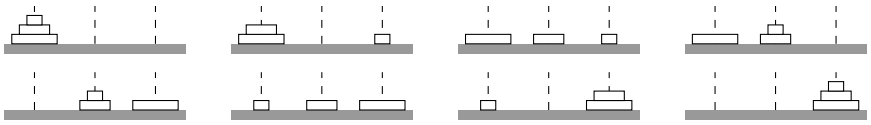
Recurrences are formulas that describe the value of a function of positive integers at an input in terms of the value of the same function at smaller inputs. We will see examples of problems in which recurrences come about and show how to solve them.

1 Towers of Hanoi

You have three posts and a stack of n disks of different sizes, stacked up from largest to smallest, on the first post.

The objective is to move the stack on to the third post, one disk at a time, so that a larger disk is never placed on top of a smaller one. Here is a way to do it for three disks:



What about four disks? You can try doing some experiments, but you might find the process quite involved. Instead, let us think inductively and use our solution for 3 disks as a "subroutine":

More generally, suppose we have solved the Towers of Hanoi problem for n disks. Here is how to solve it for $n+1$ disks: Ignore the largest disk and apply the solution for n disks to move the remaining tower to the middle pole. Then move the largest disk to the rightmost pole. Ignore the largest disk again and the apply the solution for n disks to move the remaining tower to the rightmost pole.

Let $T(n)$ be the number of moves we performed to move n disks. Then $T(n)$ satisfies the equation

$$T(n+1) = 2T(n) + 1. \tag{1}$$

Here, each of the two $T(n)$s accounts for the steps taken in each of the inductive moves applied to the tower of n smaller blocks, and the extra one is for the move of the largest block from the left pole to the right pole.

This type of equation is a *recurrence*. If we know $T(1)$, it allows us to calculate $T(2)$, $T(3)$, and so on. In our case, $T(1) = 1$ since a one block tower can be rearranged in one move.

2 Solving recurrences

One objective is to understand how a recurrence like (1) behaves for large values of n. The best thing to do is to get a closed-form formula for $T(n)$, if we can.

To do this I recommend you work backwards. From equation (1) we get

$$\begin{aligned} T(n) &= 2T(n-1)+1 \\ &= 2(2T(n-2)+1)+1 = 2^2T(n-2)+(2+1) \\ &= 2^2(2T(n-3)+1)+(2+1) = 2^3T(n-3)+(2^2+2+1). \end{aligned}$$

You start to spot a pattern: If we continue this process for enough steps until we get a $T(1)$ on the right hand side – this is $n-1$ steps – we get

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

Since $T(1)=1$, it seems that

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

using the formula for geometric series from last lecture.

We can indeed verify that this guess is correct:

Theorem 1. *The assignment $T(n)=2^n-1$ satisfies equation (1) for all $n \geq 1$.*

Proof. Assume $T(n)=2^n-1$ and $n \geq 1$. Then

$$2T(n)+1 = 2\cdot(2^n-1)+1 = 2^{n+1}-2+1 = 2^{n+1}-1 = T(n+1). \qquad \square$$

Let's do another example. Recall the Beneš network B_n from Lecture 6. How many edges does the digraph B_n have? Recall that B_n was constructed recursively as follows. B_1 is a network with two sources, two sinks, and all four possible directed edges between them. The network B_n was constructed by taking two disjoint copies of B_{n-1}, adding 2^n new sources, 2^n new sinks, and two new incoming/outgoing edges for each source/sink. The number of edges $E(n)$ of B_n is given by the recurrence

$$\begin{aligned} E(n) &= 2E(n-1)+2^{n+2} \\ E(1) &= 4. \end{aligned} \tag{2}$$

Let us try to solve this recurrence by working backwards:

$$\begin{aligned} E(n) &= 2E(n-1)+2^{n+2} \\ &= 2(2E(n-2)+2^{n+1})+2^{n+2} = 2^2E(n-2)+2\cdot 2^{n+2} \\ &= 2^2(2E(n-3)+2^n)+2\cdot 2^{n+2} = 2^3E(n-3)+3\cdot 2^{n+2} \end{aligned}$$

