UNIT 1 RECURRENCE RELATIONS

Structure   **Page No.**

1.0   INTRODUCTION

In the previous block, you have learnt to solve various types of combinatorial problems using a varied set of tools. However, there are many problems that have to do with counting which cannot be tackled only with the techniques we have presented so far. To give you one such example, look at the problem of counting the number of binary strings of length n that do not contain two consecutive zeros.

If we denote the number of such binary strings by a_n, then $a_1 = 2$ and $a_2 = 3$. Also, we can show that $a_n = a_{n-1} + a_{n-2}$ for $n \geq 2$. This is an example of a **recurrence relation.** This relates the value of a_n, a_{n-1} and a_{n-2}. We will show how to find an explicit expression for a_n using this relation in units 2 and 3. In this unit, we will discuss how to **formulate** such recurrence relations for solving combinational problems. In Sec. 1.2, we will introduce you to recurrence relations through three famous examples, the Fibonacci recurrence, Towers of Hanoi and the number of ways of parenthesising an expression.

In Sec. 1.3, we will discuss some more examples to familiarise you with the process of formulating recurrences.

In Sec. 1.4, we will formally define a recurrence relation and explain some terminology related to recurrences like order and degree of a recurrence relation.

In the Sec. 1.5, we will see how the Divide and Conquer techniques used in the design of algorithms give rise to recurrences in a natural way. Here, we will discuss recurrences associated with algorithms for finding the maximum and minimum elements of a list, fast multiplication of integers etc.

1.1   OBJECTIVES

After going through this unit, you should be able to
- define a recurrence relation;
- give examples of recurrence relations;
- set up recurrence relations;
- write recurrences for divide and conquer algorithms.

1.2 THREE RECURRENT PROBLEMS

Let us begin by exploring three sample problems that will give you an idea of what is to follow. The first two of these problems have been investigated repeatedly. All the three problems have a solution based on the idea of **recurrences**. This means that the solution to each problem depends on the solution to smaller instances of the same problem.

Example:1 (Rabbits and the Fibonacci numbers): Have you heard of the problem of breeding rabbits? This was originally posed by **Leonardo di Pisa,** also known as **Fibonacci,** in 1202 in his book **Liber abaci.** The problem is the following: One pair of rabbits, one male and one female, are left on an island. These rabbits begin breeding at the end of two months and produce a pair of rabbits of opposite sex at the end of each month thereafter. Let f_n denote the number of pairs of rabbits after n months. Then $f_1 = 1$. Note that the rabbits start breeding only after two months and the young ones will be produced one month afterwards. So, young ones are produced only at the end of third month. Therefore the number of pairs of rabbits is still 1 at the end of the second month i.e $f_2 = 1$. At the end of the third month, the pair would have produced one more pair. See Table 1 for details. To find the number of pairs after n months, we must add the number of pairs after $n - 1$ months to the number of pairs born in the nth month. But the newborns come from pairs at least two months old, i.e. from the pairs that already existed after $n - 2$ months; there are f_{n-2} of these. Therefore the sequence $\{f_n \mid n \geq 1\}$ meets the condition $f_n = f_{n-1} + f_{n-2}$ if $n \geq 3$, and the f_n are called **Fibonacci numbers**.

**Fibonacci
(1170—1250)**

Table 1: Number of Rabbits on the Island

Months	Reproducing pairs (at least two months old)	Young Pairs (not more than two months old.)
1		
2		
3		
4		
5		
6		

* * *

So, have we solved the problem? Not quite; but it uniquely defines the sequence we seek, describing its members in terms of some previous members. We can also define f_n as a function of n, as in the following exercise.

E1) Using induction, verify that $\sqrt{5}\, f_n = \left(\dfrac{1+\sqrt{5}}{2}\right)^n - \left(\dfrac{1-\sqrt{5}}{2}\right)^n$, $n \geq 1$.

We will come back to the Fibonacci sequence later. Now let us consider another important recurrent problem.

Example 2: **(The Tower of Hanoi):** This problem was invented by the French mathematician **Edouard Lucas** in 1883. We are given a tower of eight discs, initially stacked in decreasing size on one of three pegs.
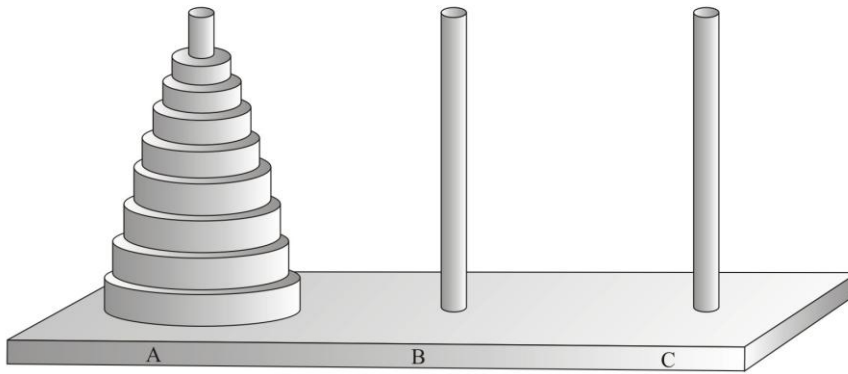
Fig. 2 : Initial position for the towers of Hanoi problem.

