IV INDUCTION

This is a general purpose proof technique that works in a bottom-up fashion. Knowing that a statement is true for a collection of instances, we argue that it is also true for a new instance, which we then add to the collection. Repeating this step, we establish the statement for a countable collection.

11 Mathematical Induction

In philosophy, *deduction* is the process of taking a general statement and applying it to a specific instance. For example: all students must do homework, and I am a student; therefore, I must do homework. In contrast, *induction* is the process of creating a general statement from observations. For example: all cars I have owned need to be repaired at some point; therefore, all cars will need to be repaired at some point. A similar concept is used in mathematics to prove that a statement is true for all integers. To distinguish it from the less specific philosophical notion, we call it *mathematical induction* of which we will introduce two forms. We begin by considering an example from Section 4, showing that the idea behind Mathematical Induction is a familiar one.

Euclid's Division Theorem. We find the smallest counterexample in order to prove the following theorem.

EUCLID'S DIVISION THEOREM. Letting $n \geq 1$, for every non-negative integer m there are unique integers q and $0 \leq r < n$ such that $m = nq + r$.

PROOF. Assume the opposite, that is, there is a non-negative integer m for which no such q and r exist. We choose the smallest such m. Note that m cannot be smaller than n, else we have $q = 0$ and $r = m$, and m cannot be equal to n, else we have $q = 1$ and $r = 0$. It follows that $m' = m - n$ is a positive integer less than m. Thus, there exist integers q' and $0 \leq r' < n$ such that $m' = nq' + r'$. If we add n on both sides, we obtain $m = (q' + 1)n + r'$. If we take $q = q' + 1$ and $r = r'$, we get $m = nq + r$, with $0 \leq r < n$. Thus, by the Principle of Reduction to Absurdity, such integers q and r exist. □

Let $p(k)$ be the statement that there exist integers q and $0 \leq r < n$ with $k = nq + r$. Then, the above proof can be summarized by

$$p(m-n) \wedge \neg p(m) \implies p(m) \wedge \neg p(m).$$

This is the contradiction that implies $\neg p(m)$ cannot be true. We now focus on the statement $p(m-n) \Rightarrow p(m)$. This is the idea of Mathematical Induction which bypasses the construction of a contradiction.

Example: sum of integers. We consider the familiar problem of summing the first n positive integers. Recall that $\binom{n+1}{2} = \frac{n(n+1)}{2}$.

CLAIM. For all $n \geq 0$, we have $\sum_{i=0}^{n} i = \binom{n+1}{2}$.

PROOF. First, we note that $\sum_{i=0}^{0} i = 0 = \binom{1}{2}$. Now, we assume inductively that for $n > 0$, we have

$$\sum_{i=0}^{n-1} i = \binom{n}{2}.$$

If we add n on both sides, we obtain

$$\begin{aligned} \sum_{i=0}^{n} i &= \binom{n}{2} + n \\ &= \frac{(n-1)n}{2} + \frac{2n}{2} \end{aligned}$$

which is $\frac{(n+1)n}{2} = \binom{n+1}{2}$. Thus, by the Principle of Mathematical Induction,

$$\sum_{i=0}^{n} i = \binom{n+1}{2}$$

for all non-negative integers n. □

To analyze why this proof is correct, we let $p(k)$ be the statement that the claim is true for $n = k$. For $n = 1$ we have $p(1) \wedge [p(1) \Rightarrow p(2)]$. Hence, we get $p(2)$ by Modus Ponens. We can see that this continues:

$$\begin{array}{rlll} p(1) \wedge [p(1) \Rightarrow p(2)] & \text{hence} & p(2); \\ p(2) \wedge [p(2) \Rightarrow p(3)] & \text{hence} & p(3); \\ \cdots & \cdots & \cdots \\ p(n-1) \wedge [p(n-1) \Rightarrow p(n)] & \text{hence} & p(n); \\ \cdots & \cdots & \cdots \end{array}$$

Thus, $p(n_0)$ and $p(n-1) \Rightarrow p(n)$ for all $n > n_0$ implies $p(n)$ for all $n \geq n_0$.

The weak form. We formalize the proof technique into the first, weak form of the principle. The vast majority of applications of Mathematical Induction use this particular form.

