Lecture 3: Asymptotic Notations

Asymptotic notations are for big pictures. Big-O is for upper bounds and Omega is for lower bounds. You need to know how to prove say $2n^2 + 4n + 3 = O(n^2)$. To do this, you need to find a constant $C$, s.t. $2n^2 + 4n + 3 \leq Cn^2$. A natural choice for $C = 2 + 4 + 3 = 9$ because $4n \leq 4n^2$ and $3 \leq 3n^2$. Now review the concepts of $O, o, \Omega, \omega, \Theta$ by reading the Chapter 3 of the textbook.

For the example of insertion sort, we previously showed its worst running time $T(n) = C_1n^2 + C_2n + C_3$. Thus, $T(n) = O(n^2)$. Note this is only the worst case running time. When the list is already sorted, insertion sort only takes $\Theta(n)$ time. One should note that insertion sort can not run faster than $\Theta(n)$ since it needs to process each element in the list at least once. The worst-case running time is the most used scheme for algorithm analysis. We say an algorithm is efficient if its worst-case running time $T(n) = O(n^d)$ for some constant $d$. This is called a polynomial time algorithm.

Finally, four common sense rules are useful when dealing with asymptotic notations.

1. Omit constants, e.g. write $14n^2$ as $O(n^2)$.
2. $n^a$ dominates (grows faster) than $n^b$ when $a > b$ (i.e. $n^b = O(n^a)$).
3. Any exponential dominates polynomial, e.g. $n^5 = O(2^n)$.
4. Any polynomial dominates logarithm, e.g. $\log^3 n = O(n)$.

A common way to compare two functions is to take a ratio and then see what the ratio will be when $n$ is large. For example, for $n^a$ and $n^b$, we take the ratio: $n^b/n^a = n^{b-a} < 1$, when $a > b$. Thus, $n^b = O(n^a)$ (remember big-O means less or equal to).

Lecture 4: Common Running Time

We now list the common running times you will see. This also helps you to get more familiar with asymptotic notations and simple algorithm running time.

1. $O(1)$. The fastest running time I can think of for any practical algorithm is constant time algorithm with $O(1)$ time. Example: pushing an element into a stack.

2. $O(\log n)$. Example: binary search. Everyone should know this. If not, you should make sure you understand it: it is widely used.

3. $O(n)$. Often called linear time. Example: find the maximum value from an array. Note you can not do much better than this; you need to look at each array element at least once to ensure you find the largest one. Another example is the problem of merging two sorted list: given sorted lists $A_1$ and $A_2$ with $n_1$ and $n_2$ elements each, generate $A$ that is also sorted and combines the elements in $A_1$ and $A_2$. Here is the algorithm.

```
1: $p_1, p_2 \leftarrow 1$
2: While $p_1 \leq n_1$ and $p_2 \leq n_2$ do
3:   if $A_1[p_1] < A_2[p_2]$ then
4:     put $A_1[p_1]$ to the end of $A$ and $p_1 \leftarrow p_1 + 1$
5:   else
6:     put $A_2[p_2]$ to the end of $A$ and $p_2 \leftarrow p_2 + 1$
```
7:   end if
8: end while
9: if there is still element in $A_1$ or $A_2$ not yet added to $A$, add them to $A$ in the order they appear in the original list.

This algorithm runs in $O(n_1 + n_2)$ time, because each element will be added to $A$ exactly one time, and each insertion takes $O(1)$ time.

4. $O(n \log n)$. One of the most common running time, e.g. in sorting.

5. $O(n^2)$. Often occurs when examining all pairs of states.

6. $O(n^3)$. Example: matrix multiplication algorithm in the homework.