The Master Method

While analyzing algorithms, we come across various types of recurrences. For example, here is a particular recurrence.

Equation 1

$$t(n) = 2t\left(\frac{n}{2}\right) + n$$

Question: In the analysis of what algorithms does this equation occur? Think of as many places as you can where this equation will be used.

[1]

One of the places the above equation occurs is in the analysis of the merge sort. $t(n)$ is the time taken by merge sort to sort n elements. Equation 1 says that the time taken to merge sort n elements is the time taken to merge sort two sets of $n/2$ elements plus the time taken to merge the two sorted lists (which is n).

Another place the above equation occurs is in the best case analysis of quick sort. In the best case, the quick sort pivot lands exactly in the center, dividing the set to be sorted into two exactly equal sorts. For the quick sort, the equation says that in the best case, the quick sort on n elements does a separation pass (taking time n) followed by two quick sorts on $n/2$ elements.

The same equation will occur in many more algorithms. Equation 1 is in fact one of the most commonly occurring recurrences in algorithm analysis.

Solving recurrences is a major part of analyzing algorithms. Recurrences generally occur directly due to recursive algorithms. Recurrences can occur due to other reasons also. (In fact, even if we were using the iterative merge sort, we could analyze it using the same recurrence relation.)

Question: What is the solution of the above recurrence?

[1] The blue horizontal fence means you should stop here and think before continuing.

Those who remember merge sort will quickly realize that the solution to the above recurrence is $t(n) \in \theta(n \log n)$. Once we know that this is the solution, we can easily prove this using constructive induction[a].

The question is how to come up with $t(n) \in \theta(n \log n)$ in the first place? Is there a general method by which we can just look at a recurrence and say what the complexity is going to be? This document is about such a general method. It's called the Master Method.

Before we go on to learn about the Master Method, let us try to analyze merge sort further for a bit.

Question: Why does merge sort take $\theta(n \log n)$ time? Can we visualize the complexity of merge sort?

2

Look again at the recurrence relation

Equation 1

$$t(n) = 2t\left(\frac{n}{2}\right) + n$$

The merge sort "sorting action" all happens in the "merge pass", in the "conquer" of the divide and conquer methodology. That is, it is the $+n$ factor in the above equation at various levels of recursion that together causes the merge sort time complexity. If we were to draw all these $+n$s at all the levels of recursion, we will get a "picture" of the complexity of merge sort.

Question: Try to draw this picture of the complexity of merge sort.

[2] This document would be most beneficial if you tried to answer the blue fence questions yourself.

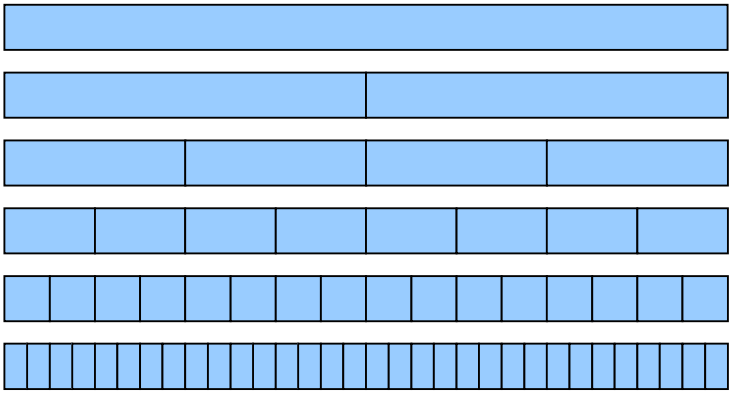


Figure 1: "Picture" of the complexity of merge sort

Question: Label this figure in your mind. What does every element mean? Can you argue using this figure that the complexity of merge sort is $\theta(n \log n)$?

3

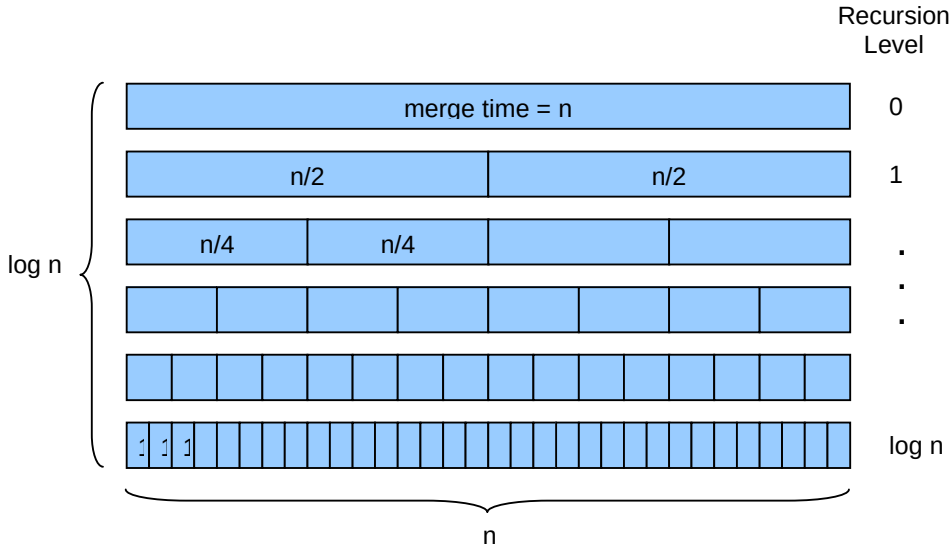


Figure 2: The complexity of merge sort is $\theta(n \log n)$

Figure 2 is Figure 1 redrawn to show how the complexity of merge sort is distributed among the recursion levels of the merge sort. The top level

[3] It is a good idea to hold a paper below the blue fence, so that you do not strain yourself trying hard *not* to look below it!

performs one merge pass of n elements. The second level performs 2 passes, each merging $n/2$ elements. Thus the total time consumed in the second level is also n. Depending on the control structure of the merge sort, these two merges may or may not happen consecutively. For a recursive merge sort, the two merges at level 2 will be separated by many deeper-level merges. But, the total amount of merging action at level 2 will amount to a total time of n. Continuing this argument, we get the diagram shown in Figure 2. Since the recursion continues to a depth of $\log n$, it is obvious[45] by looking at the diagram that the total amount of time taken by merge sort is $\theta(n \log n)$.

