Massachusetts Institute of Technology
6.042J/18.062J: Mathematics for Computer Science
Professors David Karger and Tom Leighton

Handout 32
October 17, 1995

Notes 11

1 Hanoi Towers

We'll start with a simple and famous example of a recurrent problem known as the towers of Hanoi, invented by French mathematician Edouard Lucas in 1883.

In this problem, we are given a collection of disks of different sizes and three pegs. At the beginning, all disks are stocked in decreasing order of size on one peg. Our objective is to move all disks from this peg to another one moving just one disk at a time and never moving a larger disk onto a shorter one.

At first, it is not even clear that it is possible to do. We are going to give a general method which works for n disks whatever n is (the original problem was about 64 disks) and we will estimate the number of moves needed.

Definition 1.1 *Let $T(n)$ be the number of steps needed to move the n-disk tower of the first peg to another peg.*

It is easy to check that $T(1) = 1$, $T(2) = 3$ and $T(3) \leq 7$. Now let us consider the following recursive solution to our problem:

- first, move the upper n-1 disks to the third peg,
- then move the largest disk to the second peg,
- finally move the (n-1)-disk-tower from the third peg to the second peg.

As the first and the third step take $T(n-1)$ moves each and the second step takes one move, this shows that $T(n) \leq 2T(n-1) + 1$. Therefore, there exist a solution for all n. However, it could possibly be the case that our recursive algorithm is not optimal. We are going to show that in fact, it is optimal, there is no better way to move the disks.

To see why, notice that at some point we have to move the largest disk, say, from peg 1 to peg 2. The only moment when it is possible is when all the other disks are on peg 3 and then they necessarily are in decreasing order of size. But we need at least $T(n-1)$ moves to reach that position and then we need one move to move the largest disk and then $T(n-1)$ more moves to put the (n-1)-disk-tower onto it. Therefore, $T(n) \geq 2T(n-1)+1$, thus $T(n) = 2T(n-1)+1$.

It is not hard to guess a closed formula for $T(n)$. If we compute some values using the just obtained recursion, we will conjecture that $T(n) = 2^n - 1$. A very easy inductive proof shows that this is indeed correct.

In general this "guess and verify" method is best to solve recurrences. If it is not so easy to guess what the correct formula is, one can try the so-called *Plug and Chug* or *Brute Force* method we illustrate here:

$$T(n) = 1+2T(n-1) = 1+2+4T(n-2) = 1+2+4+T(n-3) = \cdots = 1+2+4+\cdots+2^{n-1}$$

However, it is very easy to make a mistake when observing the pattern, so it is important to verify our answer is correct.

2 Merge Sort

One of the most popular sorting algorithms is as follows:

To sort n items $x_1, x_2, \cdots, x_n$

- sort $x_1, x_2, \cdots, x_{n/2}$ and $x_{n/2+1}, x_{n/2+2}, \cdots, x_n$
- merge the two lists together.

To merge two lists, simply compare the smallest items of the two lists and put the smaller at the start of the merged list and keep the other one at the top of its short list. Then do it again with the remains of the two lists. Do it as long as there are elements which haven't been put on the new merged list yet.

Definition 2.1 *Let $T(n)$ be the number of comparisons needed to sort n numbers.*

To estimate $T(n)$, let us note that it takes $2T(n/2)$ steps to sort the two lists of length $n/2$ and $n-1$ steps (at worst) to merge the two lists together. Therefore, $T(n) = 2T(n/2)+n-1$.

Applying the plug and chug method we get that

$$T(n) = (n-1) + 2T(n/2) = n-1+n-2+4T(n/4)+\cdots+n-2^i+2^iT(n/2^i)$$

, where $T(n/2^i) = 0$ for $i = log_2(n)$. Thus

$$T(n) = \sum_{i=0}^{log(n)-1}(n-2^i) = \sum_{i=0}^{log(n)-1} n - sum_{i=0}^{log(n)-1}2^i = nlog(n) - (2^{log(n)}-1) = nlog(n) - n + 1$$

The reader should verify by induction that the estimate proven for $T(n)$ is indeed correct.

