Análisis de Complejidad

<mark>Por Ariel Parra. </mark>

La eficiencia de algoritmos consiste en el tiempo de ejecucion y la cantidad de recursos consumidos como la memoria.

Esta se puede medir en funcion de su eficiencia, el costo de escribirlo, leerlo y modificarlo.

Complejidad en el Tiempo

En la mayoria de algortimos el tiempo de ejecucion depende de la cantidad de elementos y no de su magnitud.

Por ejemplo: [1000000000,20000000000000,3000000000000] < [1,2,3,4,5,6,7,8,9,10]

Funcion de tiempo de ejecucion de un algoritmo con n entradas, T(n), esta funcion se expresa sin unidades.

Tal que n: numero de operaciones elementales echas por un algortimo.

Calculo por casos

<mark>Ejecucion del mejor de los casos:</mark> Es el numero de operaciones mas favorables en la ejecucion, la desventaja es que es muy optimista.

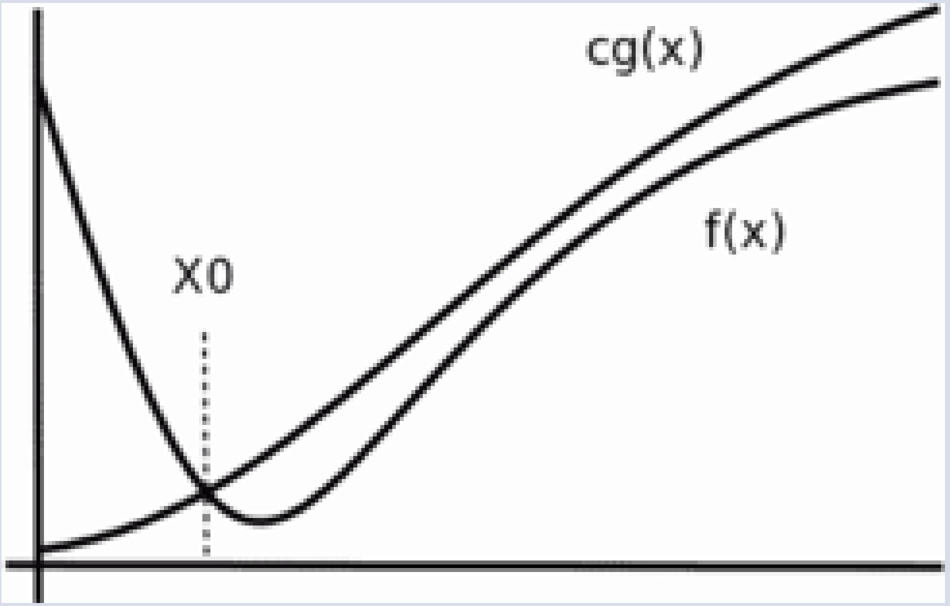
<mark>Ejecucion promedio:</mark> Utiliza la media de tiempos de la ejecucion del algortimo retornando la complejidad, la desventaja es que no siempre hay suficiente informacion para el calculo.

<mark>Ejecucion del peor de los casos:</mark> Es el mas recomendado ya que se contemplan todos los casos posinles al suponer el peor como el tiempo de ejecucion, para esto se usa la notacion de Big O.

