Recursive Algorithms

NAVNEET GOYAL

Background

1. Recursion
2. Defining Sequences, Functions, and Sets using recursion

Recursive Algorithms

1. Fibonacci Sequence
2. Factorial
3. Linear Search
4. Binary Search
5. Merge-sort (& Merging of 2 sorted lists)
6. Towers of Hanoi

Recursive Algorithms

Today we will try to identify recursion in Towers of Hanoi problem:

- Given 3 pegs and 64 different sized disks sorted on peg 1 such that the smallest (largest) disk is on top (bottom)

Q: How fast can we move the disks from peg 1 to peg 2 such that

(1) We can move only one disk at a time.

(2) No larger disk can be on top of a smaller one.

- Let H_n be the minimum # of moves required to move n disk from peg i to peg j, i != j

Recursive Algorithms

Please verify that:

$H_1=1$

$H_2=3$

$H_3=7$

$H_4=$???

…

$H_n=$???

Recursive Algorithms

Observation:

Observe that, in order to move n disks from peg 1 to peg 2, we must somehow move the largest disk from peg 1 to peg 2 (1 move) after first moving the smaller n-1 disks from peg 1 to peg 3 (H_{n-1} moves). Once the largest disk is moved to peg 2, we can then move the n-1 disks from peg 3 to peg 2 (H_{n-1} moves)

$$H_1 = 1,$$
$$H_n = H_{n-1} + 1 + H_{n-1}$$
$$= 2H_{n-1} + 1, n > 1.$$

Verify that $H_n = 2^n - 1$, for n>1

Recursive Algorithms

$$
\begin{aligned}
H_n &= 2H_{n-1} + 1 \\
&= 2(2H_{n-2} + 1) + 1 \\
&= 2^2H_{n-2} + 2 + 1 \\
&= 2^2(2H_{n-3} + 1) + 2 + 1 \\
&= 2^3H_{n-3} + 2^2 + 2^1 + 2^0 \\
&= \ldots \\
&= 2^{n-1}H_1 + 2^{n-2} + 2^{n-3} + \ldots + 2^1 + 2^0 \\
&= 2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 2^1 + 2^0 \\
&= 2^n - 1
\end{aligned}
$$

Recursive Algorithms

In the original Tower of Hanoi problem, n = 64.

Hence,

$H_n = 2^{64} - 1$

= 18,446,744,073,709,551,615

If we can move one disk per second, it will still take us more than _____ years!

Recursive Algorithms

Plane Division Problem:

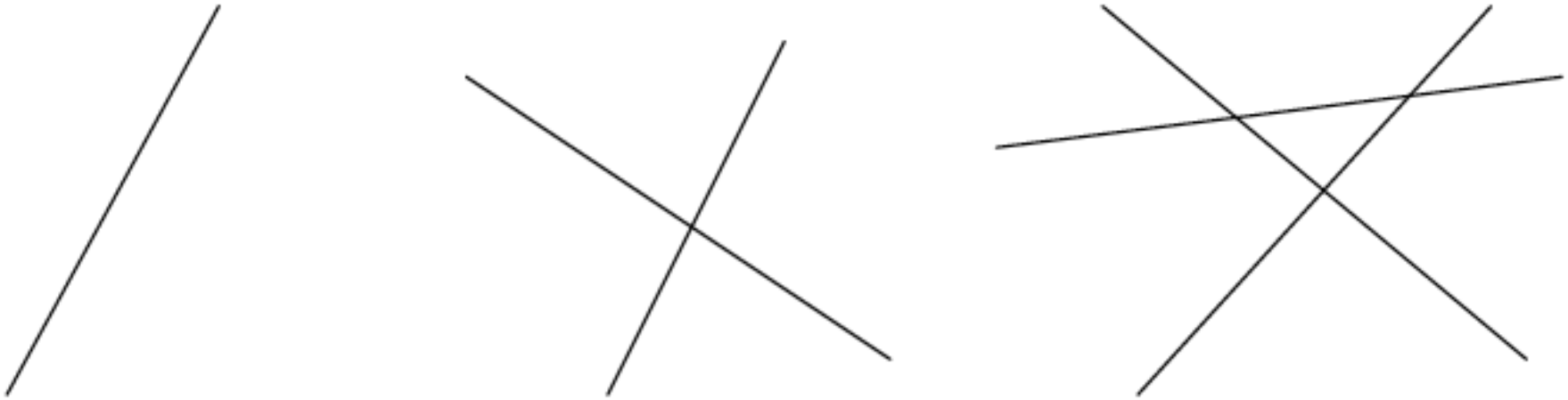
Q: How many regions can a plane be divided by n straight lines such that:

(1) Every pair of lines must intersect.