MATHEMATICAL INDUCTION (WEAK FORM). If the statement $p(n_0)$ is true, and the statement $p(n-1) \Rightarrow p(n)$ is true for all $n > n_0$, then $p(n)$ is true for all integers $n \geq n_0$.

To write a proof using the weak form of Mathematical Induction, we thus take the following four steps: it should have the following components:

Base Case: $p(n_0)$ is true.

Inductive Hypothesis: $p(n-1)$ is true.

Inductive Step: $p(n-1) \Rightarrow p(n)$.

Inductive Conclusion: $p(n)$ for all $n \geq n_0$.

Very often but not always, the inductive step is the most difficult part of the proof. In practice, we usually sketch the inductive proof, only spelling out the portions that are not obvious.

Example: sum of powers of two. If we can guess the closed form expression for a finite sum, it is often easy to use induction to prove that it is correct, if it is.

CLAIM. For all integers $n \geq 1$, we have $\sum_{i=1}^{n} 2^{i-1} = 2^n - 1$.

PROOF. We prove the claim by the weak form of the Principle of Mathematical Induction. We observe that the equality holds when $n = 1$ because $\sum_{i=1}^{1} 2^{i-1} = 1 = 2^1 - 1$. Assume inductively that the claim holds for $n-1$. We get to n by adding 2^{n-1} on both sides:

$$\begin{aligned} \sum_{i=1}^{n} 2^{i-1} &= \sum_{i=1}^{n-1} 2^{i-1} + 2^{n-1} \\ &= (2^{n-1} - 1) + 2^{n-1} \\ &= 2^n - 1. \end{aligned}$$

Here, we use the inductive assumption to go from the first to the second line. Thus, by the Principle of Mathematical Induction, $\sum_{i=1}^{n} 2^{i-1} = 2^n - 1$ for all $n \geq 1$. □

The strong form. Sometimes it is not enough to use the validity of $p(n-1)$ to derive $p(n)$. Indeed, we have $p(n-2)$ available and $p(n-3)$ and so on. Why not use them?

MATHEMATICAL INDUCTION (STRONG FORM). If the statement $p(n_0)$ is true and the statement $p(n_0) \wedge p(n_0+1) \wedge \cdots \wedge p(n-1) \Rightarrow p(n)$ is true for all $n > n_0$, then $p(n)$ is true for all integers $n \geq n_0$.

Notice that the strong form of the Principle of Mathematical Induction implies the weak form.

Example: prime factor decomposition. We use the strong form to prove that every integer has a decomposition into prime factors.

CLAIM. Every integer $n \geq 2$ is the product of prime numbers.

PROOF. We know that 2 is a prime number and thus also a product of prime numbers. Suppose now that we know that every positive number less than n is a product of prime numbers. Then, if n is a prime number we are done. Otherwise, n is not a prime number. By definition of prime number, we can write it is the product of two smaller positive integers, $n = a \cdot b$. By our supposition, both a and b are products of prime numbers. The product, $a \cdot b$, is obtained by merging the two products, which is again a product of prime numbers. Therefore, by the strong form of the Principle of Mathematical Induction, every integer $n \geq 2$ is a product of prime numbers. □

We have used an even stronger statement before, namely that the decomposition into prime factors is unique. We can use the Reduction to Absurdity to prove uniqueness. Suppose n is the smallest positive integer that has two different decompositions. Let $a \geq 2$ be the smallest prime factor in the two decompositions. It does not belong to the other decomposition, else we could cancel the two occurrences of a and get a smaller integer with two different decompositions. Clearly, $n \bmod a = 0$. Furthermore, $r_i = b_i \mod a \neq 0$ for each prime factor b_i in the other decomposition of n. We have

$$\begin{aligned} n \bmod a &= \left(\prod_i b_i\right) \bmod a \\ &= \left(\prod_i r_i\right) \bmod a. \end{aligned}$$

Since all the r_i are smaller than a and a is a prime number, the latter product can only be zero if one or the r_i is zero. But this contradicts that all the b_i are prime numbers larger than a. We thus conclude that every integer larger than one has a unique decomposition into prime factors.

Summary. Mathematical Induction is a tool to prove that a property is true for all positive integers. We used Modus Ponens to prove the weak as well as the strong form of the Principle of Mathematical Induction.

