En este caso no es necesario encontrar el intervalo donde se encuentra la solución óptima por "tanteo" (es decir, acotar la solución óptima entre potencias de dos). Efectivamente, si el poder de ataque iguala a la suma *s* de los poderes de todos los enemigos, el caballero será capaz de vencer en un único torneo. Por tanto, el poder mínimo necesario estará en el intervalo [0,*s*].

Para encontrar el poder mínimo necesario puede formularse una generalización, que recibe como entrada los extremos *inf* y *sup* del intervalo donde está dicho poder, y aplica la siguiente estrategia DV:

- Calcular el punto medio *m* = (*inf*+*sup*)/2
- Si *m* es suficiente para vencer a todos los enemigos dentro del número de torneos permitido:
    - Si *m*-1 no es suficiente, *m* será el poder mínimo necesario buscado
    - En otro caso, habrá que buscar el poder mínimo en [*inf, m-1*]
- En otro caso (*m* no era suficiente para vencer), habrá que buscar el poder mínimo en [*m+1*,*sup*]

La comprobación de si un cierto poder *p* es suficiente para vencer a los enemigos realizando, como máximo, c torneos, puede realizarse mediante una única iteración sobre el vector de enemigos, luchando en cada torneo con el máximo de enemigos posibles, en el orden en el que estos aparecen en el vector. Si es posible llegar al final, *p* será suficiente. En otro caso, no lo será.

Un aspecto delicado de este problema es el cálculo de la complejidad. Para calcular la complejidad de la estrategia DV debe tenerse en cuenta de que la función de tiempo **depende de dos variables diferentes**: (i) el número *n* de enemigos, y (ii) el tamaño (sup-inf)+1 del intervalo en el que se encuentra el poder mínimo necesario. Con ello, la función de tiempo será una función de dos variables T(n,u), donde el primer argumento representa el número de enemigos, y el segundo la longitud del intervalo, y que obedece a la siguiente recurrencia:

$T(n,1) = w(n)$

$T(n,u) = w(n) + T(n,u/2)$

donde w(.) representa la complejidad en el peor de los casos de comprobar si un determinado poder *p*, y su predecesor *p*-1, son suficientes o no para vencer a los enemigos.

Esta recurrencia **no** puede resolverse directamente con los patrones, pero sí mediante el método de expansión de recurrencias:

- Si expandimos la recurrencia llegamos a $T(n,u) = w(n) + T(n,u/2) = w(n) + w(n) + T(n,u/4) = \ldots = k\, w(n) + T(n,u/2^k)$.
- Suponiendo, asimismo, que *u* es una potencia de 2, es decir, $u = 2^v$, y expandiendo *v*-1 veces, llegamos a $T(n,u) = (v-1)w(n) + T(n,u/2^{v-1})$.
- Aplicando la recurrencia una última vez, llegamos a $T(n,u) = v\, w(n) + T(n,1) = v\, w(n) + w(n)$. Pero, como $v= \log_2(u)$, $T(n,u) = w(n) \log_2(u) + w(n)$.

Ahora bien, dado que la búsqueda inicial se aplica al intervalo [0,s], la complejidad de la fase de búsqueda de la solución vendrá dada por $w(n) \log_2(s) + w(n) \leq w(s) \log_2(s) + w(s)$, ya que el número de enemigos *n* será siempre menor o igual que la suma *s* de los poderes de dichos enemigos. Por tanto, como $w(s) \log_2(s) + w(s)$ está en O(s log s), la complejidad de la búsqueda será O(s log s).

Para determinar la complejidad final habrá que considerar la complejidad de la fase de acotación. Como dicha fase consiste, básicamente, en sumar los elementos del vector de poderes, la complejidad de la fase será O(n). Y como $n \leq s$, será también O(s).

Por tanto, la complejidad predominante será la de la fase de búsqueda, por lo que la complejidad final del algoritmo será O(s log s), con s la suma de los poderes de los enemigos.