The [**growth rate**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-growth-rate) for an algorithm is the rate at which the cost of the algorithm grows as the size of its input grows.

A growth rate of cncn (for cc any positive constant) is often referred to as a [**linear growth rate**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-linear-growth-rate) or running time. This means that as the value of nn grows, the running time of the algorithm grows in the same proportion.

An algorithm whose running-time equation has a highest-order term containing a factor of n2n2 is said to have a [**quadratic growth rate**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-quadratic-growth-rate).

The line labeled 2n2n represents an [**exponential growth rate**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-exponential-growth-rate). This name comes from the fact that nn appears in the exponent.

the difference between an algorithm whose running time has cost T(n)=10nT(n)=10n and another with cost T(n)=2n2T(n)=2n2 becomes tremendous as nn grows. For n>5n>5, the algorithm with running time T(n)=2n2T(n)=2n2 is already much slower. This is despite the fact that 10n10n has a greater constant factor than 2n22n2. Comparing the two curves marked 20n20n and 2n22n2 shows that changing the constant factor for one of the equations only shifts the point at which the two curves cross. For n>10n>10, the algorithm with cost T(n)=2n2T(n)=2n2 is slower than the algorithm with cost T(n)=20nT(n)=20n.

For constants a,b>1,naa,b>1,na grows faster than either logbnlogb⁡n or lognblog⁡nb. Finally, algorithms with cost T(n)=2nT(n)=2n or T(n)=n!T(n)=n! are prohibitively expensive for even modest values of nn. Note that for constants a,b≥1,ana,b≥1,an grows faster than nbnb.

Because the phrase "has an upper bound to its growth rate of f(n)f(n)" is long and often used when discussing algorithms, we adopt a special notation, called [**big-Oh notation**](https://traky.cs.hut.fi/Books/CSE-A1141/html/Glossary.html#term-big-oh-notation). If the upper bound for an algorithm's growth rate (for, say, the worst case) is (f(n)), then we would write that this algorithm is "in the set O(f(n))O(f(n)) in the worst case" (or just "in O(f(n))O(f(n)) in the worst case"). For example, if n2n2 grows as fast as T(n)T(n) (the running time of our algorithm) for the worst-case input, we would say the algorithm is "in O(n2)O(n2) in the worst case".

The following is a precise definition for an upper bound. T(n) represents the true running time of the algorithm. f(n)f(n) is some expression for the upper bound.

For T(n)T(n) a non-negatively valued function, T(n)is in set O(f(n))O(f(n)) if there exist two positive constants cc and n0n0 such that T(n)≤cf(n)T(n)≤cf(n) for all n>n0n>n0.

the definition says that for all inputs of the type in question (such as the worst case for all inputs of size nn) that are large enough (i.e., n>n0n>n0), the algorithm always executes in less than or equal to cf(n)cf(n) steps for some constant cc.