12 Recursion

We now describe how recurrence relations arise from recursive algorithms, and begin to look at ways of solving them. We have just learned one method that can sometimes be used to solve such a relation, namely Mathematical Induction. In fact, we can think of recursion as backwards induction.

The towers of Hanoi. Recurrence relations naturally arise in the analysis of the towers of Hanoi problem. Here we have three pegs, A, B, C, and initially n disks at A, sorted from large to small; see Figure 10. The task is to move the n disks from A to C, one by one, without ever placing a larger disk onto a smaller disk. The following

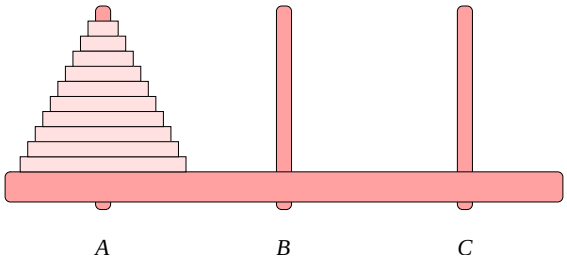


Figure 10: We have a sorted stack of disks at A and use B for temporary storage to move one disk at a time to C. We need B to avoid any inversions among the disks.

three steps solve this problem:

- recursively move $n-1$ disks from A to B;
- move the n-th disk from A to C;
- recursively move $n-1$ disks from B to C.

When we move disks from one peg to another, we use the third peg to help. For the main task, we use B to help. For the first step, we exchange the roles of B and C, and for the third step, we exchange the roles of A and B. The number of moves is given by the solution to the recurrence relation

$$M(n) \quad = \quad 2M(n-1)+1,$$

with initial condition $M(0)=0$. We may use induction to show that $M(n)=2^n-1$.

Loan payments. Another example in which recurrence relations naturally arise is the repayment of loans. This is an iterative process in which we alternate the payment of a constant sum with the accumulation of interest. The iteration ends when the entire loan is payed off. Suppose A_0 is the initial amount of your loan, m is your monthly payment, and p is the annual interest payment rate. What is the amount you owe after n months? We can express it in terms of the amount owed after $n-1$ months:

$$T(n) \quad = \quad \left(1+\frac{p}{12}\right)T(n-1)-m.$$

This is a recurrence relation, and figuring out how much you owe is the same as solving the recurrence relation. The number that we are most interested in is $n_0 m$, where n_0 is the number of months it takes to get $T(n_0)=0$. Instead of attacking this question directly, let us look at a more abstract, mathematical setting.

Iterating the recursion. Consider the following recurrence relation,

$$T(n) \quad = \quad rT(n-1)+a,$$

where r and a are some fixed real numbers. For example, we could set $r=1+\frac{p}{12}$ and $a=-m$ to get the recurrence that describes how much money you owe. After replacing $T(n)$ by $rT(n-1)+a$, we may take another step and replace $T(n-1)$ by $rT(n-2)+a$ to get $T(n)=r(rT(n-2)+a)+a$. Iterating like this, we get

$$\begin{aligned} T(n) &= rT(n-1)+a \\ &= r^2T(n-2)+ra+a \\ &= r^3T(n-3)+r^2a+ra+a \\ \cdots &\quad \cdots \\ &= r^nT(0)+a\sum_{i=0}^{n-1} r^i. \end{aligned}$$

The first term on the right hand side is easy, namely r^n times the initial condition, say $T(0)=b$. The second term is a sum, which we now turn into a nicer form.

Geometric series. The sequence of terms inside a sum of the form $\sum_{i=0}^{n-1} r^i$ is referred to as a *geometric series*. If $r=1$ then this sum is equal to n. To find a similarly easy expression for other values of r, we expand both the sum and its r-fold multiple:

$$\begin{aligned} \sum_{i=0}^{n-1} r^i &= r^0+r^1+r^2+\ldots+r^{n-1}; \\ r\sum_{i=0}^{n-1} r^i &= r^1+r^2+\ldots+r^{n-1}+r^n. \end{aligned}$$

Subtracting the second line from the first, we get

$$(1-r)\sum_{i=0}^{n-1} r^i \quad = \quad r^0 - r^n$$

and therefore $\sum_{i=0}^{n-1} r^i = \frac{1-r^n}{1-r}$. Now, this allows us to rewrite the solution to the recurrence as

$$T(n) \quad = \quad r^n b + a\frac{1-r^n}{1-r},$$

where $b = T(0)$ and $r \neq 1$. Let us consider the possible scenarios:

`Case 1.` $r = 0$. Then, $T(n) = a$ for all n.