<mark>Big O</mark>

<mark>Cota superior asintótica</mark>

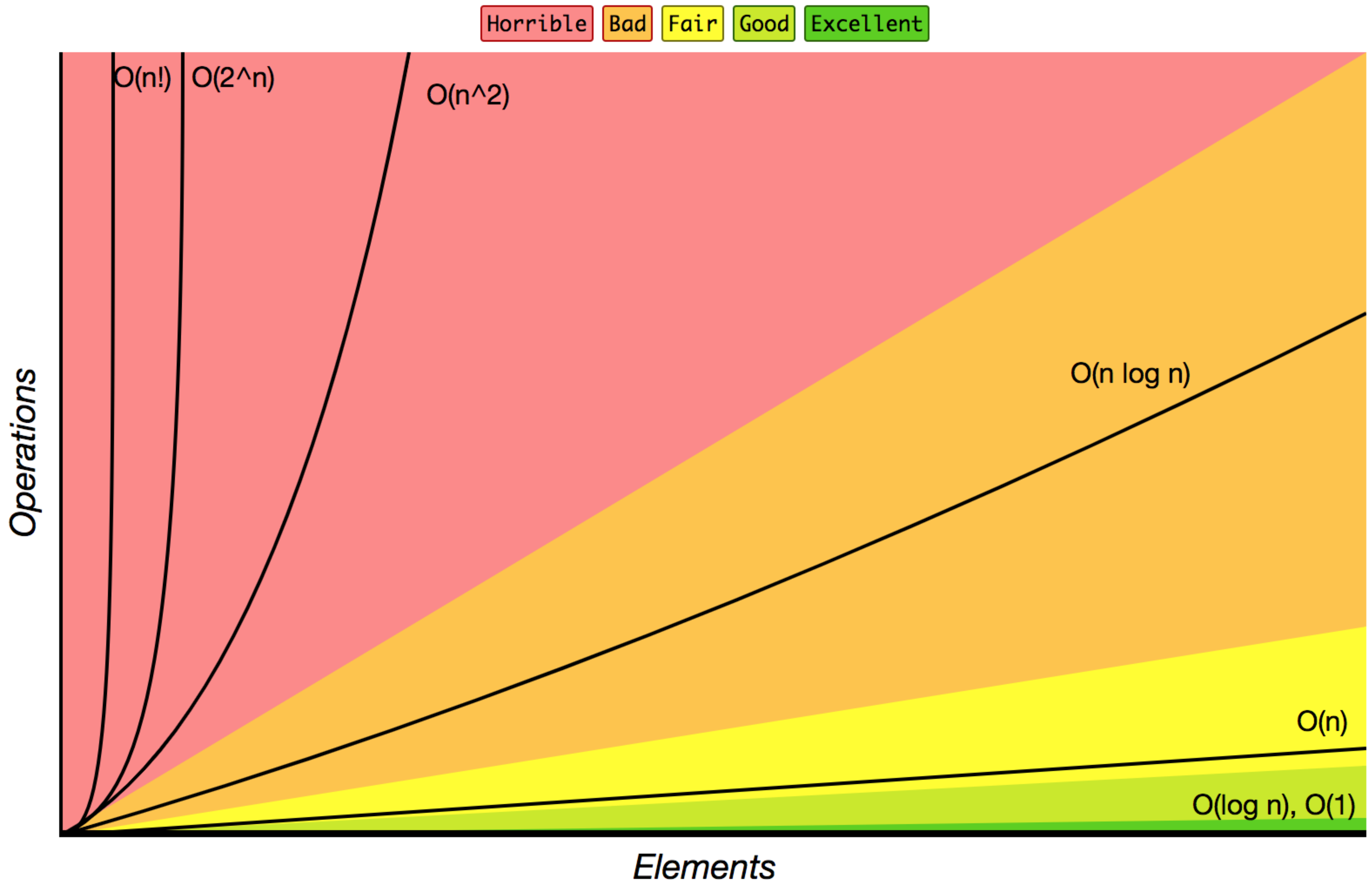
Define una cota superior o igual a la funcion del tiempo T(n) a partir de X0 tambien conocido como Xn la funcion del tiempo siempre sera igual a nuestra cota superior, en la imagen la cota superior seria cg(x) y f(x) representaria nuestra funcion de tiempo T(n). En palabras simples, definimos una funcion que siempre sera superior o igual a nuestra funcion del tiempo.



Analisis de algoritmos

notación	nombre
O(1)	constante
O(log n)	logarítmica
O(n)	lineal
O(n · log n)	lineal logarítmica o casi-lineal
O(n^2)	cuadrática
O(n^k)	potencial (k siendo cualquier constante)
O(k^n)	exponencial (k usualmente siendo 2 y n>1)
O(n!)	factorial

Big-O Complexity Chart



Analisis de Complejidad por funcion

Para determinar la complejidad se suman las complejidades por cada linea o funcion y se simplifican al mayor exponente.

