Time Complexity: is a measure of time for execution of an algorithm, commonly denoted by '***big-o notation***'.

Constant TC: O(1).

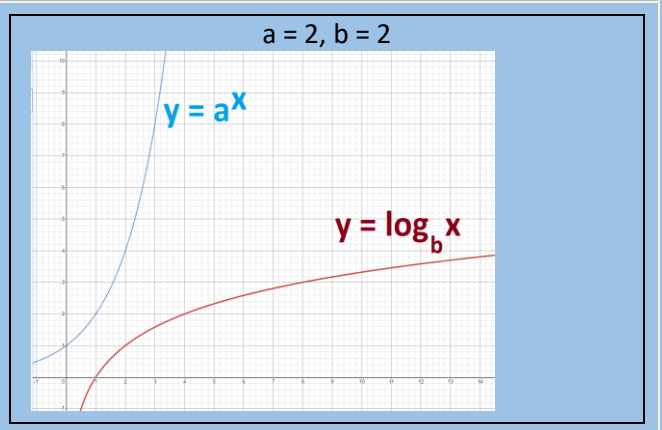
An operation/algorithm is independent of the size of the input and can be carried out in linear time. Example, **push**, **pop**, **std::vector<T>::at(index)**,

```
const int const_size = 100;
int index = 10;
int item = array[ index ];
for( int i = 0 ; i < const_size ; ++i ) {
  std::cout << array[ i ];
}
```

Logarithmic TC: O(log n).

Know,
[$\log_a x = y$] means [$a^y = x$]

Example, binary-search,
In C++ STL, search on associative containers (**set**, **map**, **multimap**, **multiset**) is guaranteed to be **O(log n)**.



All Complexities in 1 Diagram

X axis: Number of elements ----- **Y axis:** Time taken

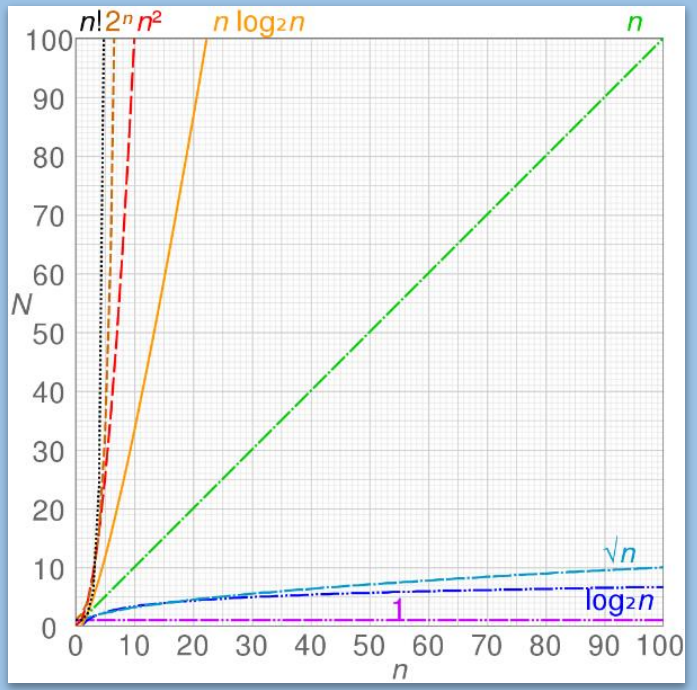


Image Source: *Wikipedia*

Linear TC: O(n)

Running time increases at most linearly with the size of the input. Examples, Sum of all elements in 1D array, Linear search, Compare pair of strings.

```
for( const auto & item : container ) {
  std::cout << item;
}
```

Linearithmic TC: O (n log n)

Example, Merge sort, Quick sort

Quadratic TC: O (n^2)

These algorithms' performance is directly proportional to squared size of input data. Example, Bubble sort,

```
for( const auto & a : container ) {
  for( const auto & b : container ) {
    std::cout << a + b;
  }
}
```

Polynomial Array TC: O (n^k)

- **Cubic TC: O (n^3)** Example, naïve multiplication of two **nxn** matrices

Exponential TC: O (2^n)

Examples, Finding all the subsets, Traveling Salesman problem

By, Prashant SG