Now, it is obvious that the above "picture form" analysis holds not only for merge sort in particular, but in general to the recurrence relation

Equation 1

$$t(n) = 2t\left(\frac{n}{2}\right) + n$$

How may we generalize the above recurrence relation? It can be done in many ways. When generalizing, always remember – the generalization should be useful. A generalization of something is something that applies to a lot of things. These "lot of things" must be things we would like to apply the generalization to. Thus, before we generalize, we should know at least a few of this lot of things. So, to avoid "*shitaawaruun bhaataachii pariikshaa*[6]", we will see a few more particular recurrences.

Well, one more, anyway –

On a particular rainy evening, n partially drunk people in a bar decide to choose the arm wrestling champion among themselves. There is only one table which may be used for arm wrestling. The (partially drunk) computer engineer tells them that they should play in divide and conquer fashion – she advises them to first pair off the n people and play one set of matches, and continue from there, with only the winners (since none of the losers is the champion) playing the next round. "Oh! Knockouts!" say the other partially drunk attendees, and they decide to do exactly that…

Question: How much time will it take to decide a winner? Can you write a recurrence relation for the time taken?

---

[4] The number of levels is actually $(\log n) + 1$ since the levels are numbered in Figure 2 starting from 0. But $(\log n) + 1$ and $\log n$ are asymptotically the same value.

[5] Whenever we say "log", the logarithm is to the base 2.

[6] Judging the rice after eating a grain

Let us assume that it takes exactly two minutes to play one arm wrestling match[7]. The recurrence relation for the total time taken to play the knockout matches between n people is

Equation 2

$$t(n) = t\left(\frac{n}{2}\right) + n$$

The above equation says that the time taken to choose a champion among n people is the time taken to play $n/2$ knockout matches (which is n), plus the time taken to choose a champion among the remaining $n/2$ candidates.

Note that if n were an odd number, we should have actually used $\lceil n/2 \rceil$ in the above equation. But in these pages, we will disregard the non-integer effects, leaving them to the careful authors of your favorite textbooks.

Question: Draw the complexity "picture" for this recurrence, and find the total time taken to find a champion.

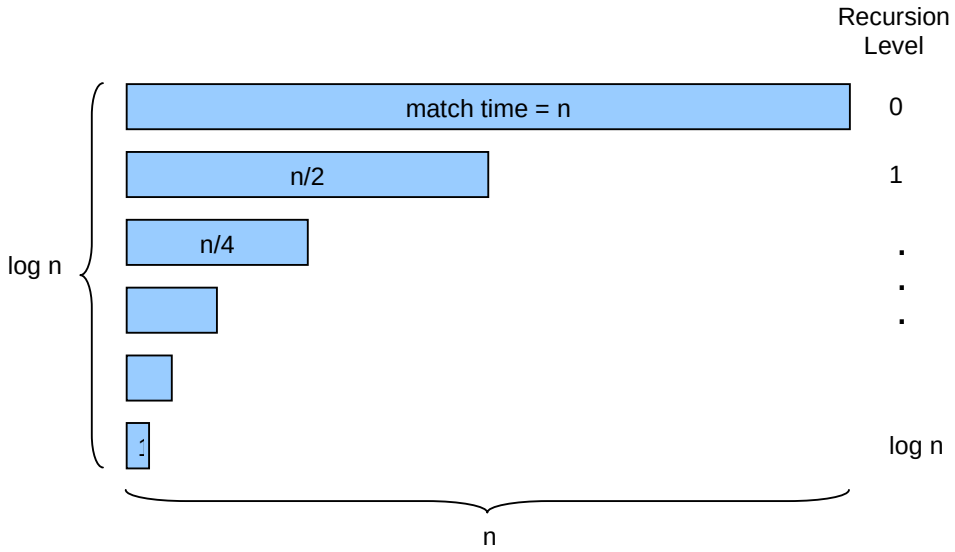


Figure 3: The time taken for n people to decide a champion among themselves.

[7] This could be any constant (as we will see later in the document) – and our asymptotic analysis will not change. Furthermore, this need not even be a sharp constant. If we had some bound like "each match takes at least one minute, and no match lasts more than three minutes" that also would be good enough.

Figure 3 depicts the total time taken in the recursion of Equation 2. As before, the actual time consumed is all in the $+n$ factor in the equation. That is as it should be. The total time taken in playing all the matches *should* be consumed in playing the matches!

Question: Give a good asymptotic approximation for the total amount of time spent in the matches, i.e. the total blue area in Figure 3.

Looking at Figure 3, it is obvious that $n \log n$ is an upper bound on the total blue area, because the whole figure is contained within the rectangle having sides n and $\log n$. But, if you picture the $n \log n$ rectangle, it should be obvious that we are over-counting. Turns out we are over-counting a lot:

From the figure, we see that the total time for deciding the champion is given by

Equation 3

$$t(n) = \underbrace{n + \frac{n}{2} + \frac{n}{4} + \ldots}_{\log n \text{ terms}}$$

Simplifying, we get

Equation 4

$$t(n) = n\underbrace{\left(1 + \frac{1}{2} + \frac{1}{4} + \ldots\right)}_{\log n \text{ terms}}$$

The terms in the brackets form a geometric progression with $\log n$ terms. But, we can overestimate it with an infinite term geometric series. Thus, we get

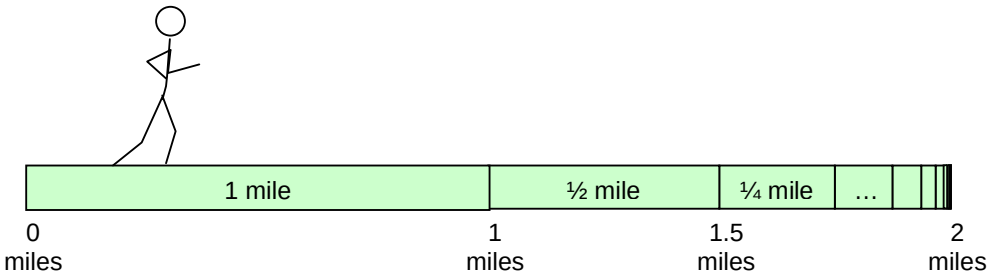
Equation 5

$$t(n) \leq n\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots\right)$$

How much does the geometric series add up to? The above geometric series $(1 + 1/2 + 1/4 + \ldots)$ is probably the most common series encountered in the analysis of algorithms, and we should just *know* that it adds up to 2.

