Units for Measuring Running Time

- We'd like to use a measure that does not depend on extraneous factors such as the speed of a particular computer, dependence on the quality of a program implementing the algorithm and of the compiler used, and the difficulty of clocking the actual running time of the program.

- We usually focus on basic operations that the algorithm executes, and compute the number of times the basic operations are executed.

Worst-, Best-, and Average-Case Complexity

- The worst-case complexity of an algorithm is its complexity for the worst-case input of size n, which is an input of size n for which the algorithm runs the longest among all possible inputs of that size.
  - What type of bound does the worst-case analysis provide on the running time?

- The best-case complexity of an algorithm is its complexity for the best-case input of size n, which is an input of size n for which the algorithm runs the fastest among all possible inputs of that size.

- Neither of these two types of analyses provide the necessary information about an algorithm's behavior on a "typical" or "random" input. This is the information that the average-case complexity seeks to provide.

Worst-case Complexity: Example 2

Algorithm 2: MatrixMultiplication.

Input: Two matrices $A[0 \dots n-1, 0 \dots k-1]$ and $B[0 \dots k-1, 0 \dots m-1]$.
Output: Matrix $C = AB$.

```
1 for i = 0 to n − 1 do
2     for j = 0 to m − 1 do
3         C[i, j] ← 0;
4         for l = 0 to k − 1 do
5             C[i, j] ← C[i, j] + A[i, l] · B[l, j];
6 return C
```

Asymptotic Analysis of Algorithms

Algorithm 3: LinearSearch.

Input: An array $A[0 \ldots n-1]$ of integers, and an integer x.
Output: The index of the first element of A that matches x, or -1 if there are no matching elements.

```
1 i ← 0;
2 while i < n and A[i] ≠ x do
3     i ← i + 1;
4 if i ≥ n then
5     i ← −1;
6 return i
```


Algorithm 4: MatrixMultiplication.

Input: Two matrices $A[0 \ldots n-1, 0 \ldots k-1]$ and $B[0 \ldots k-1, 0 \ldots m-1]$.
Output: Matrix $C = AB$.

```
1 for i ← 0 to n − 1 do
2     for j ← 0 to m − 1 do
3         C[i, j] ← 0;
4         for l ← 0 to k − 1 do
5             C[i, j] ← C[i, j] + A[i, l] · B[l, j];
6 return C
```

Algorithm 1: IsBipartite.

Input: Undirected graph $g = (V, E)$.
Output: *True* if g is bipartite, and *False* otherwise.

```
1  foreach Non-empty subset V1 ⊂ V do
2      V2 ← V \ V1;
3      bipartite ← True;
4      foreach Edge {u, v} ∈ E do
5          if {u, v} ⊆ V1 or {u, v} ⊆ V2 then
6              bipartite ← False;
7              Break;
8      if bipartite = True then
9          return True;
10 return False;
```