Ejercicio 2

Este algoritmo NO presenta caso mejor y peor, ya que el argumento que recibe la función es un entero y el bloque if siempre se ejecutará (o no) dependiendo del valor del entero, no hay forma de que para un mismo valor de n se produzcan dos ejecuciones con coste distinto.
Por tanto:

$$C_i(n) = T(n) = \begin{cases} 1, n \leq 1 \\ (n-1) + 4T(n/2), n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= (n-1) + 4T(n/2) \\ &= (n-1) + (2n-4) + 16T(n/4) \\ &= (n-1) + (2n-4) + (4n-16) + 64T(n/8) \\ &\dots \end{aligned}$$

De manera general, expresamos $T(n)$ de la siguiente manera:

$$T(n) = \sum_{i=0}^{k-1} 2^i n - \sum_{i=0}^{k-1} 4^i + 4^k T(n/2^k) \forall k : 1 \leq k \leq log_2 n$$

De donde deducimos que:

$$n/2^k = 1 \rightarrow n = 2^k \rightarrow k = log_2 n$$

Por tanto, si sustituimos el valor de k en la expresión anterior, obtenemos que el coste del algoritmo resultará:

$$T(n) = \sum_{i=0}^{log_2 n-1} 2^i n - \sum_{i=0}^{log_2 n-1} 4^i + 4^{log_2 n} T(n/2^{log_2 n})$$

$$T(n) = \sum_{i=0}^{\log_2 n-1} 2^i n - \sum_{i=0}^{\log_2 n-1} 4^i + 4^{\log_2 n}$$

$$T(n) = n(2^{log_2 n} - 1) - \frac{4^{log_2 n} - 1}{3} + 4^{log_2 n}$$

$$T(n) = n(n-1) - \frac{n^2 - 1}{3} + n^2$$

$$T(n) = 2n^2 - n - \frac{n^2 - 1}{3}$$

$$T(n) = 2n^2 - n - \frac{n^2}{3} - \frac{1}{3}$$

$$T(n) = \frac{6n^2}{3} - \frac{n^2}{3} - n - \frac{1}{3}$$

$$T(n) = \frac{5}{3}n^2 - n - \frac{1}{3} \in \Theta(n^2)$$

Ejercicio 3

Este algoritmo analiza si la cadena que recibe la función es un palíndromo a base de comparar la primera letra con la última de forma recursiva hasta llegar al centro. Por tanto, presentará caso mejor y peor dependiendo de cuántas letras tenga que recorrer hasta terminar, para dos palabras de misma longitud puede tener un coste diferente.

Complejidad temporal del caso mejor

Este caso se dará cuando la función termina lo más rápido posible, es decir, cuando encuentra una diferencia en los primeros caracteres y retorna `false` sin seguir llamando a la recursión.

$$C_i(n) = T(n) = \begin{cases} 1, & \text{si } pal[pri] \neq pal[ult] \text{ en la primera comparación} \end{cases}$$

Esto significa que la complejidad en el caso mejor es:

$$T(n) \in \Omega(1)$$

Complejidad temporal del caso peor

El caso peor ocurre cuando la función revisa todas las letras hasta el centro, lo que sucede si la cadena es efectivamente un palíndromo. En este caso, la recursión continúa hasta que los índices `pri` y `ult` se crucen o sean iguales.
Asumimos el coste de la comparación entre dos caracteres como 1 puesto que siempre va a tener lugar. Además, en cada llamada recursiva quitamos dos caracteres del string, por tanto tendremos $n-2$ caracteres. De modo que podemos expresar esto como:

$$C_i(n) = T(n) = \begin{cases} 1 + T(n-2), & \text{si } pal[pri] = pal[ult] \text{ siempre (palíndromo)} \end{cases}$$

$$\begin{aligned} T(n) &= 1 + T(n-2) \\ &= 1 + T(n-4) \\ &= 1 + T(n-6) \end{aligned}$$

...

Podemos expresar $T(n)$ de la siguiente manera después de k pasos:

$$T(n) = k + T(n - 2k)$$

Podemos despejar el valor de k sabiendo que la recursión acabará cuando n sea menor o igual que cero, siendo n la diferencia entre `pri` y `ult` (la diferencia solo será positiva cuando no se haya llegado al centro de la palabra). Por tanto:

$$n - 2k \leq 0 \rightarrow n - 2k = 0 \rightarrow 2k = n \rightarrow k = \frac{n}{2}$$

Por tanto, si sustituimos el valor de k en la expresión anterior, obtenemos que el coste del algoritmo resultará:

$$T(n) = \frac{n}{2} + T(n - 2\frac{n}{2}) = \frac{n}{2} + T(0) = \frac{n}{2} \in O(n)$$