There are many ways to visualize this. Think of the following: the mythical hero Achilles ran two miles. Well, he first ran one mile, which brought him half the distance to his goal. Then he ran another half a mile, bringing him half the remaining distance. Then he ran one fourth of a mile more, bringing him another exactly half the distance of his goal of two miles. (Actually Achilles just ran two miles, but the above description is definitely *one* valid description (albeit maybe not thrilling) of his endeavor.) If you go on making such

"segments" of Achilles' two mile run, you see that the segments form the exact geometric series we are talking about. Thus, one way of looking at Achilles' run is $(1+1/2+1/4+\ldots)$ miles, and another way of looking at it is just 2 miles![8]



The general formula for a geometric series is

Equation 6 $$1+r+r^2+r^3+\ldots=\frac{1}{1-r}$$

This formula holds provided $r<1$. On the other hand, if $r>1$, we have a divergent geometric series which has no finite limit. Even for $r=1$ the series does not converge (which is rather obvious – an infinitely many 1's are not going to add up to a finite number.)

Now, we do not have to learn Equation 6 by heart, because it is very simple to derive. If we take

$$S=1+r+r^2+r^3+\ldots$$

Multiplying that by r leads us to

$$rS=r+r^2+r^3+\ldots$$

Subtracting the second equation from the first gives us, marvelously,

$$\begin{aligned} S-rS &= 1-r+r-r^2+r^2-r^3+r^3-\ldots \\ S-rS &= 1-\cancel{r}+\cancel{r}-\cancel{r^2}+\cancel{r^2}-\cancel{r^3}+\cancel{r^3}-\ldots \\ (1-r)S &= 1 \end{aligned}$$

This gives us the formula for adding up a geometric series, which is

Equation 6 $$1+r+r^2+r^3+\ldots=\frac{1}{1-r}$$

[8] This story was used by Zeno to argue that motion does not exist! We use it to show the mundane (in comparison) fact that $1+1/2+1/4+\ldots=2$.

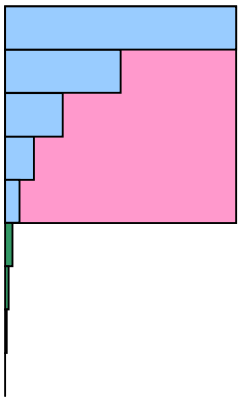
Coming back to Equation 5, we had got that the solution of the recurrence $t(n) = t(n/2) + n$ was bounded by

Equation 5
$$t(n) \le n\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots\right)$$

Now, substituting for the geometric series, we get that

Equation 7
$$t(n) \le 2n$$

Let us try to draw a picture of what we have done. Our goal was to estimate the blue area in Figure 3. There are two ways of doing so, as shown below.



The pink area is the area added by the $n \log n$ upper bound. The green area is the area added by the $2n$ bound. For sufficiently large n, the $2n$ bound gives an almost exact value for $t(n)$. In our specific case, in fact, the $2n$ bound will be off by exactly one match (two minutes), whereas the $n \log n$ bound will go on becoming worse as n increases.

Thus, for the recurrence $t(n) = t(n/2) + n$, we have found the solution $t(n) \approx 2n$. In fact, since the error is bounded by a constant, we can say $t(n) \in \theta(n)$.

It is worth noting, however, that the $n \log n$ upper bound is actually better (or sharper) than the $2n$ upper bound for the first few values of n. Here the effect lasts only for $n \le 4$. But, in some other cases (as we will see in some time), the rectangle bound will turn out to be better than the geometric series bound for a large number of values. So, even though asymptotically our conclusion would be in favor of the geometric series bound, one should be aware of the other bounds which can also be applied. The usual surgeon-general's warning about asymptotics stays: use them with care! Hidden constants can be dangerous to your health!

Question: Now take a look at the recurrence of Equation 1 and its associated picture Figure 1, as well as Equation 2 and its associated picture Figure 3. How would you like to generalize the above two equations to include a large number of cases?

There are many generalizations that you can think of the two equations:

Equation 1 $$t(n) = 2t\left(\frac{n}{2}\right) + n$$

Equation 2 $$t(n) = t\left(\frac{n}{2}\right) + n$$

One of the ways we can generalize is to change the "work" function (the "conquer" pass, if the equation is representing a divide and conquer algorithm) from the current "$+n$" to other functions like n^2 or $n\log n$ etc. This is what we will eventually do, but not yet. There is a simpler generalization that we must do first[b]:

We are going to think about the following general recurrence relation

Equation 8 $$t(n) = at\left(\frac{n}{b}\right) + n$$

The two recurrence relations in Equation 1 and Equation 2 are special cases of this general recurrence relation. Thus, we have already solved the cases $(a = 2, b = 2)$ and $(a = 1, b = 2)$. The two cases we solved are fundamentally different, in that we applied two different bound techniques (the rectangle and the geometric series) to derive our answers. We should try to extend the two cases to as many combinations of a's and b's as we can.

But, before we do that, let us try to understand our Equation 8 a little better.

Question: Can you look at Equation 8 as something arising out of a recursive algorithm. What sort of recursive algorithm would give rise to Equation 8?

Think of a divide and conquer algorithm, which takes a problem of size n, and breaks it into parts of size n/b. Because the size is n/b need not mean the problem got divided into b parts. In fact, our divide and conquer algorithm divides the problem of size n into a different parts, each of size n/b. Then it takes linear time (n) to combine the outputs. The time taken by this algorithm is the $t(n)$ in Equation 8 above.[c]

Though we haven't explicitly mentioned the terminating condition of the recurrence relation, let us say the recursion terminates when the sub problem

size becomes 1. Our algorithm can directly solve a problem of size 1, and it takes exactly unit (1) time to do so.