3 Divide and Conquer recurrences

Let $T(n) = aT(n/2) + bn$ and $T(1) = 1$, where a and b are constant. We'll examine the effect of having a instead of 2 here. When you solve this recurrence by expansion, you get that

- for $a < 2$, you get $T(n) = \frac{2bn}{2-a} = \Theta(n)$,
- for $a = 2$, you get $T(n) = bnlog(n) = \Theta(nlogn)$, and
- for $a > 2$, you get $T(n) = \frac{ab}{a-2} n^{log(a)} = \Theta(n^{log(a)})$.

Thus we can conclude that the reccurence is very sensitive to any changes in a, but it varies only linearly if we change b. Hence, if we don't care about constant factors, we can write $T(n) = aT(n/2) + \Theta(n)$, with $\Theta(1) = 1$, but we cannot write $T(n) = \Theta(T(n/2))$ without losing a lot of sharpness in oour estimate.

Like in the previous paragraph, people often use asymptotic notation when they write recurrences. While doing this, it is easy to make false proofs by mixing big O and induction. The key to avoid these mistakes is to make sure that your induction hypothesis makes sense, that is, that it states something meaningful on a number n. For example, it is meaningless to say that "for all n, T(n) = O(n), since T(n) is just a number, not a function, whereas O(n) denotes a class of functions. The correct way to state the above assertion is: "for all n, we have $T(n) \leq cn$. As a general advice, keep big O out of inductive proofs!

Now let us consider the following rather ugly recurrence:

$$S(n) = S(\lfloor n/2 \rfloor) + S(\lceil n/2 \rceil) + 1$$

and let $S(1) = 0$. It would be teadious to expand this recurrence. Therefore, we solve a recurrence which seems to be very similar to this one, and then verify that the result indeed fulfills the above recurrence, too. Namely, first we easily solve $S(n) = 1 + 2S(n/2)$, $S(1) = 0$ and get $S(n) = n - 1$. Then we guess that this solution satisfies the previous recurrence as well, and check that statement by induction, finding that our guess is correct.

Note that this recurrence looked pretty similar to that of the Towers of Hanoi and there was a tremendous difference in the results. This warns us that it is not always easy to get a good guess, especially if we are unexperienced.

So far we have seen a lot of examples on why induction is a great way to prove theorems. Now let's see some of its limits. Let us consider again the last recurrence, that is, $S(n) = 1 + 2S(n/2)$, and try to prove by induction that $S(n) < n$.

- $S(1) = 0 < 1$

- $S(n) = 1 + 2S(n/2) = 2(n/2) + 1 \leq n + 1$

which is not enough, as we need $S(n) < n$.

This doesn't mean that the result is not true, it is straightforward to prove by induction that $S(n) = n - 1$, it just means that we couldn't establish it this way, as the choice of our claim was wrong, *it was not strong enough.* This shows that sometimes it is easier to prove a stronger statement by induction than a weaker one.

Once again, we point out the importance of checking the base case. For example, here we could "deduce" the induction step of the false claim $S(n) < n - 2$, but it would not be a proof as this claim doesn't hold for $n = 1$.

4 Recurrences in a product form

Consider the recurrence $T(n) \leq nT(\sqrt{n})^2$. As we deal with a product, it is convenient to take logarithms to get

$$logT(n) \leq 2logT(\sqrt{n}) + logn.$$

Next, set $S(n) = logT(n)$ to get

$$S(n) \leq 2S(\sqrt{n}) + logn.$$

Expanding this we have $S(n) = O(log(n) \cdot log(log(n))$, which yields

$$T(n) = 2^{S(n)} = 2^{O(log(n) \cdot log(log(n))} = n^{O(log(log(n))}.$$