`Case 2.` $0 < r < 1$. Then, $\lim_{n\to\infty} r^n = 0$. Therefore, $\lim_{n\to\infty} T(n) = \frac{a}{1-r}$.

`Case 3.` $r > 1$. The factors r^n of b and $\frac{r^n-1}{r-1}$ of a both grow with growing n. For positive values of a and b, we can expect $T(n) = 0$ for a negative value of n. Multiplying with $r-1$, we get $r^n b(r-1)+ar^n-a = 0$ or, equivalently, $r^n(br - b + a) = a$. Dividing by $br - b + a$, we get $r^n = \frac{a}{br-b+a}$, and taking the logarithm to the base r, we get

$$n \quad = \quad \log_r\left(\frac{a}{br-b+a}\right).$$

For positive values of a and b, we take the logarithm of a positive number smaller than one. The solution is a negative number n.

We note that the loan example falls into Case 3, with $r = 1 + \frac{p}{12} > 1$, $b = A_0$, and $a = -m$. Hence, we are now in a position to find out after how many months it takes to pay back the loan, namely

$$n_0 \quad = \quad \log_r\left(\frac{m}{m - A_0\frac{p}{12}}\right).$$

This number is well defined as long as $m > A_0\frac{p}{12}$, which means your monthly payment should exceed the monthly interest payment. It better happen, else the amount you owe grows and the day in which the loan will be payed off will never arrive.

First-order linear recurrences. The above is an example of a more general class of recurrence relations, namely the *first-order linear recurrences* that are of the form

$$T(n) \quad = \quad f(n)T(n-1) + g(n).$$

For the constant function $f(n) = r$, we have

$$\begin{aligned} T(n) \quad &= \quad r^n T(0) + \sum_{i=0}^{n-1} r^i g(n-i) \\ &= \quad r^n T(0) + \sum_{i=0}^{n-1} r^{n-i} g(i). \end{aligned}$$

We see that if $g(n) = a$, then we have the recurrence we used above. We consider the example $T(n) = 2T(n-1) + n$ in which $r = 2$ and $g(i) = i$. Hence,

$$\begin{aligned} T(n) \quad &= \quad 2^n T(0) + \sum_{i=0}^{n-1} \frac{i}{2^{n-i}} \\ &= \quad 2^n T(0) + \frac{1}{2^n}\sum_{i=0}^{n-1} i2^i. \end{aligned}$$

It is not difficult to find a closed form expression for the sum. Indeed, it is the special case for $x = 2$ of the following result.

CLAIM. For $x \neq 1$, we have

$$\sum_{i=1}^{n} ix^i \quad = \quad \frac{nx^{n+2} - (n-1)x^{n+1} + x}{(1-x)^2}.$$

PROOF. One way to prove the relation is by induction. Writing $R(n)$ for the right hand side of the relation, we have $R(1) = x$, which shows that the claimed relation holds for $n = 1$. To make the step from $n-1$ to n, we need to show that $R(n-1) + x^n = R(n)$. It takes but a few algebraic manipulations to show that this is indeed the case. □

Summary. Today, we introduced recurrence relations. To find the solution, we often have to define $T(n)$ in terms of $T(n_0)$ rather than $T(n-1)$. We also saw that different recurrences can have the same general form. Knowing this will help us to solve new recurrences that are similar to others that we have already seen.

13 Growth Rates

How does the time to iterate through a recursive algorithm grow with the size of the input? We answer this question for two algorithms, one for searching and the other for sorting. In both case, we find the answer by solving a recurrence relation.