Question: Draw a rough "picture" for Equation 8. How much detail can you show? What is the recursion depth?

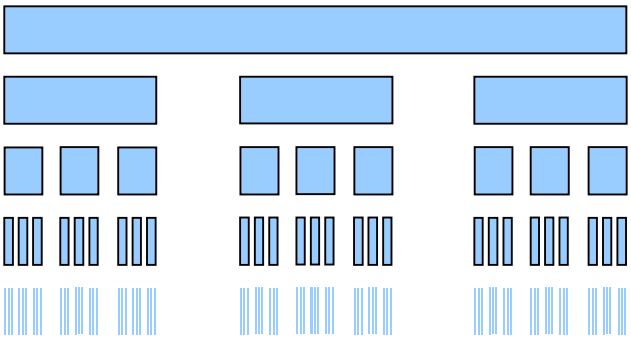


Figure 4: Recursive complexity diagram for Equation 8 for a=3 and b=5

Question: Label the above diagram. How many levels of recursion? How many recursive problems in successive levels of recursion? What is the size of problem at each level of recursion? What is the size of the total blue area?

The size of the recursive problem becomes $1/b^{\text{th}}$ every time you go down a level of recursion. Recursion will stop when the size of problem is one. Therefore it must be obvious that the final level of recursion will be at the depth of $\log_b n$.

Explanation? Here it is:

The size of problem at the topmost level is n. At a level below that, the size of the subproblem is n/b. Each of these n/b sized subproblems will now get divided into n/b^2 sized subproblems of recursion depth 2. At recursion depth k, the size of the subproblem being solved is n/b^k. This recursion will

stop at the depth that the subproblem size it 1, that is, $n/b^k = 1$. When does that happen? When $n = b^k$. So what power of b will give you the value n? That's simple: b when raised to the power of $\log_b n$, will give the value n. That's just the definition of logarithm to the base b. Thus, recursion will stop at the level numbered $\log_b n$.

Thus, the total number of levels of recursion is $(\log_b n)+1$, including the 0th level.

What is the number of subproblems at recursion depth k? That is also simple to answer. At depth zero, there is 1 subproblem. At depth 1, this problem gets divided into a subproblems. At recursion depth 2, each of these a subproblems will get divided into a subproblems, giving a total of a^2 subproblems. Continuing in this fashion, we see that at recursion depth k, the number of subproblems is a^k.

Using this knowledge of number of recursion levels, the number of subproblems at each recursion level, and the size of each subproblem, we can label Figure 4 as shown below.

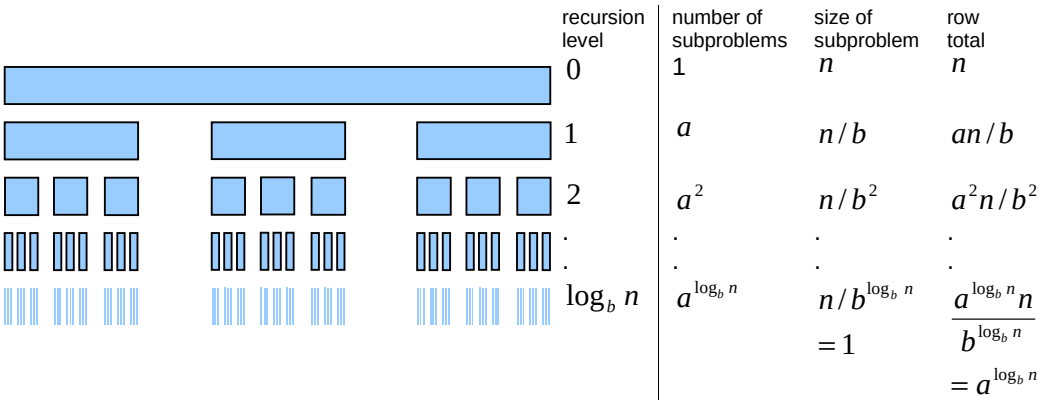


Figure 5: Recursive complexity diagram of Equation 8 with number of subproblems, size of subproblem and the total time spent in each level of recursion

Thus, the total time spent, i.e. the total blue area, i.e. the solution to Equation 8 is

Equation 9

$$t(n) = \underbrace{n + \frac{an}{b} + \frac{a^2 n}{b^2} + \cdots}_{\log_b n \text{ terms}}$$

The above equation is of the same form as Equation 3.

Question: Can Equation 9 be analyzed the same way we analyzed Equation 3? Under what circumstances would the result be "similar"?

Equation 9 is a geometric series with first term n and subsequent terms multiplied with a factor of $r = a/b$ each. We already proved that for a convergent geometric series $1 + r + r^2 + \ldots = 1/(1-r)$. Thus, overestimating the $\log_b n$ terms to infinite terms, we get that

$$t(n) \le n\left(1 + \frac{a}{b} + \frac{a^2}{b^2} + \cdots\right)$$

Equation 10

$$t(n) \le n\left(\frac{1}{1-\frac{a}{b}}\right)$$

$$t(n) \le n\left(\frac{b}{b-a}\right)$$

Equation 10 has given us an upper bound on $t(n)$, which, if we forget the constants, is a linear ($O(n)$) upper bound. Looking at Equation 9 again, it is obvious that $t(n) \ge n$. Thus $t(n)$ is sandwiched between two scales of the linear function n. Thus, $t(n) \in \theta(n)$.

Of course, this will hold only if we have a *convergent* geometric series, i.e. only if $r < 1$, i.e. only if $a/b < 1$, i.e. only if $a < b$.

Thus we have seen that for $a < b$, the solution to

Equation 8

$$t(n) = at\left(\frac{n}{b}\right) + n$$

is

Equation 11

$$t(n) \in \theta(n)$$

This will hold for any a and b, i.e. any number of divisions and any size of divisions, as long as $a < b$. The condition $a < b$ means that all the subproblems of a problem together are not as heavy as the problem itself. The sum of the parts is lesser than the whole!

