Ejercicio 1: Sea el siguiente algoritmo, que calcula el logaritmo discreto de $n \in N$ en base $b \geq 2$:

```
log: n x b → r
si n < b
   r ← 0
si no
   r ← log(n div b, b) + 1
```

a) Identifique los elementos que lo determinan como perteneciente al esquema de <<divide y venderás>>. ¿Con qué subesquema de éste lo identifica?

$t(n, b) = \lfloor log_b n \rfloor$

Se realiza una descomposición antes de la llamada recursiva (n div b).

Se lleva a cabo una resolución recursiva del problema. Caso base cuando n = b – 1.

Subesquema: simplificación o reducción. Se divide en problema en uno más pequeño. No se combinan las soluciones.

b) Analice mediante ecuaciones de recurrencia el número de operaciones aritméticas que realiza cuando n es potencia de b.

log: n x b $\rightarrow$r
si n < b
    r $\leftarrow$0
si no
    r $\leftarrow$log(n div b, b) + 1

2 operaciones aritméticas por cada llamada recursiva. Ninguna en el caso base.

Formalmente,

$t(n, b) = t(n)$ {durante la ejecución del algoritmo no cambia su valor}

$$t(n) \begin{cases} 0 & si, n < b \\ t(n \, div \, b) + 2 & si, n \geq b \end{cases}$$

$t(n) = t(n\,div\,b) + 2$, si n $\geq b \Rightarrow$ {Suponemos siempre divisible: $n = b^k$}

$t(b^k) = t(b^k\,div\,b) + 2$, si $b^k \geq b \Rightarrow$ {Simplificamos la división}

$t(b^k) = t(b^{k-1}) + 2$, si $b^k \geq b \Rightarrow$ {Simplificamos la condición}

$t(b^k) = t(b^{k-1}) + 2$, si $k \geq 1 \Rightarrow$ {Cambio de variable $T(k) = t(b^k)$}

$T(k) = T(k-1) + 2$, si k $\geq 1 \Rightarrow$ {Resolvemos la ecuación de recurrencia mediante sumatorios}

$T(k) = T(0) + \sum_{i=1}^{k} 2 \Rightarrow$ {Averigüemos cuanto vale $T(0)$}

$T(k) = t(b^k)$, luego $T(0) = t(b^0) = t(1)$,

$t(n) = 0\ si\ n < b$, n = 1 y b $\geq$ 2, luego $T(0) = t(b^0) = t(1) = 0$

$T(k) = 0 + \sum_{i=1}^{k} 2 = 0 + 2 \cdot k = 2 \cdot k \Rightarrow$ {Resolvemos el sumatorio}

$T(k) = 2 \cdot k \Rightarrow$ {deshacer el cambio de variable $T(k) = t(b^k)$}

$t(b^k) = 2 \cdot k \quad k \geq 1 \Rightarrow$ {Deshacer la hipótesis del enunciado}

si $n = b^k$ entonces $k = log_b n$.

Y respecto a la condición, si $k \geq 1 \Rightarrow log_b n \geq 1 \Rightarrow b^{log_b n} \geq b^1 \Rightarrow n \geq b$

$t(n) = 2 \cdot log_b n \quad n \geq b$

c) El número exacto de operaciones aritméticas se obtiene inmediatamente si se supone que el algoritmo es correcto. ¿Por qué?

El algoritmo realiza dos operaciones aritméticas por cada llamada recursiva.

El algoritmo se llama a si mismo $log_b n$ veces, incrementando el resultado en una unidad cada vez. Los incrementos, parten desde el valor 0.

El número de operaciones aritméticas es $2 \cdot log_b n$.

Ejercicio 2)

Los trozos son:

1. $[i, k_1]$
2. $[k_1 + 1, k_2]$
3. $[k_2 + 1, j]$

Teniendo esto en cuenta:

- $n = j - i + 1$
- $k_1 = i - 1 + n \text{ div } 3$
- $k_2 = k_1 + (j - k_1) \text{ div } 2$

Nota: $(j - k_1)$ no es más que una forma eficiente de escribir $(n - n \text{ div } 3)$.

n es divisible por 3 ➔ los tres trozos son iguales.

Si el resto es 1 (sobra un elemento) ➔ va en el tercer trozo, es decir, los dos primeros son iguales y más cortos que el tercero.

Si el resto es 2 (sobran dos elementos) ➔ uno va en el segundo trozo y otro en el tercero, es decir, el primero es más corto que los otros dos (que son iguales).

Análisis cuando n es potencia de tres.

Mejor caso: se entra sistemáticamente por la primera rama.

$$t(n) = t(n / 3) + 1$$

Peor caso: se entra siempre por la tercera rama.

$$t(n) = t(n / 3) + 2$$

a) Desarrolle esta idea escribiendo el algoritmo apropiado.