Binary Search. We begin by considering a familiar algorithm, binary search. Suppose we have a sorted array, $A[1..n]$, and we wish to find a particular item, x. Starting in the middle, we ask whether $x = A[(n+1)/2]$? If it is, we are done. If not, we have cut the problem in half. We give a more detailed description in pseudo-code.

```
l = 1; r = n;
while l ≤ r do m = (l + r)/2;
  if x = A[m] then print(m); exit
    elseif x < A[m] then r = m − 1;
    elseif x > A[m] then l = m + 1
  endif
endwhile.
```

Assuming $n = 2^k - 1$, there are $2^{k-1} - 1$ items to the left and to the right of the middle. Let $T(n)$ be the number of times we check whether $l \leq r$. We check once at the beginning, for $n = 2^k - 1$ items, and then some number of times for half the items. In total, we have

$$T(n) = \begin{cases} T(\frac{n-1}{2}) + 1 & \text{if } n \geq 2; \\ 1 & \text{if } n = 1. \end{cases}$$

In each iteration, k decreases by one and we get $T(n) = k+1$. Since $k = \log_2(n+1)$, this gives $T(n) = 1+\log_2 n$. We could verify this by induction.

A similar recurrence relation. Let us consider another example, without specific algorithm. Suppose we solve a problem of size n by first solving one problem of size $n/2$ and then doing n units of additional work. Assuming n is a power of 2, we get the following recurrence relation:

$$T(n) = \begin{cases} T(\frac{n}{2}) + n & \text{if } n \geq 2; \\ 0 & \text{if } n = 1. \end{cases} \quad (1)$$

Figure 11 visualizes the computation by drawing a node for each level of the recursion. Even though the sequence of nodes forms a path, we call this the *recursion tree* of the computation. The problem size decreases by a factor of two from one level to the next. After dividing $\log_2 n$ times, we arrive at size one. This implies that there are only $1 + \log_2 n$ levels. Similarly, the work at each level decreases by a factor of two from one level to the next. Assuming $n = 2^k$, we get

$$\begin{aligned} T(n) &= n + \frac{n}{2} + \ldots + 2 + 1 \\ &= 2^k + 2^{k-1} + \ldots + 2^1 + 2^0 \\ &= 2^{k+1} - 1. \end{aligned}$$

Hence, $T(n) = 2n - 1$.

level	#nodes	size		work per node	work per level
1	1	n		n	n
2	1	$n/2$		$n/2$	$n/2$
3	1	$n/4$		$n/4$	$n/4$
4	1	$n/8$		$n/8$	$n/8$

Figure 11: The recursion tree for the relation in Equation (1).

Merge Sort. Next, we consider the problem of sorting a list of n items. We assume the items are stored in unsorted order in an array $A[1..n]$. The list is sorted if it consists of only one item. If there are two or more items then we sort the first $n/2$ items and the last $n/2$ items and finally merge the two sorted lists. We provide the pseudo-code below. We call the function with $\ell = 1$ and $r = n$.

```
void MergeSort(ℓ, r)
  if ℓ < r then m = (ℓ + r)/2;
    MergeSort(ℓ, m);
    MergeSort(m + 1, r);
    Merge(ℓ, m, r)
  endif.
```

We merge the two sorted lists by scanning them from left to right, using n comparisons. It is convenient to relocate both lists from A to another array, B, and to add a so-called stopper after each sublist. These are items that are larger than all given items. In other words, we assume the two lists are stored in $B[\ell..m]$ and $B[m+2..r+1]$, with $B[m+1] = B[r+2] = \infty$. When we scan the two lists, we move the items back to A, one at a time.

```
void MERGE(ℓ, m, r)
  i = ℓ; j = m + 2;
  for k = ℓ to r do
    if B[i] < B[j] then A[k] = B[i]; i = i + 1;
                   else A[k] = B[j]; j = j + 1
    endif
  endfor.
```

Assume $n = 2^k$ so that the sublists are always of the same length. The total number of comparisons is then

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + n & \text{if } n \geq 2; \\ 0 & \text{if } n = 1. \end{cases}$$

To analyze this recurrence, we look at its recursion tree.

Recursion Tree. We begin with a list of length n, from which we create two shorter lists of length $n/2$ each. After sorting the shorter lists recursively, we use n comparisons to merge them. In Figure 12, we show how much work is done on the first four levels of the recursion. In this example, there are n units of work per level, and $1 + \log_2 n$ levels in the tree. Thus, sorting with the merge sort algorithm takes $T(n) = n \log_2 n + n$ comparisons. You can verify this using Mathematical Induction.

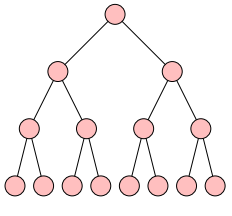
level	#nodes	size		work per node	work per level
1	1	n		n	n
2	2	$n/2$		$n/2$	n
3	4	$n/4$		$n/4$	n
4	8	$n/8$		$n/8$	n

Figure 12: The recursion tree for the merge sort algorithm.