<mark>Ejemplo de complejidad O(1):</mark>

```
int n=1000;         //es O(1), ya que n ya esta declarada
if(n%2==0)          //O(1)
    cout<<"par";    //O(1)
else                //O(1)
    cout<<"impar";  //O(1)
```

Complejidad = 1 + 1 + 1 + 1 + 1 = O(5) = O(1);

<mark>Ejemplo de complejidad O(n):</mark>

```
int n=0;                            // O(1)
cin>>n;                             // O(1)
for(int i=0;i<n;i++){               //es O(n) ya que n es el limite
    if(i%2==0)                      // O(n)
        cout<<i<<" es par";         // O(n)
    else                            // O(n)
        cout<<i<<" es impar";// O(n)
}
```

Complejidad = $1 + 1 + n + n + n + n + n = O(2 + 5n) = O(n)$;

<mark>Otro ejemplo de complejidad O(n):</mark>

```
int n=0, j=0;                              //O(1)
for(int i=0;i<n;i++){                      //O(n)
    if(i%2==0)                             //O(n)
        cout<<i<<" es par";                //O(n)
    else                                   //O(n)
        cout<<i<<" es impar";              //O(n)
}
while(j<n){                                //O(n)
    if(i%3==0)                             //O(n)
        cout<<i<<" es mutliplo de 3";      //O(n)
    else                                   //O(n)
        cout<<i<<" no es multimplo d";     //O(n)
    j++;                                   //O(n)
}
```

Complejidad = 1 + n + n + ... + n = O(1+ 11n) = O(n)

<mark>Ejemplo de complejidad O(log(n)):</mark>

```
int n=0;                      //O(1)
cin>>n;                       //O(1)
for(int i=0;i<n;i*=2){        //O(log (n)) ya incrementa con multiplicaciones en lugar de sumas
    if(i%2==0)                //O(log (n))
        cout<<i<<" es par";   //O(log (n))
    else                      //O(log (n))
        cout<<i<<" es impar"; //O(log (n))
}
```

Complejidad = 1 + 1 + log(n) + log(n) + log(n) + log(n) + log(n) = O(2 + 6log(n)) = O(log(n))

<mark>Ejemplo de complejidad O(n^2):</mark>

```
int n=0;                          //O(1)
cin>>n;                           //O(1)
for(int i=0;i<n;i++){             //O(n)
    for(int j=0;j<n;j++){ //O(n^2)
        cout<<i<<j<<endl; //O(n^2)
    }
}
```

Complejidad = 1 + 1 + n + n * n + n^2 = O(2 + n + 2n^2) = O(n^2)

<mark>Ejemplo de complejidad O(n^k):</mark>

```
int n=0;                                                                //O(1)
cin>>n;                                                                 //O(1)
for(int i=0;i<n;i++){                                                   //O(n)
    for(int j=0;j<n;j++){                                               //O(n^2)
        for(int ca=0;ca<n;ca++){                                        //O(n^3)
            for(int cb=0;cb<n;cb++){                                    //O(n^4)
                ...                                                     //O(n^5)
                    ...                                                 //O(n^6)
                        ...                                             //O(n^7)
                            cout<<i<<j<<ca<<cb<<...<<...<<...<<k<<endl; //O(n^k)
            }
        }
    }
}
```

Complejidad = $1 + 1 + n + n * n + n^3 \ldots + n^k = O(2 + n + n^2 + \ldots n^k) = O(n^k)$

<mark>Ejemplo de complejidad O(n(log(n))):</mark>

```
int n=0;                                    //O(1)
cin>>n;                                     //O(1)
for(int j=0;j<n;j++){                       //O(n)
    for(int i=0;i<n;i*=2){                  //O(n(log(n)))
        if(i%2==0)                          //O(n(log(n)))
            cout<<i<<" es par";             //O(n(log(n)))
        else                                //O(n(log(n)))
            cout<<i<<" es impar";           //O(n(log(n)))
    }
}
```

$\text{Complejidad} = 1 + 1 + n + n * \log(n) + n(\log(n)) + n(\log(n)) + n(\log(n)) + n(\log(n))$