- Análogo a la búsqueda binaria.
- Fijar el umbral al menos en $n_0 = 2$.
- Reducir sucesivamente la búsqueda en un vector de tamaño $n > n_0$ a otra en un subvector cuyo tamaño sea aproximadamente de un tercio de n.

```
busqueda-ternaria: v x i x j → p
n ← j - i + 1
si n = 1
    si x ≤ v[i]
        p ← i
    si no
        p ← i + 1
si no si n = 2
    si x ≤ v[i]
        p ← i
    si no
        si x ≤ v[j]
            p ← j
        si no
            p ← j + 1
si no
    k1 ← i - 1 + n div 3
    k2 ← k1 + (j - k1) div 2
```

```
si x ≤v[k1]
    p ←busqueda_ternaria (v, x, i, k1)
si no
    si x ≤v[k2]
        busqueda_ternaria(v, x, k1+1, k2)
    si no
        busqueda_ternaria(v, x, k2+1, j)
```

b) Haga un análisis completo cuando n es potencia de tres.

Hipótesis simplificadora $n = 3^k$

Operación crítica: las comparaciones entre elementos del vector.

Analizaremos el mejor caso. (Cuando se entra por alguna primera rama)

Luego nuestra ecuación de recurrencia quedará como sigue:

$$t(n) = \begin{cases} 1 & n = 1 \\ t(\frac{n}{3}) + 1 & n \geq 3 \end{cases}$$

Nota: no consideramos el caso n = 2 puesto que un número 3^k nunca será igual a 2.

$t(n) = t(\frac{n}{3}) + 1 \quad n \geq 3 \Rightarrow$ {Aplicamos la hipótesis simplificadora}

$t(3^k) = t(\frac{3^k}{3}) + 1 \quad 3^k \geq 3 \Rightarrow$ {Simplificamos}

$t(3^k) = t(3^{k-1}) + 1 \quad 3^k \geq 3 \Rightarrow$ {Cambio de variable, $t(3^k) = T(k)$}

$T(k) = T(k-1) + 1 \quad k \geq 1 \Rightarrow$ {Transformamos en un sumatorio y añadimos el caso base}

$T(k) = T(0) + \sum_{i=1}^{k} 1$ {Averigüemos el valor de $T(0)$}

$T(0) = t(3^0) = t(1) = 1$

$T(k) = 1 + k$ {Deshacemos el cambio de variable}

$t(3^k) = 1 + k$ {Deshacemos la hipótesis simplificadora}

Si $n = 3^k$ entonces $k = log_3 n$

$t(3^{log_3 n}) = 1 + log_3 n$

$t(n) = 1 + log_3 n \in O(log_3 n)$

Analizaremos el peor caso. (Cuando se entra por alguna segunda rama)

Luego nuestra ecuación de recurrencia quedará como sigue:

$$t(n) = \begin{cases} 1 & n = 1 \\ t(\frac{n}{3}) + 2 & n \geq 3 \end{cases}$$

Nota: no consideramos el caso n = 2 puesto que un número 3^k nunca será igual a 2.

$t(n) = t(\frac{n}{3}) + 2 \quad n \geq 3 \Rightarrow$ {Aplicamos la hipótesis simplificadora}

$t(3^k) = t(\frac{3^k}{3}) + 2 \quad 3^k \geq 3 \Rightarrow$ {Simplificamos}

$t(3^k) = t(3^{k-1}) + 2 \quad 3^k \geq 3 \Rightarrow$ {Cambio de variable, $t(3^k) = T(k)$}

$T(k) = T(k-1) + 2 \quad k \geq 1 \Rightarrow$ {Transformamos en un sumatorio y añadimos el caso base}

$T(k) = T(0) + \sum_{i=1}^{k} 2$ {Averigüemos el valor de $T(0)$ }

$T(0) = t(3^0) = t(1) = 1$

$T(k) = 1 + 2k$ {Deshacemos el cambio de variable}

$t(3^k) = 1 + 2k$ {Deshacemos la hipótesis simplificadora}

Si $n = 3^k$ entonces $k = log_3 n$

$t(3^{log_3 n}) = 1 + 2\, log_3 n$

$t(n) = 1 + 2\, log_3 n \in O(log_3 n)$.

c) Suponga que los resultados del análisis son asintóticamentes ciertos para todo n. Compárelos con los que se obtienen para la búsqueda binaria.