It is important, though, to remember that there is a hidden constant of $b/(b-a)$ in the θ in Equation 11. This constant may become pretty huge if b and a are close together. The problem that occurs when b is close to a is depicted visually in Figure 6. The blue area is the actual running time of the algorithm, which is a finite geometric series. This is extended to an infinite geometric series by us, shown as the green area. The green area is negligible on the figure on left. But for the figure on the right, where the diminishing rate of the geometric series is not that fast, the green area can add up to quite a lot. Though the $b/(b-a)$ constant is asymptotically negligible, it may be a big overestimate for the first thousands, or millions of n. In such cases, the "rectangle" bound which we will prove in some time could be practically much more useful as an upper bound.

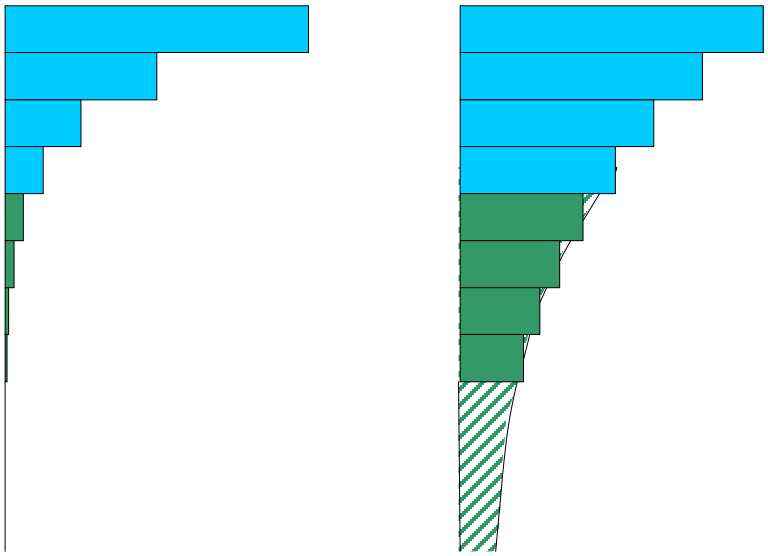


Figure 6: Area estimates given by infinite geometric series extension, for $b = 0.5a$ and $b = 0.8a$.

Ok. So that is the analysis for $a < b$. What happens if $a \not< b$?

Question: What will be the solution to Equation 8 for $a = b$?

We already saw what happens when $a = b = 2$. We get the solution $t(n) \in \theta(n \log n)$. That is because we get the "rectangle" shape of Figure 1. Every level of recursion is as heavy as every other.

A look at Figure 5 will tell us that the same will happen when $a = b =$ some other number. All the levels of recursion will have the same amount of "blue", since $n = an/b = a^2 n/b^2 = \ldots$. What will change is the depth of recursion. Earlier it was $\log_2 n$. Now it is $\log_b n$.

Thus, $\log_b n$ levels, each taking a time of n, gives us the solution $t(n) \approx n \log_b n$.

(This value can also be used as an upper bound for $a < b$, and as we discussed before, it might actually give better bounds than the "asymptotically leaner" $\theta(n)$.)

What can we say about the term $\log_b n$? How does it relate to, say, the term $\log_2 n$?

How does the term $\log_4 n$ relate to the term $\log_2 n$? Let us look at the very basic definitions:

$\log_2 n$ is the number of times you have to divide n by 2, to get to 1.

$\log_4 n$ is the number of times you have to divide n by 4 to get to 1.

Now, dividing by 4 is exactly dividing by 2 *twice*. So, the number of times you will have to divide by 4 is exactly *half* the number of times you will have to divide by 2. Quite clearly, then,

$$\log_4 n = \frac{1}{2}\log_2 n$$

Let us repeat the above for some other division factor b.

Now, how does the term $\log_b n$ relate to the term $\log_2 n$? Let us look at the very basic definitions:

$\log_2 n$ is the number of times you have to divide n by 2 to get to 1.

$\log_b n$ is the number of times you have to divide n by b to get to 1.

(Here comes the tricky part!)

Dividing by b is exactly the same as dividing by 2, repeatedly, $\log_2 b$ number of times. That is because $\underbrace{2\times 2\times 2\ldots}_{\log_2 b} = b$. Thus, the number of times you have to divide by b is exactly $1/\log_2 b$ times the number of times you have to divide by 2. Quite clearly, now,

$$\log_b n = \frac{1}{\log_2 b}\log_2 n$$
$$\log_b n = (\log_b 2)\log_2 n$$

Thus, our earlier result now becomes $t(n) \approx n\log_b n = (\log_b 2) n \log_2 n$. We can say $t(n) \in \theta(n\log n)$. And the next time someone says $\theta(n\log n)$, if you said "yeah, but log to what base?" you'll instantly know why they are laughing at you. The logarithm base is irrelevant because of the θ!

Anyway, we have seen that for $a = b$, the solution to

Equation 8 $$t(n) = at\left(\frac{n}{b}\right) + n$$

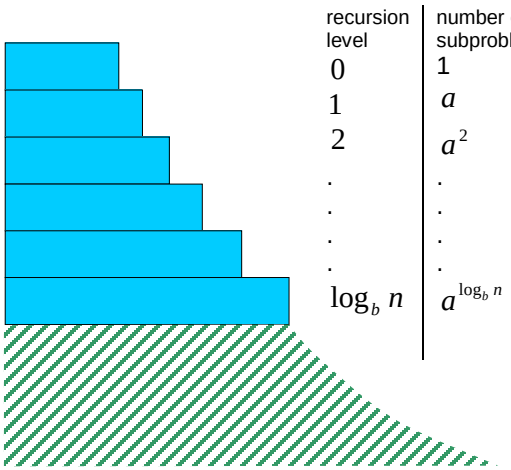
is

Equation 12 $$t(n) \in \theta(n\log n)$$

Question: So what about when $a > b$?

The geometric series $\underbrace{n + an/b + a^2 n/b^2 + \ldots}_{\log_b n \text{ term}}$ is still valid.