Continuing this reasoning for $n-1$ steps we obtain the guess

$$E(n) = 2^{n-1}E(1)+(n-1)\cdot 2^{n+2} = 2^{n-1}\cdot 4+(n-1)\cdot 2^{n+2} = n\cdot 2^{n+2}-2^{n+1}.$$

We can now verify that our guess is correct:

Theorem 2. *For all $n \geq 1$, the digraph B_n has $E(n) = n\cdot 2^{n+2}-2^{n+1}$ edges.*

Proof. We prove the theorem by induction on n. The digraph B_1 has $4 = 1\cdot 2^3-2^2$ edges as desired. Now assume B_n has $n\cdot 2^{n+2}-2^{n+1}$ vertices. By recurrence (1), B_{n+1} then has

$$2\cdot(n\cdot 2^{n+2}-2^{n+1})+2^{n+3} = n2^{n+3}+2^{n+2} = (n+1)2^{n+3}-2^{n+2}$$

edges as desired. $\square$

3 Merge sort

Merge sort is the following procedure for sorting a sequence of n numbers in nondecreasing order:

Input: A sequence of n numbers.
Merge Sort Procedure:
If $n = 1$, do nothing. Otherwise,
Recursively sort the left half of the sequence.
Recursively sort the right half of the sequence.
Merge the two sorted sequences in increasing order.

For example, if the input sequence is

10 7 15 3 6 8 1 9

the sequence is split in the first half 10 7 15 3 and the second half 6 8 1 9. Each half is sorted recursively to obtain

3 7 10 15 and 1 6 8 9

Then the two sequences are merged. To merge the first and second half, we compare the leftmost numbers in both sequences, take out the smaller of the two and write it as the next term in the output sequence until both halves become empty.

first half	second half	output
3 7 10 15	**1** 6 8 9	
3 7 10 15	6 8 9	1
7 10 15	**6** 8 9	1 3
7 10 15	8 9	1 3 6
10 15	**8** 9	1 3 6 7
10 15	**9**	1 3 6 7 8
10 15		1 3 6 7 8 9
15		1 3 6 7 8 9 10
		1 3 6 7 8 9 10 15

For example, in the first line, the leftmost numbers in the first and the second half are 1 and 3, respectively. They are compared to each other, and since 1 is smaller, it is taken out and written in the output.

In this example, exactly seven pairwise comparisons were made in the merging phase. In general, if the length of the sequence is n, the number of comparisons when merging the two halves is $n-1$.

We want to count the total number $C(n)$ of comparisons that Merge Sort performs when sorting a sequence of n numbers. We will assume that n is a power of two so that any time the sequence is split in half in a recursive call the two halves are equal.

To sort n numbers (for $n > 1$), Merge Sort makes two recursive calls to sequences of length $n/2$ and performs exactly $n - 1$ comparisons in the merging phase. This gives the recurrence

$$C(n) = 2C(n/2) + (n - 1) \qquad \text{for } n > 1 \tag{3}$$

with the base case $C(1) = 0$.

Let's try to guess a solution for this recurrence by working backwards as usual:

$$\begin{aligned} C(n) &= 2C(n/2) + (n-1) \\ &= 2(2C(n/2^2) + (n/2-1)) + (n-1) = 2^2C(n/2^2) + 2n - (2+1) \\ &= 2^2(2C(n/2^3) + n/2^2 - 1) + 2n - (2+1) = 2^3C(n/2^3) + 3n - (2^2+2+1). \end{aligned}$$

We reach $C(1)$ on the right hand side after $\log n$ steps.[1] The expression we get on the right is

$$nC(1) + (\log n)n - (2^{\log n - 1} + \cdots + 2 + 1).$$

By our base case, $C(1) = 0$. By the formula for geometric sums, the last term equals $2^{\log n} - 1 = n - 1$. This suggests the guess

$$C(n) = n \log n - n + 1.$$

This formula indeed sets $C(1)$ to zero as desired. You can now verify on your own that this guess solves the recursion by a calculation.

Theorem 3. *The assignment $C(n) = n \log n - n + 1$ satisfies the equations (3).*

4 Homogeneous linear recurrences

You are given an unlimited supply of 1×1 and 2×1 tiles. In how many ways can you tile a hallway of dimension $n \times 1$ using the tiles? For example, here are all five possible tilings for $n = 4$:

Let $f(n)$ denote the number of desired tilings of an $n \times 1$ hallway. If $n = 0$ there is exactly one possible tiling — use no tiles — so $f(0) = 1$. If $n = 1$ there is also one tiling — use a single 1×1 tile — so $f(1) = 1$.

