CSC373 Summer '22

Tutorial 1

Thursday, May 19, 2022

Master Theorem (General Version):

For constants $a \geqslant 1$ and $b > 1$, and an asymptotically positive function $f(n)$, the recurrence relation $T(n) \leqslant a \cdot T(n/b) + O(f(n))$ has the following solution.

1. If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = O\left(n^{\log_b a}\right)$.
2. If $f(n) = \Theta\left(n^{\log_b a} \log^k n\right)$ for some constant $k \geqslant 0$, then $T(n) = O\left(n^{\log_b a} \log^{k+1} n\right)$.
3. If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$ and f satisfies the regularity condition*, then $T(n) = O\left(f(n)\right)$.
   (*Regularity condition: For some constant $c < 1$ and all sufficiently large n, $a \cdot f(n/b) \leqslant c \cdot f(n)$.)

Note: There are recurrence relations which do not fall under any of these three cases (e.g. the recurrence relation $T(n) \leqslant T(n/5) + T(7n/10) + O(n)$ from QuickSelect where the smaller instances are not of uniform size, or the recurrence relation $T(n) \leqslant \sqrt{n} \cdot T(\sqrt{n}) + O(n)$ where a and b are not constants). If you're interested in how more general recurrences can be solved, there are some excellent resources available online.[1][2]

Q1 Practicing Recurrence Relations

Find the best possible asymptotic upper bound for $T(n)$ under the following recurrence relations.[3]

(a) $T(n) \leqslant 3 \cdot T(n/2) + O(n \log^3 n)$

(b) $T(n) \leqslant 4 \cdot T(n/2) + O(n^2)$

(c) $T(n) \leqslant 2 \cdot T(n/2) + O(n \log^2 n)$

(d) $T(n) \leqslant 2 \cdot T(n/4) + O(n^{0.5001})$

Q2 Monotonic Function Evaluation

Consider a monotonously decreasing function $f : \mathbb{N} \to \mathbb{Z}$ (that is, a function defined on natural numbers which takes integer values and satisfies $f(i) > f(i+1)$ for all $i \in \mathbb{N}$). Assuming we can evaluate f at any point i in constant time, we want to find $n = \min\{i \in N | f(i) \leqslant 0\}$ (that is, we want to find the first point where f becomes non-positive). Note that n is not given to us, but we are told that some point i with $f(i) \leqslant 0$ exists (i.e. n is well-defined), and we are allowed to express the running time of our algorithm in terms of n.

---

[1] `http://jeffe.cs.illinois.edu/teaching/algorithms/notes/99-recurrences.pdf`

[2] `http://web.csulb.edu/~tebert/teaching/lectures/528/recurrence/recurrence.pdf`

[3] Note that when proving an upper bound on the worst-case running time of an algorithm, you would encounter equations of the form $T(n) \leqslant \ldots$ rather than $T(n) = \ldots$, yielding $T(n) = O(\cdot)$ rather than $T(n) = \Theta(\cdot)$. To derive a lower bound, you need to explicitly construct instances on which the algorithm takes at least the claimed amount of time.

We can obviously solve the problem in $O(n)$ time by simply evaluating $f(1), f(2), f(3), \dots, f(n)$. Describe an $O(\log n)$ time algorithm.

[Hint: Try to quickly get an estimate of n, and then precisely pinpoint the exact value of n in the range you estimated.]

Q3 Maximum Subarray Sum

You are given an array $A[1 \dots n]$, and you are asked to find the *maximum subarray sum*, that is, the maximum value of $\sum_{t=i}^{j} A[t]$ over all possible (i, j) with $1 \leqslant i \leqslant j \leqslant n$. Design an $O(n)$ time divide and conquer algorithm for the problem.

[Hint: Once you divide the array into two equal halves, say $A[1 \dots \text{mid}]$ and $A[\text{mid}+1 \dots n]$, you will get the maximum subarray sum within each half. What extra information do you need from each half?

If you spend $O(n)$ time in the merge step to calculate this extra information, you will get $O(n \log n)$ running time. Can you get your recursive algorithm to return this information instead?]