Unifying the findings. We have seen several examples today, and we now generalize what we have found.

CLAIM. Let $a \geq 1$ be an integer and d a non-negative real number. Let $T(n)$ be defined for integers that are powers of 2 by

$$T(n) = \begin{cases} aT(\frac{n}{2}) + n & \text{if } n \geq 2; \\ d & \text{if } n = 1. \end{cases}$$

Then we have the following:

- $T(n) = \Theta(n)$ if $a < 2$;
- $T(n) = \Theta(n \log n)$ if $a = 2$;
- $T(n) = \Theta(n^{\log_2 a})$ if $a > 2$.

In the next lecture, we will generalize this result further so it includes our finding that binary search takes only a logarithmic number of comparisons. We will also see a justification of the three cases.

Summary. Today, we looked at growth rates. We saw that binary search grows logarithmically with respect to the input size, and merge sort grows at a rate of order $n \log_2 n$. We also discovered a pattern in a class recurrence relations.

14 Solving Recurrence Relations

Solving recurrence relations is a difficult business and there is no catch all method. However, many relations arising in practice are simple and can be solved with moderate effort.

A few functions. A solution to a recurrence relation is generally given in terms of a function, eg. $f(n) = n \log_2 n$, or a class of similar functions, eg. $T(n) = \mathrm{O}(n \log_2 n)$. It is therefore useful to get a feeling for some of the most common functions that occur. By plotting the graphs, as in Figure 13, we get an initial picture. Here we see a sequence of progressively faster growing functions: constant, logarithmic, linear, and exponential. However,

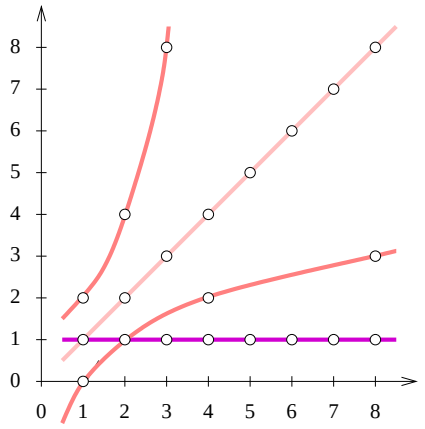


Figure 13: The graphs of a small set of functions, $f(x) = 1$, $f(x) = \log_2 x$, $f(x) = x$, $f(x) = 2^x$.

such plots can be confusing because they depend on the scale. For example, the exponential function, $f(x) = 2^x$, grows a lot faster than the quadratic function, $f(x) = x^2$, but this would not be obvious if we only look at a small portion of the plane like in Figure 13.

Three regimes. In a recurrence relation, we distinguish between the *homogeneous* part, the recursive terms, and the *inhomogeneous* part, the work that occurs. The solution of depends on the relative size of the two, exhibiting qualitatively different behavior if one dominates the other or the two are in balance. Recurrence relations that exhibit this three-regime behavior are so common that it seems worthwhile to study this behavior in more detail. We summarize the findings.

MASTER THEOREM. Let $a \geq 1$ and $b > 1$ be integers and $c \geq 0$ and $d > 0$ real numbers. Let $T(n)$ be defined for integers that are powers of b by

$$T(n) \quad = \quad \begin{cases} aT(\frac{n}{b}) + n^c & \text{if} \quad n > 1 \\ d & \text{if} \quad n = 1. \end{cases}$$

Then we have the following:

- $T(n) = \Theta(n^c)$ if $\log_b a < c$;
- $T(n) = \Theta(n^c \log n)$ if $\log_b a = c$;
- $T(n) = \Theta(n^{\log_b a})$ if $\log_b a > c$.

This behavior can be explained by recalling the formula for a geometric series, $(r^0 + \ldots + r^{n-1})A = \frac{1-r^n}{1-r}A$, and focusing on the magnitude of the constant factor, r. For $0 < r < 1$, the sum is roughly A, the first term, for $r = 1$, the sum is n, the number of terms, and for $r > 1$, the sum is roughly $r^{n-1}A$, the last term.