(2) No more than 2 lines can intersect at a common point.

Recursive Algorithms

Plane Division Problem:



Please verify:

lines: 1 2 3 4 5 6 7 8 …

regions: 2 4 7 11 16 22 29 37 …

Recursive Algorithms

Plane Division Problem:

Q: How many regions can a plane be divided by n straight lines such that:

(1) Every pair of lines must intersect.

(2) No more than 2 lines can intersect at a common point.

Recursive Algorithms

Observations:

1. The 2nd line interests the first line to yield 2 more regions.

2. The 3rd line interests the first 2 lines to yield 3 more regions.

3. The 4th line interests the first 3 lines to yield 4 more regions.

Q: Do you see the pattern?

Recursive Algorithms

In general, the nth line interests the first n-1 lines to yield n more regions!

Let R_n be the number of regions divided by n lines.

Observe that

$$R_1 = 2,$$

$$R_n = R_{n-1} + n,\ n > 1.$$

Recursive Algorithms

$$
\begin{aligned}
R_n &= R_{n-1} + n \\
&= R_{n-2} + (n-1) + n \\
&= R_{n-3} + (n-2) + (n-1) + n \\
&= \ldots \\
&= R_1 + 2 + 3 + \ldots + (n-2) + (n-1) + n \\
&= [1 + 2 \ldots + (n-1) + n] + 1 \\
&= [n(n+1)/2] + 1
\end{aligned}
$$

Recursive Algorithms

Observations:

1. The 2nd line interests the first line to yield 2 more regions.

2. The 3rd line interests the first 2 lines to yield 3 more regions.

3. The 4th line interests the first 3 lines to yield 4 more regions.

Q: Do you see the pattern?

Recursive Algorithms

The Rabbit Problem:

Given a pair of new-born rabbits, we assume that, after two months, this pair of rabbits will give birth to another pair of rabbits, and this pair of new rabbits will then go through the same cycle again.

For simplicity, we assume that the rabbits will never die and this process will go on indefinitely.

Q: How many pairs of rabbits do we have after n months?

Recursive Algorithms

Let an be the number of rabbits we have after n months.

Observe that

$a_0 = 0$,

$a_1 = 1$,

$a_2 = 1$, (before breeding starts)

$a_n = a_{n-1} + a_{n-2}$, $n > 2$.

(a_{n-2} = # of pairs of new-born rabbits)

Recursive Algorithms

Let an be the number of rabbits we have after n months

Example:

Month: 0 1 2 3 4 5 6 7 …

Pair Rabbits: 0 1 1 2 3 5 8 13 …

Observe that an is simply the nth term Fibonacci sequence number.

Q: How do you compute a_n?

Recursive Algorithms: Problems

1. How many bit strings of length “n” we can form which will not have 2 consecutive zeroes?
2. A computer system considers a string of decimal digits a valid codeword if it contains an even number of 0 digits. For instance, 1230407869 is valid, whereas 120987045608 is not valid. Let a_n be the number of valid n-digit codewords. Find a recurrence relation for a_n.

Recursive Algorithms: Problems

a) Find a recurrence relation for the number of ternary strings of length n that do not contain two consecutive 0s.

b) What are the initial conditions?

c) How many bit strings of length 6 do not contain two consecutive 0s?

Recursive Algorithms: Problems

a) Find a recurrence relation for the number of ternary strings

of length n that do not contain two consecutive 0s or two consecutive 1s.

b) What are the initial conditions?

c) How many bit strings of length 6 do not contain two

consecutive 0s two consecutive 1s?

DO it Yourself!!

