General Idea

Most programming contests require contestants to solve each of the problems within a given time limit. It is extremely important to estimate the running time of your solution, and know if it belongs to an acceptable time complexity class before sumitting it to the judge. This can be achieved by doing some simple math.

Rule of Thumb

In practice, we usually assume the machine can do at most 10^8 operations per second.
Note this number is an underestimate. However, it somehow compensates our inability to count the exact number of instructions executed by the CPU and works well in practice.

Assume the time limit of a problem is 1 second. Ideally, given a solution with time complexity $O(f(n))$, the solution will be accepted only if $f(n)$ is less than 10^8 for the biggest possible input size n.

Experienced contestants even use the input size n to guess what algorithm the problem write is expecting, which usually leads to quite good result (and sometimes, terribly bad ones).

Common Complexity Classes

Here is a list of common input sizes and the corresponding complexity classes.

time complexity	**maximum n**	**examples**	**notes**
really small	does not matter		super easy problem, or the problem writer does not care about performance
around 10	$O(n!)$ or does not matter	permutation	
around $20 - 24$	$O(2^n)$		
10^2	$O(n^3)$		uncommon, even rarely $O(n^3 \log n)$
10^3	$O(n^2)$		very rarely $O(n^2 \log n)$
10^4	case by case analysis required		very rare, $O(n^2)$ usually does not work
$10^5 - 10^6$	$O(n \log n)$	quick sort	
10^7	$O(n)$	n hashtable quries	$O(n \log n)$ may not work
$10^8, 10^9, 2 \cdot 10^9, 10^{18}$	$O(1)$ or $O(\log n)$	euclidean algorithm (gcd)	read: maximum integer value; usually math problems