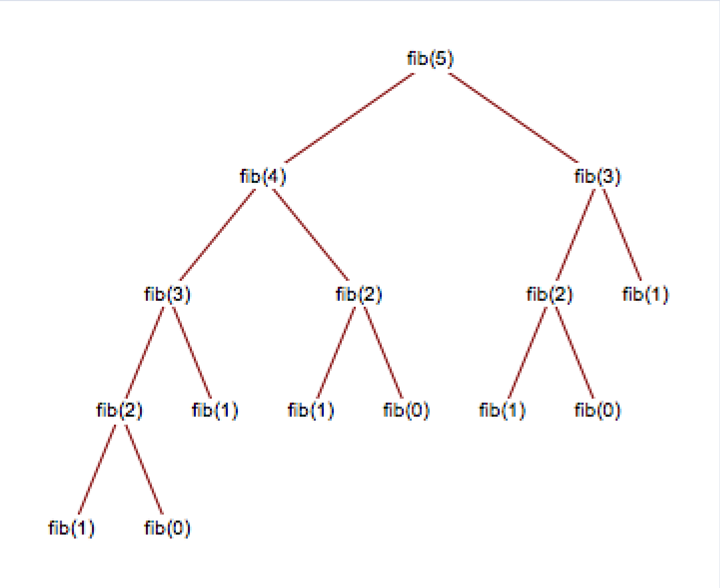
$= O(2 + n + 5n(\log(n))) = O(n(\log(n)))$

Recursividad

Muchos algoritmos recursivos suelen ser de complejidad O(k^n), siendo k la cantidad funciones recursivas dentro del return.

<mark>Ejemplo:</mark>

```
int fibonacci(int n){                              //(2^n)
    if(n<=1)                                       //(2^n)
        return 1;                                  //(2^n)
    return fibonacci(n-1)+fibonacci(n-1);          //(2^n)
}
```


fib(5)
fib(4)
fib(3)
fib(3)
fib(2)
fib(2)
fib(1)
fib(2)
fib(1)
fib(1)
fib(0)
fib(1)
fib(0)
fib(1)
fib(0)

Ejercicios en Clase

1

```
string entrada;
cin>>entrada;
int x=5;
if(entrada=="holi")
    cout<<x;
```

2

```
int n;
cin>>n;
for (int i=1; i<=n; i++)
    cout<<"hola";
```

1 Solucion

```
string entrada;      // O(1)
cin>>entrada;        // O(1)
int x=5;             // O(1)
if(entrada=="holi") // O(1)
    cout<<x;         // O(1)
```

2 solucion

```
int n;                         //O(1)
cin>>n;                        //O(1)
for (int i=1; i<=n; i++) //O(n)
    cout<<"hola";              //O(n)
```

3

```
int n, c=5;
cin>>n;
for (int i=1; i<=n; i+=c){
    for (int j=1; j<=n; j+=c){
        cout<<i+j;
    }
}
```

4

```
int n;
cin>>n;
n=2;
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        for(int k=0;k<n;k*=j){
            cout<<i*j*k;
        }
    }
}
```

3 solucion

```
int n, c=5;                        //O(1)
cin>>n;                            //O(1)
for (int i=1; i<=n; i+=c){         //O(n)
    for (int j=1; j<=n; j+=c){     //O(n^2)
        cout<<i+j;                 //O(n^2)
    }
}
```

4 solucion

```
int n;                             //O(1)
cin>>n;                            //O(1)
n=2;                               //O(1)
for(int i=0;i<n;i++){              //O(1)
    for(int j=0;j<n;j++){          //O(1)
        for(int k=0;k<n;k*=j){     //O(1)
            cout<<i*j*k;           //O(1)
        }
    }
}
```

5

```
int n,c=2;
cin>>n;
for(int i=0;i<n;i*=c){
    cout<<i;
}
```

6

```
int n,c=5,i=10;
cin>>n;
for(i=0;i<n;pow(i, c))
    cout<<i;
```

5 solucion

```
int n,c=2;                  //O(1)
cin>>n;                     //O(1)
for(int i=0;i<n;i*=c){ //O(log(n))
    cout<<i;                //O(log(n))
}
```

6 solucion

```
int n,c=5,i=10;         //O(1)
cin>>n;                 //O(1)
for(i=0;i<n;pow(i, c)) //O(log(log(n)))
    cout<<i;            //O(log(log(n)))
```

Referencias

https://yewtu.be/watch?v=CtpvpnYNNiE

https://yewtu.be/watch?v=HcDV5MGGrRE

https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/

https://yewtu.be/watch?v=MyAiCtuhiqQ

https://yewtu.be/watch?v=IZgOEC0NIbw

https://en.wikipedia.org/wiki/Big_O_notation