LECTURE 4, JAN 25, 2011, COMPLEXITY & GROWTH OF FUNCTION

An *algorithm* is a finite set of precise instructions for performing a computation or for solving a problem. Algorithms have the following properties:

(1) Input & output
(2) Definitess
(3) Correctness
(4) Finiteness
(5) Effectiveness
(6) Generality

We study those properties in details through several algorithms.

Example: *linear search*

```
1  i ← 1
2  while i ≤ n & a_i ≠ x
3      i ← i + 1
4  if  i ≤ n
5      return i
6  else
7      return 0
```

Question: how do we evaluate the effectiveness of the algorithm?

We study the computational complexity of the algorithm, including *time complexity* and *space complexity*. We focus on the former in this course. Time complexity of an algorithm can be expressed in terms of the number of operations used by the algorithm when the input has a particular size. We choose the number of operations instead of actual computer time because of the difference in time needed for different computers to perform basic operations.

For the linear search, we consider the basic operations to be the number of comparisons made. We consider the following two problems,

(1) What is the *worst-case* complexity?
At each step of the loop in the algorithm, two comparisons are performed– one to see whether the end of the list has been reached ($i \leq n$) and one to compare the element x with a term of the list ($x \neq a_i$). At the end of the loop, there is an additional comparison ($i \leq n$). Consequently, if $x = a_i$, $2i+1$ comparisons are used. The worst case happens when the element is not in the list. In this case, $2n$ comparisons are used to determine that x is not a_i, for $i = 1, 2, \ldots, n$. Then, one comparison $i \leq n$ is used to exit the loop, and one more comparison is made to exit the algorithm ($i \leq n$). Totally, it is $2n + 2$.

(2) On *average*, how many comparisons are used?

$$\frac{3+5+\cdots+(2n+1)+1}{n} = n+2+\frac{1}{n}$$

We see that average complexity is much more complicated than worst-case complexity. We also see that in $n+2+1/n$, the term n really matters. We next introduce the growth of the functions.

(1) *Big-O notation*: Let f and g be functions, we say $f(x)$ is $O(g(x))$ if there are constants C and k such that

$$|f(x)| \leq C|g(x)|, \quad x > k.$$

We can see that the average complexity of linear search if $O(n)$ as

$$|n+2+\frac{1}{n}| \leq 2n, \quad n > 3.$$

(2) *Big-Ω notation*: We say $f(x)$ is $\Omega(g(x))$ if there are constants C and k such that

$$|f(x)| \geq C|g(x)|, \quad x > k.$$

Clearly, the average complexity of linear search if $\Omega(n)$.

(3) *Big-Θ notation*: We say $f(x)$ is $\Theta(g(x))$ if

$$f(x) = O(g(x)), \quad f(x) = \Omega(g(x)).$$

The average complexity of linear search is $\Theta(n)$.

When x is large, big-O provides an upper bound; big-Ω provides an lower bound; big-Θ provides the order.

Another way to show $f(x) = \Theta(g(x))$ is to prove

$$f(x) = O(g(x)), \quad g(x) = O(f(x)).$$

Suppose that $f_1(x) = O(g_1(x))$ and $f_2(x) = O(g_2(x))$. Then

(1) $(f_1 + f_2)(x) = O(\max(|g_1(x)|, |g_2(x)|))$.
(2) $(f_1 f_2)(x) = O(g_1(x)g_2(x))$