The only problem is that now, it is not a convergent geometric series if you extended it to infinite terms. The problem is depicted in Figure 7. The green hatched portion (extended to infinity below) would be our overestimate!

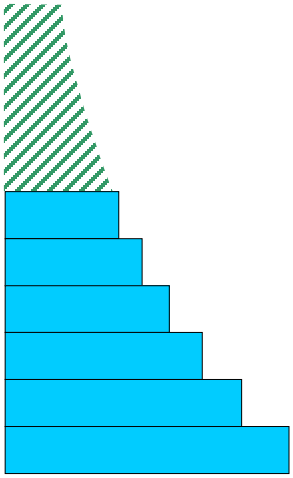


recursion level	number of subproblems	size of subproblem	row total
0	1	n	n
1	a	n/b	an/b
2	a^2	n/b^2	$a^2 n/b^2$
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
$\log_b n$	$a^{\log_b n}$	$n/b^{\log_b n} = 1$	$\frac{a^{\log_b n} n}{b^{\log_b n}} = a^{\log_b n}$

Figure 7: The case where $a > b$

Question: So what do we do?

Look at the blue section in Figure 7 again. It is not a convergent geometric series *downwards*, but it *is* a convergent geometric series *upwards*! Look at the extension in



recursion level	number of subproblems	size of subproblem	row total
0	1	n	n
1	a	n/b	an/b
2	a^2	n/b^2	a^2n/b^2
.	.	.	.
.	.	.	.
$\log_b n$	$a^{\log_b n}$	$n/b^{\log_b n} = 1$	$\frac{a^{\log_b n} n}{b^{\log_b n}} = a^{\log_b n}$

Figure 8: Upward convergence!

Looked at this way, our geometric series starts with the bottommost level, with the value of $a^{\log_b n}$, and moves upward with a multiplicative factor of b/a. Thus, the geometric series is

$$t(n) \le a^{\log_b n}\left(1+\frac{b}{a}+\frac{b^2}{a^2}+\cdots\right)$$

$$t(n) \le a^{\log_b n}\left(\frac{1}{1-\frac{b}{a}}\right)$$

$$t(n) \le a^{\log_b n}\left(\frac{a}{a-b}\right)$$

As before, the lower bound of $a^{\log_b n}$ is obvious, so that $t(n)$ is bounded above and below by scales of $a^{\log_b n}$, giving us that $t(n) \in \theta(a^{\log_b n})$.

Now, we should be pretty happy with this shining analysis of ours, but are we? We are not, you see. Because the question remains, what kind of a function is $a^{\log_b n}$? It's not a logarithm of n. It's not an exponential function of n either. It's an exponential of a logarithm. What does that mean? If the exponential and the logarithm had used the same base, we would have said "log and antilog cancel" and our answer would have been n. But here a and b are different numbers.

Let us consider, as an example $a = 4$ and $b = 2$. What will $4^{\log_2 n}$ be?

Let's go back to the definition of the logarithm.

$\log_2 n$ is the number of times we have to multiply 2 with itself to get n.

So, $4^{\log_2 n}$ is what happens when we find out the number of 2s required to get to n, and multiply those many 4s instead.

(Now comes the usual tricky part!)

Multiplying a 4 is like multiplying a 2 *twice*.

Thus, multiplying as many 4s as you should have multiplied 2s to get to n will get you to n *twice*. Thus, it will get you to n^2. Convince yourself of this: we have seen that

$$4^{\log_2 n} = n^2$$

Now, let us repeat the above procedure for any a and b.

Question: how much is $a^{\log_b n}$? Answer:

Let us go back to the definition of the log:

$\log_b n$ is the number of times we have to multiply b with itself to get to n.

So, $a^{\log_b n}$ is what happens when we find out the number of bs required to get to n, and multiply those many as instead!

But, multiplying with an a is like multiplying with a b $\log_b a$ times.

Thus, multiplying as many as as you should have multiplied bs to get to n, will get you to n (multiplicatively) exactly $\log_b a$ times. Thus, it will get you to $n^{\log_b a}$.

This is great. We know what kind of function is $a^{\log_b n}$! It is a polynomial in n:

$$a^{\log_b n} = n^{\log_b a}$$

If you are not convinced by the (absolutely brilliant) argument above, you should be, by the sequence of steps below:

$$(\log_b n)(\log_b a) = (\log_b a)(\log_b n)$$
$$\log_b (a^{\log_b n}) = \log_b (n^{\log_b a})$$
$$a^{\log_b n} = n^{\log_b a}$$

(The short but "sleigh of hand" sequence above is exactly the same as the "proof" we gave before it. If you have some spare time, try to find the correspondence. Some people like it this way, some people like it that way.)

What we have seen is, when $a > b$, the solution to

Equation 8 $$t(n) = at\left(\frac{n}{b}\right) + n$$

is

Equation 13 $$t(n) \in \theta(n^{\log_b a})$$

(Now, you might be tempted to say that we will write $\theta(n^{\log a})$, since we can forget the logarithm base in the θ notation, but that's not true. Because though the base affects only up to a constant, in $n^{\log_b a}$, the constant is important since it changes the power of n.)

Putting the results we have found above together, our theorem becomes:

The solution to

Equation 8 $$t(n) = at\left(\frac{n}{b}\right) + n$$

is

Equation 14 $$t(n) \in \begin{cases} \theta(n) & \text{when } a < b \\ \theta(n \log n) & \text{when } a = b \\ \theta(n^{\log_b a}) & \text{when } a > b \end{cases}$$

These three cases are depicted in Figure 9.

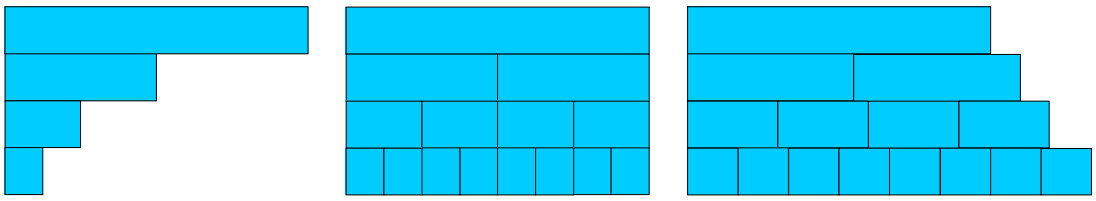


Figure 9: The three cases for Equation 8

Is this the master method? Well, not exactly. It is a little simpler than the master method. (Maybe we should call it the "apprentice method"?) There is one more generalization which will make it the master method. That generalization is going to be using a function $f(n)$ in place of the "$+n$" linear function. We will come to that in a moment.