When $n \geq 2$, we distinguish two possible types of tilings. If the tiling starts with a 1×1 tile, then the remaining part of the corridor can be tiled in $f(n-1)$ ways. If the tiling starts with a 2×1 tile, then there are $f(n-2)$ possible tilings. Since these two possibilities cover each possibility exactly once, we obtain the recurrence

$$f(n) = f(n-1) + f(n-2) \qquad \text{for every } n \geq 2. \tag{4}$$

This is an example of a *homogeneous linear recurrence*. Such recurrences can be solved using the following guess verify method.

Initially, we forget about the "base cases" $f(0) = 1, f(1) = 1$ and focus on the equations (4). We look for solutions of the type $f(n) = x^n$ for some nonzero real number x. The reason we expect such solutions to work out is because for every $n \geq 2$, the equation $x^n = x^{n-1} + x^{n-2}$ is the same as

$$x^2 = x + 1$$

since all we did was scale down both sides by a factor of x^{n-2}. This is a quadratic equation in x and we can use the quadratic formula to calculate its solutions

$$x_1 = \frac{1+\sqrt{5}}{2} \qquad \text{and} \qquad x_2 = \frac{1-\sqrt{5}}{2}.$$

[1] In computer science, we use log for base two logarithms.

It is easy to check that both $f(n) = x_1^n$ and $f(n) = x_2^n$ satisfy all the equations (4). But what about the requirements $f(0) = 1$ and $f(1) = 1$? Neither solution satisfies these additional conditions.

Now we notice that every *linear combination* $f(n) = sx_1^n + tx_2^n$ also solves the equations (4), so if we can find numbers s and t such that

$$1 = f(0) = sx_1^0 + tx_2^0 = s + t$$

and

$$1 = f(1) = sx_1^1 + tx_2^1 = s \cdot \frac{1+\sqrt{5}}{2} + t \cdot \frac{1-\sqrt{5}}{2}$$

then the assignment $f(n) = sx_1^n + tx_2^n$ must be the solution to our problem.

To find s and t, we need to solve two equations with two unknowns. These equations are simple enough that we can solve them by hand; no need for the computer. After subtracting the first equation scaled by $1/2$ from the second one we get

$$\frac{1}{2} = \frac{\sqrt{5}}{2}(s - t)$$

or $s - t = 1/\sqrt{5}$. Adding this to $s + t = 1$ we get $2s = 1 + 1/\sqrt{5}$, from where

$$s = \frac{1}{2} + \frac{1}{2\sqrt{5}} = \frac{1}{\sqrt{5}} \cdot \frac{1+\sqrt{5}}{2}$$

and

$$t = 1 - s = -\frac{1}{\sqrt{5}} \cdot \frac{1-\sqrt{5}}{2}$$

from where

$$f(n) = sx_1^n + tx_2^n = \frac{1}{\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \frac{1}{\sqrt{5}} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}. \tag{5}$$

You can now verify in the usual way that this solution is indeed correct.

Theorem 4. *The assignment (5) satisfies the equations (4) and the equations $f(0) = 1$, $f(1) = 1$.*

The number $f(n)$ is called the *n-th Fibonacci number*. The formula (5) tells us how these numbers behave when n is large. Since $(1 + \sqrt{5})/2 \approx 1.618$ and $(1 - \sqrt{5})/2 \approx -0.618$, the term $((1 + \sqrt{5})/2)^n$ will grow when n becomes large and the term $((1 - \sqrt{5})/2)^n$ will become vanishingly small; eventually, it becomes smaller than any fixed constant. Therefore we can write

$$f(n) = \frac{1}{\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^{n+1} + o(1).$$

For an even rougher estimate, we can disregard all the constants in the first term to get

$$f(n) = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right).$$

In general, this method can be used to solve any homogeneous linear recurrence, that is a recurrence of the type

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots + a_k f(n-k)$$

as long as k "base cases" are provided. If you can't figure out the procedure, please see Section 22.3.2 of the textbook.

