22C:21 Lecture Notes
Run-time analysis

Aug 24, 2005

The goal of run-time analysis is to obtain a "pen and paper" estimate of how efficient an algorithm or a program or a data structure is.

Suppose A and B are two different programs that sort a given sequence of integers. You run A on a machine on a certain input and it completes in 2 seconds. You friend runs B on a different machine, on some input of her choice, and comes back reporting that B completed in 0.001 seconds. Can you conclude that B is much more efficient than A?

No. Because the running time of programs typically depends on the size of the input and A and B may have been run on inputs of different sizes. Furthermore, the machines on which these programs are run may have significantly different speeds.

Of the two factors mentioned above: input size and machine speed, run-time analysis focuses on the first and ignores the second. More precisely, the goal of run-time analysis is to obtain a machine independent estimate of the running time of an algorithm or a program as a function of the size of the input.

Example 1.

```
for(i = 0; i < n; i++)
      sum = sum + i;
```

Here n is the input. The above code can be expanded to

```
1.  i = 0
2.  if i >= n then goto line 6
3.  sum = sum + i
4.  i++
5.  goto line 2
6.
```

Assuming that on some hypothetical machine, line i takes c_i units of time, then the total running time of the above code equals

$$c_1 + c_2(n+1) + c_3 n + c_4 n + c_5 n = n(c_2 + c_3 + c_4 + c_5) + (c_1 + c_2).$$

Note that this has the form $An + B$, where A and B are constants (i.e.,independent of n). Such a function is called a linear function. We say that this code fragment has linear running time. The constants A and B are machine dependent and we ignore them. Instead we focus on the fact that the running time grows linearly with respect to n.

Example 2.

```
for(i = 0; i < n; i++)
{
      sum = sum + i;
      prod = prod*(i+1);
}
```

Again, here n is the input. The statements inside the for-loop still take a constant amount of time. Therefore, as in the previous example, this code fragment also has linear running time.

Example 3.

```
for(i = 0; i < n; i++)
      for(j = 0; j < n; j++)
            sum = sum + i * j;
```

This code can be expanded as

```
1.  i = 0
2.  if (i >= n) goto 6
3.  INNER LOOP
4.  i++
5.  goto 2
6.
```

We know that the INNER LOOP takes linear time, i.e., $An + B$ for some constants A and B. Assuming that the each line i other than line 3, executes in time c_i, we get that the total running time is:

$$c_1 + c_2(n+1) + (An+B)(n) + c_4 n + c_5 n = An^2 + (c_2 + B + c_4 + c_5)n + (c_1 + c_2).$$

This has the form $Xn^2 + Yn + Z$, where X, Y, and Z are all machine-dependent constants. This is a *quadratic function* in n and the code fragment is said to run in *quadratic time*. The constants X, Y, and Z are not important, what is important is the fact that the running time of the code fragment grows quadratically with respect to n.
