el número que pensó el jugador $J_A$ es mayor que $mid$.

  - Si es verdadero, vuelve a calcular el valor de $mid$ de la siguiente forma:

$$mid = \left\lceil \frac{(mid+N)}{2} \right\rceil$$

◦ Si es falso, vuelve a calcular el valor de $mid$ de la siguiente forma:

$$mid = \left\lceil \frac{(1+mid)}{2} \right\rceil$$

★ Lo siguiente que hace el jugador $J_B$ es preguntar si el número que pensó el jugador $J_A$ es $mid$, y regresa a hacer los casos [★].

Notemos que el jugador $J_B$ realizará este proceso hasta que adivine el número que pensó el jugador $J_A$.

Ahora, la complejidad de esta estrategia es $O(log_2(n))$ ya que cada vez que el jugador $J_B$ pregunta si el número que pensó el jugador $J_A$ es mayor al valor de $mid$ que calculó, descarta la mitad de los números que están en el rango que le dió el jugador $J_A$.

***Inciso (b)*** La estrategia sería:
Como en este caso, el jugador $J_A$ **NO** indica el rango del número que pensó, el jugador $J_B$ propone un número $n = 2^k$ y le pregunta al jugador $J_A$ si el número que pensó es igual a $n$.

- Si resulta ser verdadero, entonces el jugador $J_B$ adivinó el número y gana.
- Si resulta ser falso, entonces el jugador $J_B$ pregunta si el número que pensó el jugador $J_A$ es menor que $n$.
  - Si es verdadero, el jugador $J_B$ decide realizar el mismo proceso que se explicó en el ***inciso a)*** utilizando el rango $[1..n] = [2^0..2^k]$.
  - Si es falso, significa que el número que pensó el jugador $J_A$ es mayor que $n$ y por tanto, el jugador $J_B$ duplica el valor de $n$ a $2n$ y le pregunta al jugador $J_A$ si el número que pensó es menor o igual a $2n$.
    - ◇ Si resulta ser verdadero, entonces el jugador $J_B$ decide realizar el mismo proceso que se explicó en el ***inciso a)*** utilizando el rango $[n..2n] = [2^k..2^{k+1}]$.
    - ◇ Si resulta ser falso, entonces el jugador $J_B$ duplica el valor de $2n$ a $4n$ y le pregunta al jugador $J_A$ si el número que pensó es menor o igual a $4n$.....

Notemos que el jugador $J_B$ realizará este proceso de duplicar el valor de $n$ hasta que pueda "acotar" el rango de búsqueda y logre adivinar el número que pensó el jugador $J_A$.

Por último, el jugador $J_B$ utiliza tantas comparaciones con el número que pensó el jugador $J_A$ como intervalos creados, los cuales son de la forma $[2^i..2^{i+1}]$. Así, tenemos que se forman $log_2(n) = k$ intervalos. Más aún, en uno de los intervalos utilizamos la estrategia del ***inciso a)***, que ya vimos que toma tiempo $O(log_2(n))$ en ejecutarse. Por lo tanto, la complejidad de esta estrategia es $O(log(n))$.

■

4. Consideremos el Algoritmo de Búsqueda por Interpolación.
   **(a)** Presentar un ejemplo de al menos 300 datos para el cual el algoritmo termine la búsqueda (exitosa o no) en pocas iteraciones. Ejemplificar.
   **(b)** Dar un ejemplo de, al menos, 300 datos para el cual el algoritmo termine la búsqueda (exitosa o no) en muchas iteraciones. Ejemplificar.
   Justificar la respuesta, en ambos casos.

---

## Respuesta.

***Inciso (a)*** La secuencia $S$ tendrá 300 datos, los cuales serán los primeros 300 números enteros que vayan de 1 en 1. Es decir, todos los números $j$ tal que $j = 1, 2, 3, \ldots, 300$.

Ahora, sabemos que el *Algoritmo de Búsqueda por Interpolación* es muy eficiente cuando la secuencia $S$ de entrada consiste de elementos distribuidos en forma equidistante.
En este caso, tenemos una secuencia de este tipo porque la distancia entre un elemento, el elemento predecesor a él y el elemento sucesor a él siempre será 1.

Veamos un ejemplo de ejecución:
Sea $z = 84$ el elemento a buscar en la secuencia $S$.
Entonces, aplicamos la búsqueda con los siguientes valores:

$$izq = 1, \;\; der = 300, \;\; z = 84, \;\; S[1] = 1, \;\; S[300] = 300$$

$$\begin{aligned}
p &= izq + \left\lceil \frac{(z - S[izq])(der - izq)}{S[der] - S[izq]} \right\rceil \\
&= 1 + \left\lceil \frac{(84-1)(300-1)}{300-1} \right\rceil \\
&= 1 + \left\lceil \frac{(84-1)\cancel{(300-1)}}{\cancel{300-1}} \right\rceil \\
&= 1 + \left\lceil (84-1) \right\rceil \\
&= 1 + \left\lceil 83 \right\rceil \\
&= 1 + 83 \\
&= 84
\end{aligned}$$

Por lo tanto, tenemos que $p = 84$.
Esto significa que el algoritmo sólo tuvo que calcular la posición $p$ una vez.

