Big-O gives you the upper-bound on the function's growth given the arguments.

# of statements
Assume most build-in functions run in constant time.
sum()

* The constant doesn't matter
n, 5n?
n^2, 2n^2
10^-9 n^2 vs. 10^9 n

* Only the highest degree term matters.
5000n^3 + 20 n^2 + n
500logn + n

O(1) < O(log[n]) < O(n) < O(n*log[n]) < O(k^n) < O(n!)
prove base doesn't matter for base.

best, worst and average case analysis.
find(), all_pair(), binary_search

# multiple input variables
O(mn), O(E + VlogV)

Answers:
O(n), O(1), O(1), O(n^2), O(n), O(n^2logn), O(logn)

```
int choose(int k, int n)
{
    if(k == 0 || k == n) return 1;
    return choose(k-1, n-1) + choose(k, n-1);
}

void solveHanoi(int n, int src, int dest, int buf)
{
    if (n == 1) {
        cout << "Move disk " << n << " from tower " << src << " to "
<< dest << endl;
    } else{
        solveHanoi(n-1, src, buf, dest);
        cout << "Move disk " << n << " from tower " << src << " to "
<< dest << endl;
        solveHanoi(n-1, buf, dest, src);
    }

}
```