For now, stare a little at Equation 8, Equation 14 and Figure 9. The three cases are the "geometrically decreasing" $a < b$, the "rectangular" $a = b$ and the "geometrically increasing" $a > b$. In all the cases, the first (topmost) level takes n time, whereas the bottommost level takes $n^{\log_b a}$ time. In the first case, the topmost level is the "indicative" one, giving a running time of $\theta(n)$. In the third case, the bottommost level is the "indicative" one, giving a running time of $\theta(n^{\log_b a})$. In the middle case, *all* levels are equally important, giving a running time of $\theta(n \log n)$.

Question: How do we extend this to any additive function $f(n)$?

The equation we are trying to solve now, is

Equation 15 $$t(n) = at\left(\frac{n}{b}\right) + f(n)$$

Let us take the case $a = 4$ and $b = 2$.

Question: For $a = 4$ and $b = 2$, what will happen if $f(n) = n$?

For $f(n) = n$, using the "apprentice method", we see that we get the "geometrically increasing" case. (The solution will be $\theta(n^{\log_2 4}) = \theta(n^2)$.)

Question: What $f(n)$ should we use to get from the "geometrically increasing" case to the "rectangular" case? Should we use a function "thicker" than n, or "thinner" than n?

There is some chance that you may have confused yourself on the above question. The correct answer is "thicker". Though the "rectangular" case seems thinner than "geometrically increasing", we have to get to it using a thicker $f(n)$. This is because when we use a thicker $f(n)$, it causes the level zero $f(n)$ to grow more than it causes the level one $f(n/b)$ to grow, thus making the hierarchy shape thinner – not really thinner, but relatively thinner.

If that is still too much mush, let us come to the correct answer. The answer is $f(n) = n^2$. Why? Here's why:

We are solving

Equation 16 $$t(n) = 4t\left(\frac{n}{2}\right) + n^2$$

On the zeroth level, the amount of work done will be n^2.

On the first level, the amount of work done will be $(n/2)^2 = n^2/4$. But there will be 4 such pieces, giving total work $4(n/2)^2 = n^2$.

On the second level, the work done will be $4^2(n/2^2)^2 = n^2$.

.
.
.

On the last level, the work done will be $n^{\log_b a} = n^{\log_2 4} = n^2$.

Viola! The rectangular case. And the solution is (almost) $n^2 \log n$. We can call this $f(n)\log n$ or we can call it $n^{\log_b a}\log n$.

Now, in general, when does the rectangular case occur? When the first level and last level have (at least asymptotically) the same amount of stuff. The first level has $f(n)$. The last level has $n^{\log_b a} f(1)$. (There are $a^{\log_b n} = n^{\log_b a}$ divisions, each taking time equaling $f(1)$.) $f(1)$ is just some constant, we will forget it. The rectangular case occurs when $f(n) \in \theta(n^{\log_b a})$ for whatever a and b there are in the recurrence.

Thus, we have seen[9] that, when $f(n) \in \theta(n^{\log_b a})$, the solution to

Equation 15 $$t(n) = at\left(\frac{n}{b}\right) + f(n)$$

is

Equation 17 $$t(n) \in \theta(f(n)\log n)$$

Let us try to get into the "geometrically decreasing" case.

Question: For $a = 4$ and $b = 2$ (as above), what choice of $f(n)$ will get us into the geometrically decreasing case?

[9] With a *lot* of hand-waving!

To get from "geometrically increasing" to "rectangular", we made $f(n)$ thicker, from the linear n to the quadratic n^2. To get into the "geometrically decreasing" case, we have to make $f(n)$ even thicker.

Take for example $f(n) = n^3$. The recurrence

Equation 18

$$t(n) = 4t\left(\frac{n}{2}\right) + n^3$$

will follow the "geometrically decreasing" pattern. The zeroth level will have n^3 stuff, followed by the first level which will have $4(n/2)^3 = n^3/2$, followed by the second level which will have $4^2(n/2^2)^3 = n^3/4$, and so forth. This can be extended to the infinite series $n^3 + n^3/2 + n^3/4 + n^3/8 + \ldots$. This series has a solution which is $\theta(n^3)$.

Now, when will the geometrically decreasing case actually arise? We might be tempted to say "whenever $f(n) > n^2$", but that is obviously not the right answer. (Counter example: $2n^2$!) Then maybe we will say, "Aha! We know! Whenever $f(n) \in \Omega(n^2)$, but *not* in $f(n) \in \theta(n^2)$." Now that would be a very good attempt, but, alas, still not true.

Consider the function $f(n) = n^2 \log n$. Since this is thicker than n^2, we *will* get a "decreasing" pattern, but, unfortunately, not a geometrically decreasing one. It's simple to see. When n changes to $n/2$, n^2 changes to $n^2/4$, but $\log n$ changes only to $(\log n) - 1$. Thus the first level will have $4\left(\frac{n^2}{4}(\log n - 1)\right)$ stuff. Though this is lesser than the zeroth level's $n^2 \log n$, the difference between the two is just n^2, which is not a constant scale of $n^2 \log n$.

Thus, though the $\log n$ factor caused a decreasing pattern, the decrease was not good enough to give a geometric series. We cannot apply the geometrically decreasing case.[10] To apply the geometrically decreasing case, the additive function $f(n)$ has to be *polynomially* thicker than n^2. I.e., $f(n)$ should at least be $n^{2+\varepsilon}$, where ε is some positive constant. Thus, n^3 will do,

[10] We can still use the rectangular case and easily give an *upper* bound of $t(n) \in O(f(n) \log n) = O(n^2 (\log n)^2)$. That much should be obvious, and within the purview of the master method. What may not be obvious is that $n^2(\log n)^2$ is also a lower bound in case, and we have the (asymptotically) exact solution $t(n) \in \theta(n^2 (\log n)^2)$!

