PSO #3 Solutions Sketch (Week 4)

Week of 2021-09-13

1 Master Theorem Practice

Note: just a bunch of master theorem practice. Ask how comfortable they feel, and skip or only do a few if desired

1. $T(n) = 3T(n/2) + n^2 \to T(n) = \Theta(n^2)$ (case 3)
2. $T(n) = 7T(n/2) + n^2 \to T(n) = \Theta(n^{\log 7})$ (case 1)
3. $T(n) = 4T(n/2) + n^2 \to T(n) = \Theta(n^2 \log n)$ (case 2)
4. $T(n) = 4T(n/2) + \log n \to T(n) = \Theta(n^2)$ (case 1)
5. $T(n) = T(n-1) + n \to T(n) = \Theta(n^2)$ (MT doesn't apply, can find solution by iteration)
6. $T(n) = 2T(n/4) + n^{0.51} \to T(n) = \Theta(n^{0.51})$ (case 3)
7. $T(n) = 2T(n/4) + \sqrt{n} \to T(n) = \Theta(n^{1/2} \log n)$ (case 2)
8. $T(n) = 7T(n/3) + n^2 \to T(n) = \Theta(n^2)$ (case 3)
9. $T(n) = 16T(n/4) + n \to T(n) = \Theta(n^2)$ (case 1)

2 Proof/Induction Practice

Note: take this problem slow and it solve it with your students as a sort of "guided proof", making sure they're understanding each step

2.1 Setup and Algorithm

Oh no! A new mathematical monopoly has placed a massive tax on all uses of the multiplication and division operations. However, this has not excused your responsibilities in finishing your CS 381 homework. Luckily, your homework this time is relatively simple: compute the square of an integer n. Of course, being an algorithms course and not a math course, they ask you to write a recursive algorithm to complete the task using only bitshifting, addition, and subtraction.
Reminder: Bitshifting is the operation of shifting all the bits in the binary representation of a number by a certain amount to either the left or right. Essentially, shifting to the left by 1 bit is the same as multiplying by 2, increasing the significance of each of the bits that were in the number by 1 place and leaving a 0 in the least significant position. Shifting to the right, on the other hand, is essentially an integer division by 2, decreasing the significance of each of the bits that were in the number by 1 place, eliminating what was previously in the least significant position.

This is the algorithm that you and your collaborators came up with: If n is even, then we know that $n^2 = (n >> 1)^2 << 2$, and if n is odd, then we know that $n^2 = ((n-1)+1)^2 = ((n-1) >> 1)^2 << 2 + 1 + (n-1) << 1$. Thus, by recursively calling these two relationships on n, we may evaluate n^2 by proceeding until the number we give to the relationship has absolute value less than 2, in which case those numbers squared are equal to themselves so there is no longer a need to calculate. For example, if we start out with $n = 4$ and we must compute n^2, we know 4 is even so we have $4^2 = (4 >> 1)^2 << 2$, so we now have to compute $y = (4 >> 1)^2$, and if we can do that, then we will have $4^2 = y << 2$. Because y is just another number squared, then we can evaluate $4 >> 1 = 2$, so our goal is to compute 2^2, and since 2 is even, $2^2 = (2 >> 1)^2 << 2$, and since $2 >> 1 < 2$, we have reached a base case and no longer need to call recursively. So, from here, we know that $y = 2^2 = 1 << 2 = 4$, so $4^2 = y << 2 = 4 << 2 = 16$, which means we have solved the problem.

2.2 Proof of correctness

Base Cases: $n = 0, n = 1$. Both 0, 1 are the squares of themselves, thus we do not need to execute any recursive calls on these two numbers.
Inductive Hypothesis: Assume for all $k \leq n$ that recursively calling on k the two equations above eventually yields a series of operations that reconstruct k^2.
Inductive Step: If we take $(k+1)^2$, we have two cases: one where $k+1$ is even, the other where it is odd.
Case 1: $k+1$ is even. We know $(k+1)^2 = ((k+1) >> 1)^2 << 2$, but since we know from inductive hypothesis that $((k+1) >> 1)^2$ has a sequence of steps that can be reconstructed until $(k+1) >> 1$ is 1 or 0, we know simply substitute those steps in place of $((k+1) >> 1)^2$ in the expression for $(k+1)^2$ and we determine that $k+1$ also has such a sequence of recursive calls to reach 1 or 0.
Case 2: $k+1$ is odd. We know $(k+1)^2 = (k >> 1)^2 << 2 + 1 + k << 1$, and since we know by inductive hypothesis $(k >> 1)^2$ has a construction of steps that can start from when k goes down

to 0 or 1, we know $k + 1$ also has such steps.

2.3 Proof of Runtime

The runtime of this algorithm is $O(\log n)$, as for a number n, for each step of the recursive call, it is divided in 2 via a right-shift by 1 bit. To be more precise, the recurrence would be $T(n) = T(n/2)+c$, where c is the amount of work required to do the excess additions and shifting at each step, which always takes effectively the same amount of time (we ignore the differences in addition time for larger and smaller numbers and assume it is constant).

3 Another Induction

problem:

Given DeMorgan's Law for 2 sets, prove the generalized version for n sets.
Given: $\overline{A \cap B} = \overline{A} \cup \overline{B}$
Prove: $\overline{A \cap B \cap \ldots} = \overline{A} \cup \overline{B} \cup \ldots$

solution:

Base case: $n = 2$, holds by DeMorgan's Law
Inductive hypothesis: Assume $\overline{A_1 \cap A_2 \cap \ldots \cap A_k} = \overline{A_1} \cup \overline{A_2} \cup \ldots \cup \overline{A_k}$
Prove $\overline{A_1 \cap A_2 \cap \ldots \cap A_k \cap A_{k+1}} = \overline{A_1} \cup \overline{A_2} \cup \ldots \cup \overline{A_k} \cup \overline{A_{k+1}}$
For ease of notation, let $B = A_1 \cap A_2 \cap \ldots \cap A_k$

$$\begin{aligned} \overline{A_1 \cap A_2 \cap \ldots \cap A_k \cap A_{k+1}} &= \overline{B \cap A_{k+1}} && \text{definition of B} \\ &= \overline{B} \cup \overline{A_{k+1}} && \text{De Morgan's Law} \\ &= \overline{A_1} \cup \overline{A_2} \cup \ldots \cup \overline{A_k} \cup \overline{A_{k+1}} && \text{IH} \end{aligned}$$