Sabemos que para la búsqueda binaria:

- $t_{min}(n) = log_2 n \quad n \geq 1$
- $t_{max}(n) = log_2 n \quad n \geq 1$

Sabemos que para la búsqueda ternaria:

- $t_{min}(n) = 1 + log_3 n \quad n \geq 1$
- $t_{max}(n) = 1 + 2 \cdot log_3 n \quad n \geq 1$

$$2 \cdot log_2(n+1) - 2 < 1 + log_3$$

{El peor caso de búsqueda binaria es mejor que el peor caso de búsqueda ternaria, igualmente para los mejores casos.}

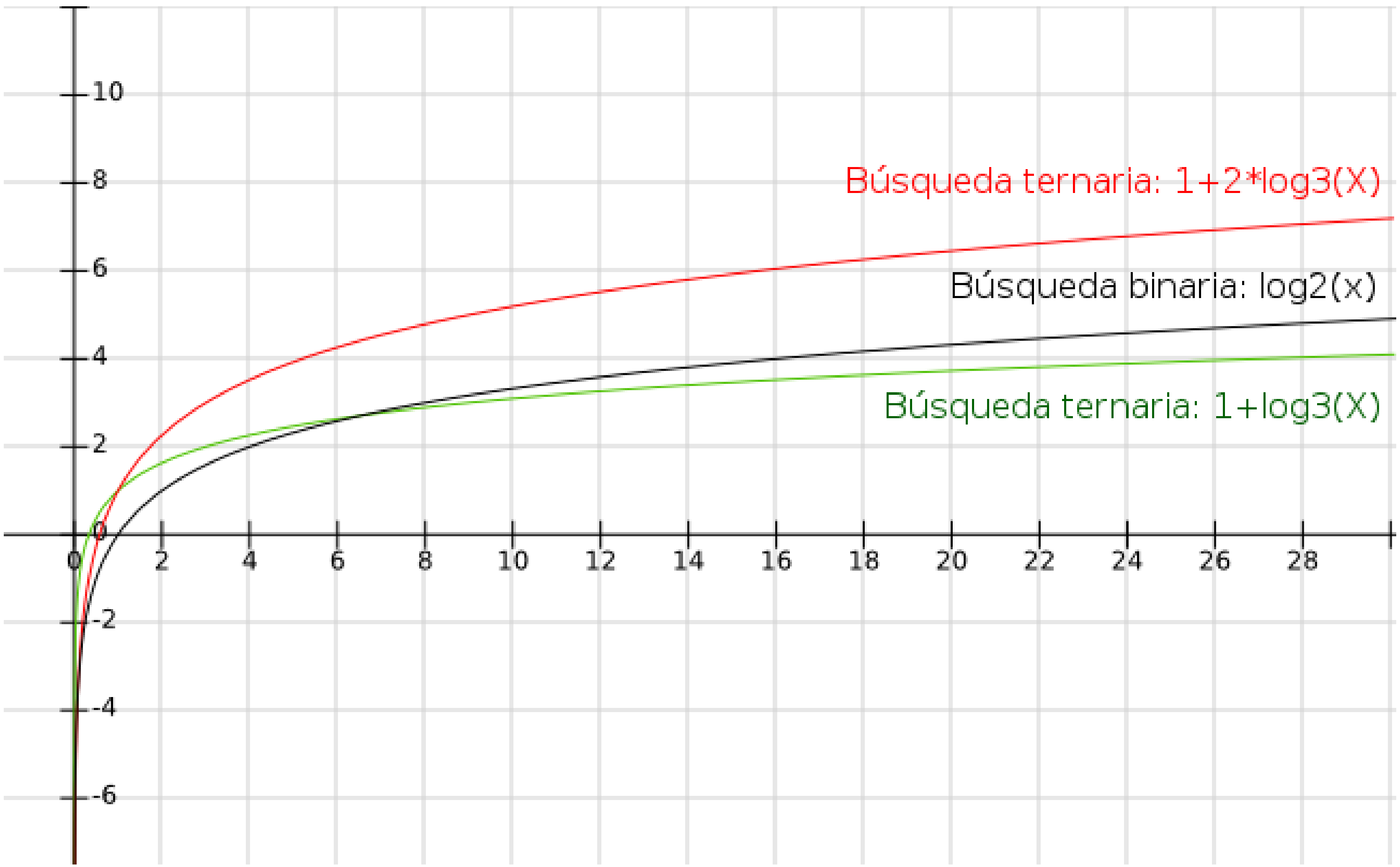
Conclusión, es mejor la búsqueda binaria que la búsqueda ternaria.

Una comparación muy simple es la siguiente:

- El tiempo en el peor caso de la búsqueda binaria es, aproximadamente, $\log_2 n$ comparaciones.
- El de la ternaria, es, aproximadamente, $2\log_3 n$ comparaciones.

$$2\log_3 n = 2(\log_2 n / \log_2 3) \approx 1{,}26\log_2 n.$$

Por lo tanto, en el peor caso, la búsqueda ternaria realiza un 26% más de comparaciones que la binaria (además, realiza más operaciones de otros tipos).


Búsqueda ternaria: 1+2*log3(X)
Búsqueda binaria: log2(x)
Búsqueda ternaria: 1+log3(X)
10
8
6
4
2
0
-2
-4
-6
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28