Recurrence Equations

Recitation Week 2

Solve the following recurrence using the master theorem, a recursion tree, and the substitution method.

$$T(n) = T(n/2) + \Theta(n)$$

Need to describe what happens at base level. Say for n <= 2, we perform O(1) work.

THE MASTER THEOREM

$$T(n) = T(n/2) + \Theta(n)$$

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Which case of the master theorem applies?

a = ?
b = ?
f(n) = ?

We apply case 3 of the master theorem. Let $a = 1$ and $b = 2$, we get $\log_b a = \log_2 1 = 0$. If we let $\epsilon = 0.5$, then $0 + 0.5 = 0.5$ and $f(n) = kn \in \Omega(n^{0.5})$. Taking $c = 0.9$ we see that $k(n/2) \leq 0.9kn$. This means that the third rule is applicable and $T(n) = \Theta(n)$.

RECURSION TREE

$$T(n) = T(n/2) + \Theta(n)$$

- WHAT IS THE DEPTH OF THE TREE?
- WHAT IS THE WIDTH?
- WHAT IS THE COST AT EVERY LEVEL?

We can see that the size of the problem halves at every step, meaning that the depth of the tree is $\log n$. The cost at every level is a constant times the size of the problem. Therefore we can write the total work as

$$\sum_{i=0}^{\log n} n/2^i \leq n \sum_{i=0}^{\infty} \frac{1}{2^i} \leq 2n.$$

We have that the total cost is then $O(n)$. To see that the cost is also $\Omega(n)$, we can see that the cost is at least n even just from the top level. This implies that the cost is $\Theta(n)$.

SUBSTITUTION METHOD

$$T(n) = T(n/2) + \Theta(n)$$

- WE CAN USE THE RECURSION TREE METHOD TO FIND A CANDIDATE GUESS (O(N))
- USE THE GUESS AS THE INDUCTIVE ASSUMPTION AND SUBSTITUTE TO SHOW

We guess that the function is $O(n)$ and make an inductive substitution. Assume that $T(n') \leq cn'$ for all $n' < n$. Then we have $T(n) = T(n/2) + \Theta(n) \leq c(n/2) + kn$. Let $c = 2k$, then we have that $T(n) \leq kn + kn \leq 2kn \leq cn$ and our induction holds.

We can similarly show the lower bound by guessing that the function is $\Omega(n)$. Assume that $T(n') \geq cn'$ for all $n' < n$. Then we have $T(n) = T(n/2)+\Theta(n) \geq c(n/2)+kn$. Let $c = k$. Then $T(n) \geq kn/2+kn \geq kn \geq cn$, so our induction holds.

Solve the following recurrence using the ~~master theorem,~~ a recursion tree, and the substitution method.

$$T(n) = T(n/2) + \text{T(n/4) + T(n/8) + } \Theta(n)$$

RECURSION TREE

$$T(n) = T(n/2) + T(n/4) + T(n/8) + n$$

- WHAT IS THE DEPTH OF THE TREE?
- WHAT IS THE WIDTH?
- WHAT IS THE COST AT EVERY LEVEL?

The total cost at the first level is n, and the total cost of the second level is $7n/8$. In general, at depth d, the total cost of the tree is at most $(7/8)^d n$, so the total cost when we sum up over all levels is going to be at most the following geometric series:

$$\sum_{d=0}^{n} (7/8)^d n \leq \sum_{d=0}^{\infty} (7/8)^d n = 8n.$$

Therefore, we know that the total cost is $O(n)$. Since the first level costs n already, we have that the total cost is also $\Omega(n)$. Finally, this shows that the total cost is $\Theta(n)$.

SUBSTITUTION METHOD

$$T(n) = T(n/2) + T(n/4) + T(n/8) + n$$

- WE CAN USE THE RECURSION TREE METHOD TO FIND A CANDIDATE GUESS (O(N))
- USE THE GUESS AS THE INDUCTIVE ASSUMPTION AND SUBSTITUTE TO SHOW

From our recursion tree, we have the guess of $\Theta(n)$. We assume that $T(n') \leq cn'$ for all $n' < n$, and we want to show that $T(n) \leq cn$. Upon substitution, we get:

$$\begin{aligned} T(n) &\leq c(n/2) + c(n/4) + c(n/8) + n \\ &= (7c/8)n + n = (7c/8 + 1)n. \end{aligned}$$

If we take $c = 8$, we get

$$T(n) \leq (7c/8 + 1)n = 8n = cn.$$

Thus, we know that $T(n) = O(n)$.

