Lecture 6: Examples of non-recursive algorithm analysis, substitution method for recursive algorithm (4.3)

CS303: Algorithms

Last update: January 22, 2014

1 Review

- Properties of asymptotic notations
- Commonly used summation formulas and properties
- How to compare algorithm efficiencies other than asymptotic notations

2 Example of non-recursive algorithm: Bubble sort (problem 2-2 in chapter 2)

Bubble sort is a sorting algorithm that is popular. What it does is repeatedly swapping adjacent elements that are out of order. `http://www.youtube.com/watch?v=lyZQPjUT5B4` (Hungarian folk dance)

```
 Bubble sort (A[1,..., n])
1. for i = 1 to n-1
2.     for j=n downto i+1
3.        if(A[j]<A[j-1])
4.            exchange A[j] with A[j-1]
```

Example: sort 9 6 1 2 3
2:/ Are there three cases?
A:/ Yes, when the array is already sorted, then there is very little swapping done. For the worst case, the array is reversely sorted.

Q3:/ What is the worst case running time
A:/ $T(n) = \sum_{i=1}^{n-1} \sum_{j=n}^{i+1} 2 = \sum_{i=1}^{n-1} (n-i) = (n-1) + (n-2) + ... + 1 = \frac{n(n-1)}{2}$
Q4:/ Can we improve it?
A:/ Yes, if we found out that there is no swapping in one run, then we are done

3 Analysis of recursive algorithms

Recursive algorithms usually relies on solving smaller instances of the same problem and construct the result from there. There are no good algorithms to solve recurrences. Thus it requires some special techniques to analyze its efficiency. We will cover three approaches. Some of them works sometimes and sometimes it doesn't.

3.1 Substitution method (always work but need an oracle))

1. guess the form of the solution (hard, usually we don't have to bother with the constants)
2. verify by induction in the exact form
3. Solve for constants

e.g.
$T(n) = 4T(n/2) + n$
Base case: $T(1) = \Theta(1)$

To solve this problem, we need to guess the form. If we double n, then the workload basically increases by 4. The addition term n is not that important.
Q:/ What is the function that if you double the input, then the workload increases by 4?
A:/ n^2 might be the answer

Let's guess a more vague case
Guess: $T(n)$ is $O(n^3)$
Assume that $T(k) \leq ck^3$ for all $k < n$
Then $T(n) = 4T(n/2) + n \leq 4c(n/2)^3 + n = \frac{c}{2}n^3 + n = cn^3 - (\frac{c}{2}n^3 - n) \leq cn^3$ if $\frac{c}{2}n^3 - n \geq 0$ for all $c > 2$.
Note: You can't use O notation in induction

Or else We can prove using substitution method that $T(n) = n$ is $O(1)$
for $n = 1$, it is true since 1 is $O(1)$
for k, if k is $O(1)$, then $k + 1$ which is $O(1) + 1$ is also $O(1)$. The reason is the constant terms are changing as you induct and it is dependent on 1. Thus you can't induct on asymptotic notations

Base case: $T(1) = \Theta(1) \leq c$ where c has to be sufficiently large
Q:/ Can we also prove $O(n^2)$
A:/ Try it
Guess: T(n) is $O(n^2)$
Assume that $T(k) \leq ck^2$ for all $k < n$
$T(n) = 4T(n/2) + n \leq 4 * c(\frac{n}{2})^2 + n = cn^2 + n \rightarrow$ we are stuck.
Q:/ Now what?
A:/ We need some divine inspirations and some lower terms substracted.
Assume $T(k) <= c_1k^2 - c_2k$ for all $k < n$
Thus
$T(n) = 4T(n/2) + n \leq 4(c_1(\frac{n}{2})^2 - c_2(\frac{n}{2})) + n = c_1n^2 + (1 - 2c_2)n = c_1n^2 - c_2n - (c_2 - 1)n.$
Obviously if $c_2 > 1$ then we are all set $T(n) = 4T(n/2) + n \leq c_1n^2 - c_2n$

Q:/ What is the requirement for c_1?
A:/ c_1 has to be large enough for the base case to work. $T(1) = c_1 - c_2$ and since $T(1)$ is $\Theta(1)$