The recurrence that counts the number of moves for the Towers of Hanoi

$$T(n) = 2T(n-1) + 1$$

is *not* homogeneous because of the extra +1 term on the right hand side. It can be turned into a homogeneous recurrence like this: If I add another +1 to both sides I can rewrite this equation as

$$T(n) + 1 = 2(T(n-1) + 1)$$

Now we can introduce the "change of variables" $T'(n) = T(n) + 1$ and get the homogeneous recurrence

$$T'(n) = 2T'(n-1)$$

which is then easy to solve.

5 Divide-and-conquer recurrences

Divide-and-conquer is a paradigm for solving problems that works like this: We split a problem of a large size into a fixed number of subproblems of a fraction of its size and combine their solutions in some manner to obtain a solution to the bigger problem. Merge Sort is an example of a divide-and-conquer algorithm.

In general, if a problem of size n is split into a_1 subproblems of size $b_1 n$, a_2 subproblems of size $b_2 n$, up to a_k subproblems of size $b_k n$ (where $b_1, \ldots, b_k$ are constants between 0 and 1) the recurrences that arise in its analysis have the form

$$T(n) = a_1 T(b_1 n) + \cdots + a_k T(b_k n) + g(n). \tag{6}$$

Here, $g(n)$ describes the "complexity" of combining the solutions to the subproblems. For example, in Merge-Sort we split a problem of size n into two problems of size $n/2$ and combined their solutions using $n-1$ comparisons to obtain the recurrence $C(n) = 2C(n/2) + n - 1$.

There is a general formula that tells us the *asymptotic behavior* of the function $T(n)$ that satisfies the recurrence (6), assuming that the function $g(n)$ is "sufficiently nice." (We'll explain precisely what this means shortly.) The asymptotic behaviour of T is given by the formula

$$T(n) = \Theta\left(n^p\left(1 + \int_1^n \frac{g(u)}{u^{p+1}}\, du\right)\right)$$

where p is the unique constant such that $a_1 b_1^p + \cdots + a_k b_k^p = 1$.

For example, in the Merge Sort recurrence, $k = 1$, $a_1 = 2$ and $b_1 = 1/2$ so p is the number that solves the equation $2 \cdot (1/2)^p = 1$, namely $p = 1$. Then

$$\int_1^n \frac{u-1}{u^{p+1}} = \int_1^n \frac{u-1}{u^2} = \ln u + \frac{1}{u}\Big|_1^n = \ln n - 1 + \frac{1}{n}$$

The dominant term here is $\ln n$, so $C(n) = \Theta(n \log n)$, as we already know.

Let's do another example. The following recurrence arises in the analysis of the running time of a certain algorithm for multiplying matrices:

$$M(n) = 7M(n/2) + \Theta(n^2).$$

In this case, we do not even know what the function g is precisely, but only that it grows asymptotically as n^2. The value p needs to satisfy $7 \cdot (1/2)^p = 1$, so $p = \log_2 7$. Let us now calculate the relevant integral, first assuming that $g(x)$ is *exactly* equal to x^2:

$$\int_1^n \frac{u^2}{u^{p+1}}\, du = \int_1^n u^{1-\log_2 7}\, du = O(1)$$

because the exponent is $1 - \log_2 7 \approx -1.807$, which is smaller than minus one, and so the integral converges. (You can come to the same conclusion by calculating the integral.)

The actual integral $\int_1^n g(u)/u^{p+1}\, du$ may not have the exact same value, but since $g(u)$ is within a constant factor of u^2 (when u is sufficiently large), so will be the two integrals and we can conclude that $\int_1^n g(u)/u^{p+1}\, du = O(1)$. Therefore

$$M(n) = \Theta\left(n^p\left(1 + \int_1^n \frac{g(u)}{u^{p+1}}\, du\right)\right) = \Theta(n^p) = \Theta(n^{\log_2 7}).$$

Here is a precise statement of the theorem that allows us to obtain the asymptotic behaviour of solutions to a large number of divide-and-conquer recurrences.

Theorem 5. *Assume the function $T(x)$ is nonnegative and bounded when $0 \le x \le x_0$ and satisfies the recurrence*

$$T(x) = a_1 T(b_1 x) + \cdots + a_k T(b_k x) + g(x)$$

for $x > 0$ for some positive constants $a_1, \ldots, a_k$ and constants $b_1, \ldots, b_k$ between zero and one.