SUBSTITUTION METHOD

$$T(n) = T(n/2) + T(n/4) + T(n/8) + n$$

- WE CAN USE THE RECURSION TREE METHOD TO FIND A CANDIDATE GUESS (O(N))
- USE THE GUESS AS THE INDUCTIVE ASSUMPTION AND SUBSTITUTE TO SHOW

From our recursion tree, we have the guess of $\Theta(n)$. We then assume that $T(n') \geq dn$ and we want to show that $T(n) \geq dn$. Following a symmetric sequence of arguments, we get:

$$\begin{aligned} T(n) &\geq d(n/2) + d(n/4) + d(n/8) + n \\ &= (7d/8)n + n = (7d/8 + 1)n. \end{aligned}$$

If we take $d = 8$, we get

$$T(n) \geq (7d/8 + 1)n = 8n = dn,$$

which proves that $T(n) = \Omega(n)$.

EXAMPLES

- GIVE ASYMPTOTIC UPPER AND LOWER BOUNDS FOR $T(n)$. ASSUME THAT $T(n)$ IS CONSTANT FOR SUFFICIENTLY SMALL n

1. $T(n) = 4T\left(\frac{n}{3}\right) + n \log n$
2. $T(n) = 4T\left(\frac{n}{2}\right) + n^2\sqrt{n}$
3. $T(n) = 3T\left(\frac{n}{3}\right) + n/\log n$

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

We first seek to apply the master theorem. If we let $f(n) = n/\log n, a = 3$, and $b = 3$, we see that $n^{\log_b a} = n^{\log_3 3} = n$. Since $n/\log n \in o(n)$, it is clear that the second and third rule of the master theorem cannot apply. But, as it happens, the first rule does not apply either, since $n/\log n \in \omega(n^{1-\epsilon})$ for any constant $\epsilon > 0$.

DERIVING A RECURRENCE EQUATION

- FIND A RECURRENCE EQUATION FOR THE WORST CASE RUNNING TIME OF THE FOLLOWING CODE BLOCK AND FIND THE ASYMPTOTIC RUNNING TIME USING THE MASTER THEOREM. THEN, FIND THE *EXACT* WORST CASE RUNNING TIME USING A RECURSION TREE

```
1  function mystery(A, k)
2  r := ⌊A.length/2⌋
3  if k = A[r] then
4      return r
5  else
6      if A.length ≤ 1 then
7          return −1
8      else
9          if A[r] > k then
10             return mystery(A[r + 1..A.length], k)
11         else
12             return mystery(A[1..r], k)
```

BINARY SEARCH

- THE RECURRENCE EQUATION CAN BE FOUND BY OBSERVING THAT THE PROBLEM SIZE HALVES EVERY TIME AND THAT WE DO A CONSTANT AMOUNT OF WORK AT EACH RECURSIVE DEPTH. FINALLY, THE RECURSIVE BEHAVIOR TERMINATES WHEN THE ARRAY IS OF SIZE LESS THAN OR EQUAL TO 1.

$$T(n) = T(n/2) + \Theta(1)$$

Theorem 4.1 (Master theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

We apply case 2 of the master theorem. Let $a = 1$ and $b = 2$, we get $\log_b a = \log_2 1 = 0$. We have that $n^{\log_2 1} = 1$ and $f(n) \in \Theta(1)$, so case 2 holds. Therefore, we have that the running time is $\Theta(n^0 \log n) = \Theta(\log n)$.