Let us consider again the recursion tree and, in particular, the total work at its i-th level, starting with $i = 0$ at the root. There are a^i nodes and the work at each node is $(\frac{n}{b^i})^c$. The work at the i-th level is therefore

$$a^i \left(\frac{n}{b^i}\right)^c \quad = \quad n^c \frac{a^i}{b^{ic}}.$$

There are $1 + \log_b n$ levels, and the total work is the sum over the levels. This sum is a geometric series, with factor $r = \frac{a}{b^c}$. It is therefore dominated by the first term if $r < 1$, all terms are the same if $r = 0$, and it is dominated by the last term if $r > 1$. To distinguish between the three cases, we take the logarithm of r, which is negative, zero, positive if $r < 1$, $r = 1$, $r > 1$. It is convenient to take the logarithm to the basis b. This way we get

$$\begin{aligned} \log_b \frac{a}{b^c} \quad &= \quad \log_b a - \log_b b^c \\ &= \quad \log_b a - c. \end{aligned}$$

We have $r < 1$ iff $\log_b a < c$, In which case the dominating term in the series is n^c. We have $r = 1$ iff $\log_b a = c$, in which case the total work is $n^c \log_b n$. We have $r > 1$ iff $\log_b a > c$, in which case the dominating term is $d \cdot a^{\log_b n} = d \cdot n^{\log_b a}$. This explains the three cases in the theorem.

There are extensions of this result that discuss the cases in which n is not a lower of b, we have floors and ceilings in the relation, a and b are not integers, etc. The general behavior of the solution remains the same.

Using induction. Once we know (or feel) what the solution to a recurrence relation is, we can often use induction to verify. Here is a particular relation defined for integers that are powers of 4:

$$T(n) \quad = \quad \begin{cases} T(\frac{n}{2}) + T(\frac{n}{4}) + n & \text{if} \quad n > 1 \\ 1 & \text{if} \quad n = 1. \end{cases}$$

To get a feeling for the solution, we group nodes with equal work together. We get n once, $\frac{n}{2}$ once, $\frac{n}{4}$ twice, $\frac{n}{8}$ three times, $\frac{n}{16}$ five times, etc. These are the Fibonacci numbers, which grow exponentially, with basis equal to the golden ratio, which is roughly 1.6. On the other hand, the work shrinks exponentially, with basis 2. Hence, we have a geometric series with factor roughly 0.8, which is less than one. The dominating term is therefore the first, and we would guess that the solution is some constant times n. We can prove this by induction.

CLAIM. There exists a positive constant c such that $T(n) \leq cn$.

PROOF. For $n = 1$, we have $T(1) = 1$. Hence, the claimed inequality is true provided $c \geq 1$. Using the strong form of Mathematical Induction, we get

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + n \\ &= c\frac{n}{2} + c\frac{n}{4} + n \\ &= \left(\frac{3c}{4} + 1\right) n. \end{aligned}$$

This is at most cn provided $\frac{3c}{4} + 1 \leq c$ or, equivalently, $c \geq 4$. □

The inductive proof not only verified what we thought might be the case, but it also gave us the smallest constant, $c = 4$, for which $T(n) \leq cn$ is true.

Finding the median. Similar recurrence relations arise in practice. A classic example is an algorithm for finding the k-smallest of an unsorted set of n items. We assume the items are all different. A particularly interesting case is the middle item, which is called the *median*. For odd n, this is the k-smallest with $k = \frac{n+1}{2}$. For even n, we set k equal to either the floor or the ceiling of $\frac{n+1}{2}$. The algorithm takes four steps to find the k-smallest item.

STEP 1. Partition the set into groups of size 5 and find the median in each group.

STEP 2. Find the median of the medians.

STEP 3. Split the set into S, the items smaller than the median of the medians, and L, the items larger than the median of the medians.

STEP 4. Let $s = |S|$. If $s < k-1$ then return the $(k-s)$-smallest item in L. If $s = k-1$ then return the median of the medians. if $s > k-1$ then return the k-smallest item in S.

The algorithm is recursive, computing the median of roughly $\frac{n}{5}$ medians in Step 2, and then computing an item either in L or in S. To prove that the algorithm terminates, we need to show that the sets considered recursively get strictly smaller. This is easy as long as n is large but tricky for small n. We ignore these difficulties.

