BUSQUEDA BINARIA

Sabemos que la complejidad temporal de un algoritmo muestra la dependencia del tiempo que demora un algoritmo respecto al tamaño de los datos de entrada. En el caso de la complejidad lineal el tiempo que demora un algoritmo es proporcional del tamaño "N" de sus datos de entrada; sin embargo el tiempo que demora el algoritmo de búsqueda binaria **no depende directamente del tamaño "N" de sus datos de entrada** sino del **$\log_2 N$**. Un ejemplo que demuestra esto:

- **Se tiene un array ordenado de 16 elementos (N=16) y se buscará un valor en ese array.**
- En el algoritmo de búsqueda binaria se compara el valor buscado con aquel de una posición central del array, puede ser el de la posición 8, como estamos en el peor de los casos no lo va encontrar en esa posición, por tanto divide al array en 2 partes de 8 items cada una y prosigue la búsqueda en una de las partes resultantes, así seguirá buscando hasta que no se pueda dividir más el array porque la parte a dividir solo tiene un elemento.

- Se entiende el patrón de búsqueda y división en el peor de los casos como sigue:

Tamaño del Array por analizar	¿Se encontró el valor buscado en el elemento central?	Tamaño del nuevo array por analizar luego de dividir en dos el Array analizado.
16 items	No	8 items
8 items	No	4 items
4 items	No	2 items
2 items	No	1 item
1 item	No	El algoritmo termina porque ya no se puede dividir más el array

- Con un valor de N=16 según el patrón recurrente de la tabla la **cantidad de búsquedas no exitosas** son 5, es decir 4 en donde se divide el array y 1 para salir del algoritmo: 4+1. Lo anterior se puede poner en forma logarítmica y en función de N sería: $5 = \log_2 \mathbf{16} + 1 \rightarrow \log_2 \mathbf{N} + 1 \in O(\log N)$.

- Ahora para cualquier valor de N que no es potencia de 2 por ejemplo 30, entonces se puede estimar el número de búsquedas de la siguiente manera:

Tamaño del Array por analizar	¿Se encontró el valor buscado en el elemento central?	Tamaño del nuevo array por analizar luego de dividir en dos el Array analizado.
30 items	No	15 items
15 items	No	7 items
7 items	No	3 items
3 item	No	1 item
1 item	No	El algoritmo termina porque ya no se puede dividir más el array

  - Si N=30
    - Para N=16 → N° de búsquedas = 5
    - **Para N= 30 → N° de búsquedas = 5**
    - Para N=32 → N° de búsquedas = 6
    - Entonces $2^{(5-1)} <= 30 < 2^5$, **donde 5 ese el número de búsquedas para N=30**,

luego nos libramos de las bases $(5-1) <= \log_2 30 < 5 \rightarrow$ **$5-1 <= \log_2 30$**, lo que quiere decir que para vincular $\log_2 30$ con el valor 5 lo único que tengo que hacer es aplicar la operación suelo(el valor entero inferior) y luego sumar 1: **$5-1 = \text{suelo}(\log_2 30)$**, sumando 1 a ambos tenemos: $5 =$ **$\text{suelo}(\log_2 30) + 1$** $\rightarrow$ $\text{suelo}(\log_2 \mathbf{N}) + 1 \in O(\log N)$.

- Generalizando para cualquier N
  - Entonces $2^{(k-1)} <= N < 2^k$, **donde k ese el número de búsquedas para N**, luego nos libramos de las bases $(k-1) <= \log_2 N < k \rightarrow (k-1) <= \log_2 N$, lo que quiere decir que para vincular $\log_2 N$ con el valor k lo único que tengo que hacer es aplicar la operación suelo y luego sumar 1: $(k-1) = \text{suelo}(\log_2 N)$, sumando 1 al primer par de términos tenemos: $k =$ **$\text{suelo}(\log_2 N) + 1$** $\rightarrow$ $\text{suelo}(\log_2 \mathbf{N}) + 1 \in O(\log N)$.