Orden de menor a mayor complejidad:

1. log₂(n)

2. (log₂(n))²

3. √n

4. n / log₂(n)

5. n

6. n log₂(n)

7. n^1.5

8. n²

9. 2^n

10. n!

2.

|  |  |
| --- | --- |
| T(n) | Algortimo |
| T1(n) = 3n^2 + 50n | Insertion Sort / Bubble Sort (peor caso) |
| T2(n) = 8nlog2(n) + 200 | MergeSort / HeapSort |
| T3(n) = 0.2n^3 | Floyd–Warshall |
| T4(n) = 2^n | Backtracking / Fuerza bruta |

import java.util.\*;

public class Main {

public static void main(String[] args) {

// ---------- Comparación 1 ----------

System.out.println("Comparación T1 (O(n²)) vs T2 (O(n log n)):");

for (int n = 1; n <= 1000; n++) {

double T1 = 3 \* Math.pow(n, 2) + 50 \* n;

double T2 = 8 \* n \* (Math.log(n) / Math.log(2)) + 200;

if (T2 < T1) {

System.out.println("→ Umbral encontrado en n = " + n);

System.out.println("A partir de aquí Merge/Heapsort supera a Insertion/Bubble.\n");

break;

}

}

// ---------- Comparación 2 ----------

System.out.println("Comparación T3 (O(n³)) vs T4 (O(2ⁿ)):");

for (int n = 1; n <= 50; n++) {

double T3 = 0.2 \* Math.pow(n, 3);

double T4 = Math.pow(2, n);

if (T4 > T3) { // el exponencial empieza a dominar

System.out.println("→ Umbral encontrado en n = " + n);

System.out.println("A partir de aquí el algoritmo exponencial crece más rápido.\n");

break;

}

}

}

}

Isaac y los intervalos mágicos - Análisis

1. Entrada

• Un número entero N, que indica cuántos intervalos serán consultados.  
• Luego, N pares de enteros (a, b), donde a ≤ b, que representan los límites de cada intervalo.

Ejemplo de entrada:  
3  
1 10  
5 20  
30 50

2. Salida

• Para cada intervalo, un número entero que indique la cantidad de números primos en ese rango [a,b].  
• Se debe producir N líneas de salida, una por cada consulta.

Ejemplo de salida:  
4  
6  
5

3. Lógica y Análisis

El problema consiste en responder rápidamente cuántos primos existen en un rango. La dificultad está en que pueden ser muchos intervalos y con números grandes.

a) Enfoque ingenuo (ineficiente)

• Para cada intervalo (a,b), recorrer todos los números entre a y b y verificar si son primos.  
• Complejidad: O(N · (b-a)√m), demasiado lento si a,b son grandes.

b) Enfoque eficiente: Criba de Eratóstenes + Sumas prefix

1. Encontrar el máximo valor de b entre todos los intervalos.  
2. Usar una Criba de Eratóstenes hasta ese máximo → determinar qué números son primos.  
3. Construir un arreglo acumulado (prefix sum), donde:  
 P[i] = cantidad de primos ≤ i  
4. Para responder una consulta (a,b):  
 primos en [a,b] = P[b] - P[a-1]  
5. Complejidad:  
 • Preprocesamiento: O(M log log M), donde M = max(b).  
 • Cada consulta: O(1).