July 15, 2010

Analysis of Algorithms

We can measure the speed of an algorithm internally using some Java System calls.

Here are the steps to getting the run time of an algorithm in milliseconds:

1. Record the current start time of the system clock.
2. Let the software run and complete.
3. Record the current end time of the system clock.
4. Subtract the start time from the end time.
5. Print this value.

```
long start = System.currentTimeMillis();

// Some operation

System.out.println("Time: "+(System.currentTimeMillis()-start)+" millisec.");
```

System.currentTimeMillis() returns the current system time in milliseconds since midnight, January 1, 1970. In computer terms, "midnight, January 1, 1970" is known as the Epoch of computer time. We wish to study the execution time of an algorithm in terms of its number of operations. We do this in what's called "O-notation." O-notation represents the worse case number of primitive operations that an algorithm must perform.

Primitive operations include the following:

- Assignment
- Calling methods
- Basic arithmetic
- Comparing two numbers
- Calling an array index
- Dereferencing objects.
- Returning from a method

7 Basic Growth Functions

Constant Time - O(1)

The fastest of all algorithms are constant time algorithms. They work in the same number of steps each time and have no loops.

The mathematician Carl Gauss has a famous story about his childhood. When he was very young, his teacher assigned him the math problem of adding up all the numbers from 1 to 100.

```
int sum(int x) {
	int sum = 0;
	for (int i = 1; i <= 100; i++)
		sum += i;
	return sum;
}
```

This works by looping an addition/assignment statement x times. If we increase the x value, the algorithm takes more time. Gauss discovered that this could be done instantly:

```
int sum(int x) {
	return (x * (x+1))/2;
}
```

This has no loops, yet it returns the same value. If we increase the value of x, it still works with 1 addition, 1 multiplication, and 1 division.

Logarithmic Time - O(Log₂ N)

On the game show, "The Price is Right," Bob Barker played a game called, "The Clock Game." In this game, Bob told the contestant that the value of "the item up for bid" was between a high and low value. The user then had to guess the value of the item within 30 seconds. After each guess, Bob would say "higher" if the guess was too low or "lower" if the guess was too high. It is the only game on "The Price is Right" where it is possible to win every time without knowing anything about the product as long as the contestant follows a logarithmic time algorithm.

```
low <- Bob's Low Bound
high <- Bob's High Bound
secret <- Bob's Secret Value
while (clock is ticking)
      guess <- (high + low) / 2
      if (secret < guess)
            Bob Says "LOWER!"
            high <- guess + 1
      end if
      if (secret > guess)
            Bob Says "HIGHER!"
            low <- guess + 1
      end if
      if (secret > guess)
            Bob Says "RIGHT!"
            break
      end if
end while
```

This algorithm has been proven to never take more than "Log2(high-low)" guesses.

Linear Time - O(N)

Any algorithm which has to index through an array is usually an O(N) array.

<verbatim> int sum(int[] array) {

```
  int sum = 0;
  for (int i = 0; i < array.length; i++)
        sum += array[i];
  return sum;
```

} </code>

Linear-Logarithmic Time - O(N*Log₂ N)

Any algorithm that combines a linear time algorithm with a logarithmic time algorithm is considered a Linear-Logarithmic Time algorithm. We haven't covered any of this class of algorithms just yet, but there are two that need mentioning: quicksort and mergesort. These are generally considered the fastest of the sorting algorithms.

Squared Time - O(N²)

Any time we have a for loop within a for loop, it is considered an O(N2) time algorithm, typically when each element is compared to every other element. There are several examples of O(N2) time algorithms, such as simple sorting techniques like the Bubble Sort and Matrix Addition.

Cubed Time - O(N³)

Any time we have a for loop within a for loop within a for loop, it is considered an O(N3) time algorithm. The most common task are for video processing, 3D Matrices, and Matrix Multiplication.