The running time of an algorithm can be described by its best, average and worst case. Talking about asymptotic notation, we use three different notations: Big-Theta, Big-O and Big-Omega notations.

If we assume the running time of an algorithm is f(n) in which n is the large enough input sizes, Big-O notation is related to the worst case of running time or an upper bound for the growth rate of a given function f(n). For example, if g(n) be an arbitrary time complexity, therefore in Big-O notation we can say that: f(n)<k\*g(n). The algorithm runtime ( f(n) ) will not grow faster than g(n).

Big-Omega notation provides an asymptotic notation for the best case. It means that, it provides a lower bound for the growth rate of the algorithm runtime: f(n)>k\*g(n). Here, the algorithm runtime will not grow slower than g(n). the runtime is always more than g(n).

Big-Theta notation or asymptotically tight bound gives us a more precise knowledge about the growth rate of algorithm (f(n)). It will produce an upper and lower bound for f(n): k1\*g(n)< f(n) < k2\*g(n).

What is still unclear for me in these notations is that why we would need to know about Big-O and Big-Omega, if the Big-Theta is providing us with a precise running time?