BINARY SEARCH

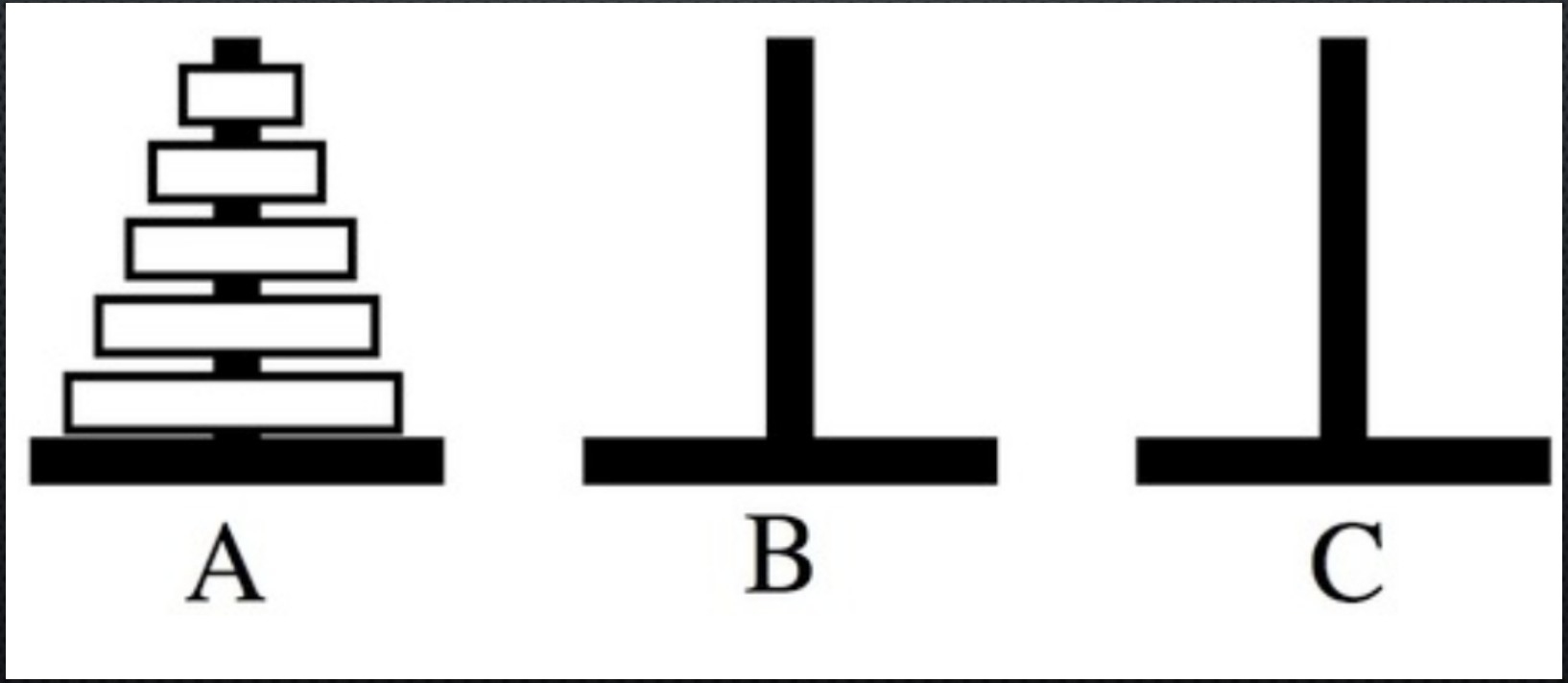
- WHAT IS THE DEPTH OF THE TREE?
- WHAT IS THE TOTAL WORK AT EACH LEVEL?

We can see that the size of the problem halves at every step, meaning that the depth of the tree is $\lceil \log n \rceil + 1$. The cost at every level is a constant. Therefore we can write the total work as

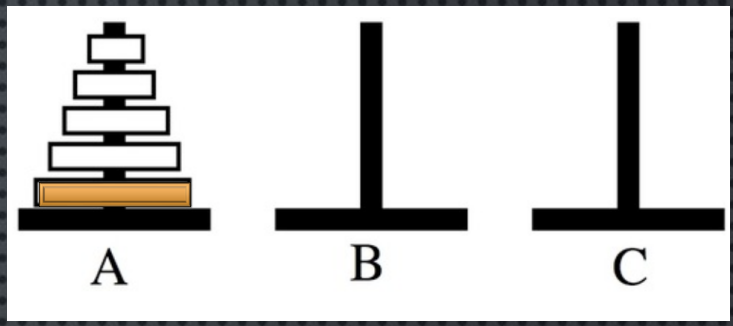
$$\sum_{i=0}^{\lceil \log n \rceil + 1} c = c(\lceil \log n \rceil + 1).$$

TOWERS OF HANOI

- In this problem, we will characterize the "difficulty" of the classic Towers of Hanoi puzzle (i.e., how long it takes to complete). In the puzzle, you are to move the entire tower on A to C one disc at a time. However, a larger disc cannot be moved on top of a smaller disc. Suppose that moving one disc from one tower to another takes constant time in any implementation. Propose a recursive algorithm to solve this problem and analyze it. Then, find a lower bound for the running time of any solution implementation.