If g is non-negative and $|dg(x)/dx|$ is $O(x^C)$ for some constant C, then

$$T(n) = \Theta\left(n^p\left(1 + \int_1^n \frac{g(u)}{u^{p+1}}\, du\right)\right)$$

where p is the unique constant such that $a_1 b_1^p + \cdots + a_k b_k^p = 1$.

In the Merge Sort example, $g(x) = x - 1$ so $|dg(x)/dx| = 1$ and our use of Theorem 5 is legitimate. In the other example, we can say $M(n) = \Theta(n^{\log_2 7})$ assuming that the function represented by $\Theta(n^2)$ in the definition of M has the absolute value of its derivative bounded by a polynomial.

The textbook states a more general version of the theorem accounting for the possibility that the subproblem sizes might not be exactly $b_1 n, \ldots, b_k n$ but perhaps deviate a bit from these values. Instead of showing the general version, let me just say that when this deviation is sufficiently small (e.g. a constant) then the theorem continues to hold.

6 Expanders and superconcentrators*

The Beneš networks B_n is for $N = 2^n$ packets with $(n+1)2^{n+2} = \Theta(N \log N)$ edges. If we measure the "cost" of a switching network by the number of edges, this is a large improvement over the complete bipartite network in which each source-sink pair is connected for a total of N^2 edges when N grows large. Can we do even better and design a switching network for N packets with $o(N \log N)$ edges when N is large? In the next homework you will show that this is impossible. It is, however, possible to construct a related type of digraph that we describe next.

Definition 6. A *superconcentrator* for N packets is a digraph with N sources and N sinks in which for every set S of sources and every set T of sinks of the same size, which we call k, there exists a collection of k vertex-disjoint paths from the vertices in S to the vertices in T.

This sounds a lot like our definition of a switching network. What is the difference? A switching network can realize a collection of vertex-disjoint paths for *every possible* pairing of sources and sinks, while the superconcentrator merely guarantees the *existence* of a pairing for which vertex-disjoint paths exist. For example, if the sources are Alice, Bob, and Charlie and the sinks are Dave, Eve, and Faye, a superconcentrator guarantees that there exist vertex-disjoint paths from Alice and Bob to Dave and Faye, but not necessarily that Alice can be paired up with Dave and Bob with Faye. If we think of Alice, Bob, and Charlie as customers and Dave, Eve, and Faye as service representatives, then any k customers can get to talk to any k service representatives that happen to be at work, but they do not get to choose who talks to whom.

The distinction between switching networks and superconcentrators sounds technical. However, superconcentrators can have substantially fewer edges when the number of packets is large.

Theorem 7. *There exist a superconcentrator S_N for N packets with $O(N)$ edges for every N.*

The proof of Theorem 7 makes use of an important type of graph called an expander. We will call a bipartite graph with vertex partition (L, R) an *expander* if for every subset S of L of size at most $|L|/2$, the set of neighbors of S is at least as large as S, namely $|N(S)| \geq |S|$.[2]

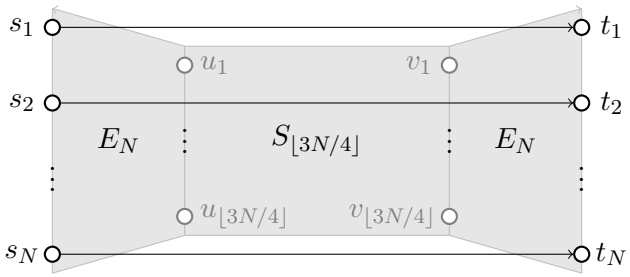
Lemma 8. *For every $N > 64$ there exists an expander E_N with $|L| = N$, $|R| = \lfloor 3N/4 \rfloor$ in which all vertices in L have degree (at most) 32.*

In particular, the expander E_N has (at most) $32N$ edges. The proof of Lemma 8 is not too difficult but is beyond the scope of this course. It turns out that if every vertex in L is connected with 32 vertices in R independently at random the resulting graph is an expander with good probability.

Armed with Lemma 8 we can now prove Theorem 7. If $N \leq 64$ then we can take the superconcentrator S_N to be the Beneš network B_6, discarding some sources and sinks if necessary. This is clearly a superconcentrator.