Recursive Algorithms: Problems

Let an be the number of ternary strings of length n that do not contain two consecutive O's or two consecutive l's.

start with a 2 and follow with a string of length n - 1 not containing two consecutive O's or two consecutive l's,

or

start with 02 or 12 and follow with a string of length n - 2 not containing two consecutive O's or two consecutive l's,

or

start with 012 or 102 and follow with a string of length n - 3 not containing two consecutive O's or two consecutive l's,

or

start with 0102 or 1012 and follow with a string of length n - 4 not containing two consecutive O's or two consecutive l's, and so on…

Recursive Algorithms: Problems

Once we encounter a 2, we can, in effect, start fresh, but the first 2 may not appear for a long time.

Before the first 2 there are always two possibilities-the sequence must alternate between 0's and 1's, starting with either a 0 or a 1.

Furthermore, there is one more possibility-that the sequence contains no 2's at all, and there are two cases

in which this can happen: 0101 ... and 1010 ....

Putting this all together we can write down the recurrence relation, valid for all n>=2:

$a_n = a_{n-1} + 2a_{n-2} + 2a_{n-3} + 2a_{n-4} + \ldots + 2a_o + 2$

Recursive Algorithms: Problems

It turns out that the sequence also satisfies the recurrence relation a,, = $2a_{n-1} + a_{n-2}$, which can be derived algebraically from the recurrence relation we just gave by subtracting the recurrence for a_{n-1} from

the recurrence for a_n.

Can you find a direct argument for it?

Recurrence Relations

Many recursive algorithms take a problem with a given input and divide it into one or more smaller problems.

Reduction is successively applied until the solutions of the smaller problems can be found quickly.

Recurrence Relations

BINARY SEARCH

We perform a binary search by reducing the search for an element in a list to the search for this element in a list half as long. We successively apply this reduction until one element is left.

MERGE SORT

When we sort a list of integers using the merge sort, we split the list into two halves of equal size and sort each half separately. We then merge the two sorted halves.

Recurrence Relations

These procedures follow an important algorithmic paradigm known as **divide-and-conquer**, and are called **divide-and-conquer algorithms**, because they divide a problem into one or more instances of the same problem of smaller size and they conquer the problem by using the solutions of the smaller problems to find a solution of the original problem, perhaps with some additional work.

“Divide et impera” - Julius Caesar

Divide-and-Conquer Recurrence Relations

Suppose that a recursive algorithm divides a problem of size n into “a” sub-problems, where each sub-problem is of size n/b (for simplicity, assume that n is a multiple of b; in reality, the smaller problems are often of size equal to the nearest integers either less than or equal to, or greater than or equal to, n/b).

Divide-and-Conquer Recurrence Relations

Also, suppose that a total of g(n) extra operations are required in the conquer step of the algorithm to combine the solutions of the sub problems into a solution of the original problem.

Then, if f (n) represents the number of operations required to solve the problem of size n, it follows that f satisfies the recurrence relation

$f(n) = a\,f(n/b) + g(n)$.

This is called a **divide-and-conquer recurrence relation**.

Divide-and-Conquer Recurrence Relations

Binary search algorithm reduces the search for an element in a search sequence of size n to the binary search for this element in a search sequence of size n/2, when n is even.

Hence, the problem of size n has been reduced to one problem of size n/2

Two comparisons are needed to implement this reduction (one to determine which half of the list to use and the other to determine whether any terms of the list remain).

Hence, if f (n) is the number of comparisons required to search for an element in a search sequence of size n, then

$f(n) = f(n/2) + 2$, when n is even.

Divide-and-Conquer Recurrence Relations

Maximum and Minimum elements of a sequence a1, a2, . . . , an. If n = 1, then a1 is the maximum and the minimum.

If n > 1, split the sequence into two sequences, either where both have the same number of elements or where one of the sequences has one more element than the other.

The problem is reduced to finding the maximum and minimum of each of the two smaller sequences.

The solution to the original problem results from the comparison of the separate maxima and minima of the two smaller sequences to obtain the overall maximum and minimum.

Divide-and-Conquer Recurrence Relations

Let f (n) be the total number of comparisons needed to find the maximum and minimum elements of the sequence with n elements.

We have shown that a problem of size n can be reduced into two problems of size n/2, when n is even, using two comparisons, one to compare the maxima of the two sequences and the other to compare the minima of the two sequences.

f (n) = 2f (n/2) + 2, when n is even.