Órdenes de crecimiento

- **Logarítmico:** $Log(n)$

  Algoritmo: Búsqueda Binaria, básicamente la búsqueda en la lista ordenada se da haciendo particiones de la misma lista, calculando en cada iteración el centro de la lista y comparando si el valor en el centro es igual al valor buscado, si es mayor entonces a llamar al algoritmo pero con los valores mayores que el centro anteriormente calculado, lo mismo pasa si el valor buscado es menor, lo que se hace es agarrar ahora los valores menores al centro anterior.

  Análisis: Este algoritmo puede ser programado utilizando recursión, además que al ir partiendo la lista siempre en la mitad y que gracias que la lista está ordenada, al hacer las comparación con el centro se van eliminando comparaciones innecesarias lo que hace que al medir graficar el tiempo en función de la entrada se puede decir que tiene un comportamiento logarítmico o sea que crece lentamente en el eje y.

  Ayudas para determinar si es logarítmico:
  En Algoritmos con iteración o recursión.

- **Lineal:** n
  Algoritmo: Se recorre una lista y se duplica sus valores.

  Análisis:
  Por el propósito de este algoritmo se tiene que toda la lista, por eso es que tiene una complejidad de n ya que siempre se recorrerá todo el largo de la lista. Y esto se puede observar con claridad en el for que se utiliza para recorrer dicha lista.

  for(int i = 0; i < n; i++) ➞ **n**

  Ayudas para determinar si es lineal:

  En bucles simples cuando la complejidad de las operaciones internas es constante o en algunos algoritmos con recursión.

- **Lineal Logarítmico:** $n * Log(n)$

  for(int i = 0; i < n; i++) ➞ **n**

  for(int j = 1; j < n; j = j * 2) ➞ **log(n)**

  [Bloque de código básico] ⇒ No tiene ramificaciones ni loops

  Análisis:

  El primer bloque for se representa como n porque se ejecuta n veces, por lo cual tiene un comportamiento lineal.

  El segundo for también se ejecutaría n veces, sin embargo, al crecer de dos en dos (j = j * 2), se comporta como log(n).

  Para reconocer este tipo de algoritmos se pueden identificar loops anidados.

- **Cuadrático:** n^2

  arr = [ ] [ ]

  for(int i = 0; i < n; i++) ➞ **n**

  for(int j = 1; j < n; j ++) ➞ **n**

  print arr[i] [j]

  Análisis:

  Ambos bloques for se representan como n, se ejecuta n veces, por lo tanto se comportan como una función cuadrática

  Ayudas para determinar si es cuadrático:

  Aparece en bucles o recursiones doblemente anidados

- **Cúbico:** N^3

```
arr = [ ][ ] → n
for(int i = 0; i < n; i++)  → n

        for(int j = 1; j < n; j ++)  → n

                print arr[ i ] [ j ]
```

La asignación Ambos bloques for se representan como n, se ejecuta n veces, por lo tanto tienen un comportamiento cúbico.

Ayudas para determinar si es cúbico:

En bucles o recursiones triples

- **Exponencial:** 2^N

Algoritmo: Fibonacci

```
int fibonacci(int n){
   if (n>1){
     return fibonacci(n-1) + fibonacci(n-2);  //función recursiva  → 2^N
   }
   else if (n==1) {  // caso base
      return 1;
   }
   else if (n==0){  // caso base
      return 0;
   }
   else{ //error
      return -1;
   }
}
```

Análisis:

Se hacen dos llamadas recursivas en cada ejecución, al hacer estas llamadas para calcular fibonacci se cuenta como 2^N.

Ayudas para determinar si es exponencial:

Suele aparecer en subprogramas recursivos que contengan dos o más llamadas internas

- **Factorial:** $N!$

Algoritmo: Generar todas las permutaciones de una lista

Análisis:

```
def permutation(array, start, result)
   if (start == array.length) then
      result << array.dup
   end
   for i in start..array.length-1 do
      array[start], array[i] = array[i], array[start]
      permutation(array, start+1,result)
      array[start], array[i] = array[i], array[start]
   end
   result
end
```

Como se puede ver en la representación del algoritmo,dentro del “for” hay un llamada recursiva, esto es para que cada elemento en el array, sea intercambiado en la position, hay es donde se alcanza la complejidad de n factorial porque cada permutación es una operación y hay n! Operaciones.

Ayudas para determinar si es factorial:

Llamadas recursivas dentro de un ciclo.