If $N > 64$, we construct the digraphs S_N recursively as follows:

1. Take N sources $s_1, \ldots, s_N$, N sinks $t_1, \ldots, t_N$, and two sets of internal vertices $u_1, \ldots, u_{\lfloor 3N/4 \rfloor}$ and $v_1, \ldots, v_{\lfloor 3N/4 \rfloor}$.
2. Add edges between the sets $L = \{s_1, \ldots, s_n\}$ and $R = \{u_1, \ldots, u_{\lfloor 3N/4 \rfloor}\}$ as in the expander E_N from Lemma 8. Direct all edges from L to R.
3. Add edges between the sets $L' = \{t_1, \ldots, t_n\}$ and $R' = \{v_1, \ldots, v_{\lfloor 3N/4 \rfloor}\}$ as in the expander E_N from Lemma 8. Direct all edges from R' to L'.
4. Add an arbitrary perfect matching between L and L'.
5. Include a copy of $S_{\lfloor 3N/4 \rfloor}$, identifying its sources and sinks with the vertices in R and R', respectively.



[2]The actual definition is in fact more general and involves several parameters; if you are interested see the reference.

The next claim says that the construction is indeed correct:

Claim 9. *For every N, the graph S_N is a superconcentrator.*

Proof. The proof is by strong induction on N. For the base case(s) $N \leq 64$, S_N is a switching network, so it is in particular a superconcentrator. For the inductive step, we will assume that $S_{\lfloor 3N/4 \rfloor}$ is a superconcentrator and prove that S_N must also be one.

The proof is by cases depending on the size k of the sets S and T in the definition of superconcentrators. Let us first consider the case $k \leq N/2$. By the definition of expanders, every subset S' of S has at least $|S'|$ neighbors in R. By Hall's theorem, there exists a matching Ξ that matches the vertices in S to some subset U consisting of k vertices in R using edges from the first copy of E_N. For the same reason, the vertices in T can be matched to some subset V consisting of k vertices in R' in the second copy of E_N via a matching Ξ'. Because $S_{\lfloor 3N/4 \rfloor}$ is a superconcentrator there must exist a collection Π of k vertex-disjoint paths from U to V in $S_{\lfloor 3N/4 \rfloor}$. The vertex-disjoint paths from S to T can then be obtained by taking the union of Ξ, Π, and Ξ'. This concludes the proof for the case $k \leq N/2$.

If $k > N/2$, there must exist at least $2k - N$ indices i for which both s_i is in S and t_i is in T. (This can be proved easily using the inclusion-exclusion principle, which we will introduce in Lecture 10.) The matching from step 5 in the construction of S_N gives disjoint paths of length 1 between these $2k - N$ sources and sinks. After discarding these from the digraph S_N, there remain at most $k - (2k - N) = N - k$ sources and sinks to match. This number is at most $N/2$, so by the analysis of the case $k \leq N/2$ there exist a vertex-disjoint collection of paths in S_N among the remaining sources and sinks. This concludes the proof for the case $k > N/2$ and the inductive step. □

To finish the proof of Theorem 7 it remains to upper bound the number of edges $E(N)$ in the digraph S_N. By construction, when $N \geq 64$ the number of edges in S_N equals the number of edges in $S_{\lfloor 3N/4 \rfloor}$, plusthe number of edges in each of the two copies of E_N ($32N$ each), plus N edges for the matching in step 5. Adding the contribution of all these terms, we obtain the recurrence

$$E(N) = E(\lfloor 3N/4 \rfloor) + 65N \qquad \text{when } N > 64$$

with $E(N) \leq 3,328$ for $N \leq 64$ (by Theorem 2). We can therefore bound $E(N)$ by

$$E(N) \leq (65 \cdot N + 65 \cdot (3/4)N + 65 \cdot (3/4)^2 N + \cdots) + 3,328 = 260N + 3,328$$

for all N.

References

This lecture is based on Chapter 22 of the text *Mathematics for Computer Science* by E. Lehman, T. Leighton, and A. Meyer. Theorem 5 was discovered by Mohammad Akra and Louay Bazzi in 1996. You can read more about it and find its proof in these notes of Leighton. Section 6 is based on Chapter 1 of the survey *Expander graphs and their applications* by Hoory, Linial, and Wigderson.