**Notación Asintótica**

Para entender el concepto de la notación asintótica debemos comprender dos conceptos básicos. Primero, el tiempo que se tarda un algoritmo dependiendo del tamaño de su entrada. Para esto tomamos en cuenta que el número máximo de intentos en las búsquedas lineales y las búsquedas binarias aumentan a medida que la longitud del arreglo aumenta. Siguiendo esta idea podemos concluir que el tiempo de ejecución del algoritmo es una función del tamaño de su entrada.

Segundo debemos saber en qué tan rápido crece una función por su tamaño de entrada, esto se llama tasa de crecimiento del tiempo de ejecución. Para ser mas eficaces con este concepto, tenemos que simplificar la función y extraer la parte mas importante. Tomemos de ejemplo un algoritmo que corre con una entrada de tamaño , se tarda instrucciones de maquina es ser completado. El termino , sin importar el valor de n, va a crecer a un ritmo mas acelerado que los términos .

Gráfico, Gráfico de líneas

Descripción generada automáticamente  
*Ilustración 2: Grafico tiempo de ejecución*

Se puede concluir que el tiempo de ejecución crece como , descartando el coeficiente y los términos restantes, siempre que el tiempo de ejecución sea para siempre habrá un valor de para que sea mayor al resto de términos.

Al descartar estos términos no significativos y con coeficientes constantes, nos enfocamos en la tasa de crecimiento de un algoritmo, en este momento utilizamos la notación asintótica. Tiene tres formas, la notación grande (Big), notación O grande (Big O) y notación Ω grande (Big Ω).

**Notación θ grande (Big θ):**

Tenemos como ejemplo el código:

**void** **busqueda\_lineal**(std::string funcion, **char** valor) {

**for** (**int** intentos = **0**; intentos < funcion.length(); guess++) {

**if** (funcion[intentos] === valor) {

**return** intentos;//valor encontrado y su posición

}

}

**return** -**1**;// no se encontro el valor

};

En este código el tamaño del string sería el cual es el número máximo de veces que se puede ejecutar el bucle for.

Cada que el bucle for itera, tiene que hacer varios cálculos:

* Compara “intentos” con el tamaño del string.
* Compara el valor de la posición del string con el valor.
* Regresar el valor de intentos (si es que encuentra el valor).
* Incrementar el valor de intentos.

Cada calculo toma un tiempo constante y si el bucle itera veces, entonces el tiempo para todas las iteraciones sería donde es la suma de los tiempos para el cálculo en una iteración del bucle. no tiene un valor fijo dado que depende de muchos factores. es el tiempo que el bucle for y también es una constante. Por tanto, el tiempo de búsqueda lineal, en el caso de que no se encuentre el valor, es de .

Para valores menores de , no es importante el tiempo de ejecución. Pero cuando es lo suficientemente grande utilizamos Theta de , o (), y se concluye que el tiempo de ejecución estará en el intervalo inferior de y .

Gráfico, Gráfico de líneas

Descripción generada automáticamente  
Ilustración 3: Tasa de crecimiento Big Theta.

En el calculo de Theta, no importan las unidades de tiempo que usemos, al utilizar la notación grande, decimos que tenemos una *cota* *asintóticamente ajustada* sobre el tiempo de ejecución. Asintóticamente porque solo importan los valores grandes de , Cota ajustada porque ajustamos el tiempo de ejecución dentro del rango de una constante mayor y menor.

Las funciones en notación asintótica se clasifican de mayor a menor dependiendo que tan rápido crecen. Es así como la primera crece a un ritmo lento y la ultima a un ritmo bastante acelerado:

1. (1)
2. ()
3. ()
4. ()
5. ()
6. ()
7. ()
8. ())
9. )

**Notación O grande (Big O):**

Utilizamos Big O para acotar asintóticamente el crecimiento de factores dentro de constantes por arriba y por abajo. Aunque también se pueden acotar solo por arriba.

Por ejemplo, el peor tiempo de ejecución de una búsqueda binaria es , aunque no siempre se ejecuta en ese tiempo. Si se encuentra el valor en la primera iteración entonces es valido decir que se ejecuto en un tiempo . En conclusión, el tiempo de ejecución nunca es peor que pero si puede ser mejor.

Usamos Big O significa que el tiempo de ejecución crece a lo más por una constate dada, pero este tiempo también puede crecer más lentamente.

Un tiempo de ejecución para una lo suficientemente grande, el tiempo de ejecución es para una constante de esta forma:  
Gráfico, Gráfico de líneas

Descripción generada automáticamente  
Ilustración 4: Tasa de crecimiento Big O.

Utilizamos Big O para cotas superiores asintóticas, ya que la cota es para el tiempo de ejecución por arriba de entradas con entradas suficientemente grandes.

Entonces, si comparamos la definición de Big Theta y Big O vemos que tienen similitudes, pero Big O se concentra únicamente en la cota de arriba. Se puede decir que el tiempo de ejecución en una búsqueda binaria va a ser siempre para cualquier valor de n.  
Una búsqueda binaria puede ejecutarse siempre en un tiempo pero no siempre se ejecuta en un tiempo .

Big O dará un valor que puede ser mayor al tiempo de ejecución, pero esta en el rango, podemos utilizar una analogía como ejemplo. Si tenemos 10 dólares en el bolsillo y afirmamos a un amigo que tenemos menos de 100 dólares, la afirmación va a ser correcta pero no muy precisa. Así mismo funciona la notación Big O.

**Notación Omega grande (Big Ω)**

En la notación Omega grande decimos que el algoritmo toma por lo menos una cierta cantidad de tiempo, para una suficientemente grande, el tiempo de ejecución es por lo menos para una constante . El tiempo de ejecución se grafica de la forma:

Gráfico, Gráfico de líneas

Descripción generada automáticamente  
Ilustración 4: Tasa de crecimiento Big Ω.

A diferencia de Big O, Big Ω utiliza los límites asintóticos inferiores, ya que acota los crecimientos por debajo de un lo suficientemente grande.

Así como implica , también implica , por ende, el peor caso de búsqueda binaria para Big Ω es

Con la misma analogía del dinero en el bolsillo utilizado en Big O, podemos decir que, “tengo por lo menos 1 dólar en el bolsillo”. Sigue siendo correcto, pero no muy preciso, de igual forma es el caso de la búsqueda binaria de Big Ω al afirmar que el mejor de los casos es, es correcto, pero no preciso pues sabemos que hay al menos un tiempo constante.

**Bibliografía:**

Cormen T, Leiserson, Rivest and Stein. (2002) *Introduction to Algorithms, 2nd edition.* Disponible en: <https://athena.nitc.ac.in/summerschool/Files/clrs.pdf>