***Inciso (b)*** La secuencia $S$ tendrá 300 datos, los cuales serán:

$$S : [1, 2, 4, 7, 11, 16, 22, 29, 37, 46, 56, \ldots, 44851]$$

, donde la distancia entre un elemento y el elemento siguiente va aumentando en 1 cada vez. Es decir, si en la posición 1 tenemos $S[1] = 1$, entonces en la posición 2 tenemos $S[2] = 2$. Luego, en la posición 3 tenemos $S[3] = 4$, en la posición 4 tenemos $S[4] = 7$, y así sucesivamente.

Ahora, sabemos que el *Algoritmo de Búsqueda por Interpolación* es muy eficiente cuando la secuencia $S$ de entrada consiste de elementos distribuidos en forma equidistante.
En este caso, **NO** tenemos una secuencia de este tipo porque la distancia entre un elemento, el elemento predecesor a él y el elemento sucesor a él **NO** es la misma.

Como ya vimos más arriba, en la posición 2 tenemos $S[2] = 2$, en la posición 3 tenemos $S[3] = 4$ y en la posición 4 tenemos $S[4] = 7$. Así, la diferencia entre la posición 2 y la posición 3 es: $4 - 2 = 2$; y la diferencia entre la posición 3 y la posición 4 es: $7 - 4 = 3$.

Veamos un ejemplo de ejecución:
Sea $z = 7$ el elemento a buscar en la secuencia $S$.
Entonces, aplicamos la búsqueda con los siguientes valores:

$$izq = 1, \ \ der = 300, \ \ z = 7, \ \ S[1] = 1, \ \ S[300] = 44851$$

$$\begin{aligned} p &= izq + \left\lceil \frac{(z - S[izq])(der - izq)}{S[der] - S[izq]} \right\rceil \\ &= 1 + \left\lceil \frac{(7-1)(300-1)}{44851 - 1} \right\rceil \\ &= 1 + \left\lceil \frac{(6)(299)}{44850} \right\rceil \\ &= 1 + \left\lceil \frac{1794}{44850} \right\rceil \\ &= 1 + \left\lceil 0.04 \right\rceil \\ &= 1 + 1 \\ &= 2 \end{aligned}$$

Tenemos que $p = 2$, pero $S[2] = 2 \neq 7 = z$.
Ahora, aplicamos la búsqueda con los siguientes valores:

$$izq = 2, \ \ der = 300, \ \ z = 7, \ \ S[2] = 2, \ \ S[300] = 44851$$

$$\begin{aligned} p &= izq + \left\lceil \frac{(z - S[izq])(der - izq)}{S[der] - S[izq]} \right\rceil \\ &= 2 + \left\lceil \frac{(7-2)(300-2)}{44851 - 2} \right\rceil \\ &= 2 + \left\lceil \frac{(5)(298)}{44849} \right\rceil \\ &= 2 + \left\lceil \frac{1490}{44849} \right\rceil \\ &= 2 + \left\lceil 0.033 \right\rceil \\ &= 2 + 1 \\ &= 3 \end{aligned}$$

Tenemos que $p = 3$, pero $S[3] = 4 \neq 7 = z$.
Ahora, aplicamos la búsqueda con los siguientes valores:

$$izq = 3, \ \ der = 300, \ \ z = 7, \ \ S[3] = 4, \ \ S[300] = 44851$$

$$
\begin{aligned}
p &= izq + \left\lceil \frac{(z - S[izq])(der - izq)}{S[der] - S[izq]} \right\rceil \\
&= 3 + \left\lceil \frac{(7-4)(300-3)}{44851 - 4} \right\rceil \\
&= 3 + \left\lceil \frac{(3)(297)}{44847} \right\rceil \\
&= 3 + \left\lceil \frac{891}{44847} \right\rceil \\
&= 3 + \left\lceil 0.019 \right\rceil \\
&= 3 + 1 \\
&= 4
\end{aligned}
$$

Por lo tanto, tenemos que $p = 4$ y $S[4] = 7 = z$.
Con lo anterior, podemos notar que para esta secuencia las posiciones obtenidas por interpolación avanzan de uno en uno, lo que se debe a que la distancia entre los datos no es la misma. Con esto, nos podemos imaginar que para elementos más grandes en la secuencia, el algoritmo toma tiempo $O(n)$ en encontrarlo porque se recorre cada elemento en la secuencia.

$\blacksquare$