$n^{2.5}$ will do, and $n^{2.0000001}$ will do[11], but $n^2 \log n$ will not do, and neither will $n^2 (\log n)^3$.

When we have $f(n) = n^{2+\varepsilon}$, we get the series

$$
\begin{aligned}
& f(n) + af\left(\frac{n}{b}\right) + a^2 f\left(\frac{n}{b^2}\right) + \cdots \\
&= n^{2+\varepsilon} + a\left(\frac{n}{b}\right)^{2+\varepsilon} + a^2\left(\frac{n}{b^2}\right)^{2+\varepsilon} + \cdots \\
&= n^{2+\varepsilon}\left(1 + \frac{a}{b^{2+\varepsilon}} + \frac{a^2}{b^{2(2+\varepsilon)}} + \cdots\right) \\
&= n^{2+\varepsilon} \frac{1}{1 - \frac{a}{b^{2+\varepsilon}}}
\end{aligned}
$$

The last step holds because $a = 4$ and $b = 2$, thus giving a rate of less than one.

Whenever $f(n) = n^{2+\varepsilon}$, we will get the geometrically decreasing pattern, and the solution will be $t(n) \in \theta(f(n))$.

In general, for any a and b, whenever $f(n) \in \Omega(n^{\varepsilon + \log_b a})$, the solution to

Equation 15 $$t(n) = at\left(\frac{n}{b}\right) + f(n)$$

should be

Equation 19 $$t(n) \in \theta(f(n))$$

Unfortunately, it isn't. There are some weird functions which are in the above mentioned Ω, but will not give the above result[12,13].

---

[11] though for $n^{2.0000001}$, the asymptotics will be useful after a huge n, and it would be practically better to use the rectangle bound.

[12] (If you have a polynomial $f(n)$, these weird cases will never arise. But they could easily be your everyday non-polynomial function.)

[13] This happens because the Ω just specifies a lower bound on $f(n)$. These particular $f(n)$s generally "cheat" by growing large fast, and then "holding still" for some time, thus giving a "false impression" about themselves to the θ in the solution. (At least that's how they would behave if they appeared as characters on your favorite soap…)

We need a stronger condition. A stronger condition that will definitely work is easy to derive. You just *say* that the rate of growth r, should be smaller than one.

Now, the first level has $f(n)$ stuff. The second level has $af(n/b)$ stuff. We want that there should exist a constant $r<1$ such that the second level has at most r times the stuff in the first level. I.e., we want an $r<1$ such that

$$af\left(\frac{n}{b}\right) \le rf(n)$$

If such a constant r was found such that the above equation will hold (for that same r) for every n, then every level will have at most r times its previous level, thus giving a geometrically decreasing pattern.

Seems too harsh? OK, how about we exempt the first few ns from having to abide by these rules? (Hey, it's asymptotics, the first few ns don't matter. Right?[d])

Finally what we get is, if $af(n/b) \le rf(n)$ for some constant $r<1$ for all sufficiently large n, the solution to

Equation 15 $$t(n) = at\left(\frac{n}{b}\right) + f(n)$$

is

Equation 19 $$t(n) \in \theta(f(n))$$

The above stronger condition is called the "smoothness" criterion.

The smoothness criterion is not required for the geometrically increasing case. Because the O is considerably stricter with $f(n)$ than Ω is.

Thus, the geometrically increasing case will occur whenever $f(n) \in O(n^{-\varepsilon+\log_b a})$. In this case, the solution will be asymptotically equivalent to the amount of stuff in the last level. The amount of stuff in the last level is $n^{\log_b a} f(1)$.

Thus, if $f(n) \in O(n^{-\varepsilon+\log_b a})$ for some constant $\varepsilon > 0$, the solution to

Equation 15 $$t(n) = at\left(\frac{n}{b}\right) + f(n)$$

is

Equation 20 $$t(n) \in \theta(n^{\log_b a})$$

The above completes the three cases of the master method. If we recollect what we learned in Equation 17, Equation 19 and Equation 20, we get what is known as the "master theorem":

The solution to

Equation 15

$$t(n) = at\left(\frac{n}{b}\right) + f(n)$$

is

Equation 21

$$t(n) \in \begin{cases} \theta(n^{\log_b a}) & \text{when } f(n) \in O(n^{-\varepsilon+\log_b a}) \text{ for some } \varepsilon > 0 \\ \theta(f(n)\log n) & \text{when } f(n) \in \theta(n^{\log_b a}) \\ \theta(f(n)) & \text{when } af(n/b) < rf(n) \text{ for some } r > 0 \end{cases}$$

This is the master theorem, and solving recurrences using this is the master method.

This document was supposed to teach the fundamental workings of the master method, hopefully rendering it more easy to grasp and use than otherwise. Your favorite algorithmics textbook picks up where this document leaves. There they would prove everything correctly, take into account all the borderline cases, the effects added by integer-rounding and be prim and proper about everything. That is generally a good thing. It would be beneficial, at this point to read the book.

---

[a] The inductive step goes

$$\begin{aligned} t(n) &= 2t(n/2) + n \\ &= 2a(n/2)\log(n/2) + n \\ &= an((\log n) - 1) + n \\ &= an\log n - an + n \\ &\le an\log n \end{aligned}$$

which is true for $a = 1$.

[b] You might have even thought that we will have more than one recursion functions, like say, in the recurrence $t(n) = 2t(n/5) + t(n/4) + n$. Though the Master Method does not directly tackle such equations, it does provide upper and lower bounds, and provides us insight into how the recurrence may actually work. (In this recurrence, for example, can you *see* that the answer is going to be $t(n) \in \theta(n)$?)

[c] The recursive algorithm is a nice "realization" for the recurrence relation, which is why we are using it. The analysis we develop here will stand for the recurrence relation itself, whether it arose from a recursive algorithm or not.

[d] Not exactly right, actually.