TOWERS OF HANOI – HINTS AND SOLUTION



- How can we move the orange disc to stack C?
- Since N is the number of discs, what would a recurrence relation T(N) look like?
- Is there any way to avoid moving all discs on top of the orange to stack B before moving the orange disc in C?

For any implementation, we must move the entire stack above the bottom disc on A to B. Then we may move the bottom disc to C. Finally, we can move the smaller stack now at B to C. This gives us a recursive structure implying a recurrence:

$$T(n) = 2T(n-1) + \Theta(1).$$

We can solve this directly with a recurrence tree. Observe that the depth is linear. Further there are 2^d nodes at each level each with constant cost. This implies that the total cost is $\sum_{d=0}^{n} c \cdot 2^d \leq c \cdot \frac{2^{n+1}-1}{2-1} \leq 2c \cdot 2^n = \Theta(2^n)$.

ONE MORE RECURRENCE

- Solve the following recurrence when $T(n) = \Theta(1)$ for $n \leq 2$

$$T(n) = \sqrt{n}T(\sqrt{n}) + n$$

- Can we use the master method?
- Hint: Try making a change of variables (Page 86 in the text) with $m = \log(n)$

$$T(n) = \sqrt{n}T(\sqrt{n}) + n$$

In order to solve this recurrence equation we will be performing a change of variables (see Page 86 of your textbook). In particular, just like in the textbook, we will be renaming $m = \log n$, which yields $T\left(2^m\right) = 2^{m/2}T\left(2^{m/2}\right) + 2^m$. If we divide both sides with 2^m, we get

$$\frac{T\left(2^m\right)}{2^m} = \frac{T\left(2^{m/2}\right)}{2^{m/2}} + 1.$$

We can now rename $S(m) = \frac{T(2^m)}{2^m}$, which reduces our initial recurrence equation to $S(m) = S(m/2) + 1$, which is much easier to solve. To use the master theorem to solve $S(m)$, we must identify a, b, and $f(m)$ in our new recurrence equation. We have $a = 1$, $b = 2$ and $f(m) = 1$, so $m^{\log_b a} = m^0 = 1$. Therefore, $f(m) = 1 \in \Theta(1) = \Theta(m^{\log_b a})$ and, using the second case of the master theorem, we get $S(m) \in \Theta(\log m)$. Since $S(m) = \frac{T(2^m)}{2^m}$, this implies that $T(2^m) = 2^m S(m) \in \Theta(2^m \log m)$. Finally, since $m = \log n$, we conclude that

$$T(n) = \Theta(n \log \log n).$$

$$T(n) = \sqrt{n}T(\sqrt{n}) + n$$

Does this mean that mergesort can be made asymptotically better by splitting the problem into $\sqrt{n}$ parts?

In order to solve this recurrence equation we will be performing a cl
86 of your textbook). In particular, just like in the textbook, we wi
which yields $T\left(2^m\right) = 2^{m/2}T\left(2^{m/2}\right) + 2^m$. If we divide both sides wit

$$\frac{T\left(2^m\right)}{2^m} = \frac{T\left(2^{m/2}\right)}{2^{m/2}} + 1.$$

We can now rename $S(m) = \frac{T(2^m)}{2^m}$, which reduces our initial recurrence equation to $S(m) = S(m/2) + 1$, which is much easier to solve. To use the master theorem to solve $S(m)$, we must identify a, b, and $f(m)$ in our new recurrence equation. We have $a = 1$, $b = 2$ and $f(m) = 1$, so $m^{\log_b a} = m^0 = 1$. Therefore, $f(m) = 1 \in \Theta(1) = \Theta(m^{\log_b a})$ and, using the second case of the master theorem, we get $S(m) \in \Theta(\log m)$. Since $S(m) = \frac{T(2^m)}{2^m}$, this implies that $T(2^m) = 2^m S(m) \in \Theta(2^m \log m)$. Finally, since $m = \log n$, we conclude that

$$T(n) = \Theta(n \log \log n).$$