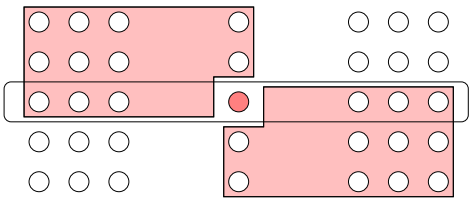


Figure 14: The upper left shaded region consists of items smaller than the median of the medians. Symmetrically, the lower right shaded region consists of items larger than the median of the medians. Both contain about three tenth of all items.

To get a handle on the running time, we need to estimate how much smaller than n the sets S and L are. Consider Figure 14. In one iteration of the algorithm, we eliminate either all items smaller or all items larger than the median of the medians. The number of such items is at least the number in one of the two shaded regions, each containing roughly $\frac{3n}{10}$ items. Hence, the recurrence relation describing the running time of the algorithm is

$$T(n) \quad = \quad \begin{cases} T(\frac{7n}{10}) + T(\frac{n}{5}) + n & \text{if} \quad n > n_0 \\ n_0 & \text{if} \quad n \leq n_0, \end{cases}$$

for some large enough constant n_0. Since $\frac{7}{10} + \frac{1}{5}$ is strictly less than one, we guess that the solution to this recurrence relation is again $\mathrm{O}(n)$. This can be verified by induction.

Fourth Homework Assignment

Write the solution to each problem on a single page. The deadline for handing in solutions is 18 March 2009.

Question 1. ($20 = 10 + 10$ points).

(a) Prove the following claim:

$$1 + 7 + \cdots + (3n^2 - 3n + 1) = n^3.$$

(b) (Problem 4.1-11 in our textbook). Find the error in the following proof that all positive integers n are equal. Let $p(n)$ be the statement that all numbers in an n-element set of positive integers are equal. Then $p(1)$ is true. Let $n \geq 2$ and write N for the set of n first positive integers. Let N' and N'' be the sets of $n-1$ first and $n-1$ last integers in N. By $p(n-1)$, all members of N' are equal, and all members of N'' are equal. Thus, the first $n-1$ elements of N are equal and the last $n-1$ elements of N are equal, and so all elements of N are equal. Therefore, all positive integers are equal.

Question 2. (20 points). Recall the Chinese Remainder Theorem stated for two positive, relatively prime moduli, m and n, in Section 7. Assuming this theorem, prove the following generalization by induction on k.

CLAIM. Let $n_1, n_2, \ldots, n_k$ be positive, pairwise relative prime numbers. Then for every sequence of integers $a_i \in \mathbb{Z}_{n_i}$, $1 \leq i \leq k$, the system of k linear equations,

$$x \bmod n_i \quad = \quad a_i,$$

has a unique solution in $\mathbb{Z}_N$, where $N = \prod_{i=1}^{k} n_i$.

Question 3. ($20 = 10 + 10$ points).

(a) (Problem 4.2-13 in our textbook). Solve the recurrence $T(n) = 2T(n-1) + 3^n$, with $T(0) = 1$.

(b) (Problem 4.2-17 in our textbook). Solve the recurrence $T(n) = rT(n-1)+n$, with $T(0) = 1$. (Assume that $r \neq 1$.)

Question 4. ($20 = 7 + 7 + 6$ points). Consider the following algorithm segment.

```
int FUNCTION(n)
  if n > 0 then
    n = FUNCTION(⌊n/a⌋) + FUNCTION(⌊n/b⌋)
  endif
  return n.
```

We can assume that $a, b > 1$, so the algorithm terminates. In the following questions, let a_n be the number of iterations of the `while` loop.

(a) Find a recurrence relation for a_n.

(b) Find an explicit formula for a_n.

(c) How fast does n grow? (big Θ terms)

Question 5. ($20 = 4+4+4+4+4$ points). (Problem 4.4-1 in our textbook). Use the Master Theorem to solve the following recurrence relations. For each, assume $T(1) = 1$ and n is a power of the appropriate integer.

(a) $T(n) = 8T(\frac{n}{2}) + n$.

(b) $T(n) = 8T(\frac{n}{2}) + n^3$.

(c) $T(n) = 3T(\frac{n}{2}) + n$.

(d) $T(n) = T(\frac{n}{4}) + 1$.

(e) $T(n) = 3T(\frac{n}{3}) + n^2$.