The objective is to transfer the entire tower to one of three pegs, moving only one disc at a time without ever moving a larger disc onto a smaller one. Lucas furnished this toy with a legend about a much larger **Tower of Brahma,** which supposedly had 64 discs of pure gold resting on three diamond needles. "At the beginning of time", he said, "God placed these golden diamond needles", "God placed these golden discs on the first needle and said that a group of priests should transfer them to a third, according to the rules above. The Tower will crumble and the world will come to an end once the task is finished."

Let us generalise the problem and see what happens if we have n discs instead. Let us say that T_n is the minimum number of moves that will transfer n disks from one peg to another under the rules. Clearly, $T_1 = 1$, and $T_2 = 3$ (see Fig.3). A little bit of experimentation on three disks leads us to the general strategy: We first transfer the $n-1$ smallest disks to B (requiring T_{n-1} moves), then move the largest (requiring one move; remember, it must move!) to C. A is now empty and can be used to transfer the discs on peg B to C. Thus, we can transfer n discs
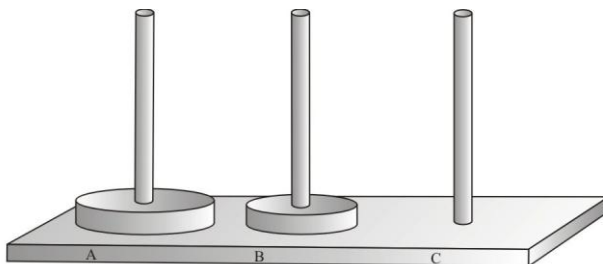
3a) Transfer disc 2 to peg B
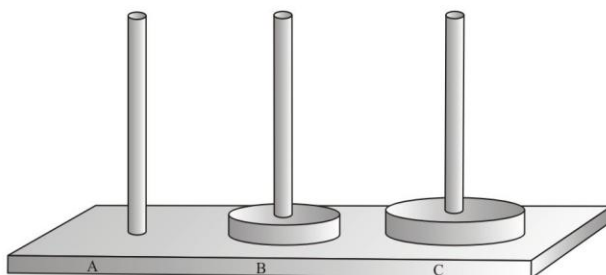
3b) Transfer disc 1 to peg C
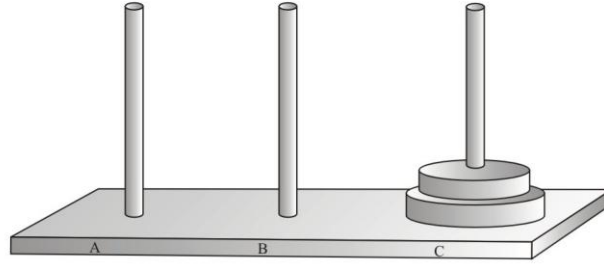
3c) **Transfer disc 2 to peg C.**

Fig. 3 : The moves involved in transferring 2 discs.

(for $n \geq 2$) in **at most** $2T_{n-1} + 1$ moves. So, $T_n \leq 2 T_{n-1} + 1$, if $n \leq 2$. Why have we used " $\leq$ " instead of " $=$ " here? Our construction proves only that $2T_{n-1} + 1$ moves are enough but can we transfer the disks with lesser number of moves? The answer is "No". At some point, we must move the largest disc. When we do, the $n - 1$ smallest must be on a single peg (why?), and it has taken at least T_{n-1} moves to put them there. After moving the largest disc for the last time, we must transfer the $n - 1$ smallest discs (which must again be on a single peg) back onto the largest; this too requires T_{n-1} moves. Hence, $T_n \geq 2T_{n-1} + 1$ if $n \geq 2$. Both the inequalities, $T_n \geq 2 T_{n-1} + 1$ and $T_n \leq 2 T_{n-1} + 1$ can be true only when $T_n = 2T_{n-1} + 1$.

* * *

As with the first example, we shall postpone solving the recurrence relation just obtained to Unit 3. Incidentally, once you have done the following exercise, you will note that the priests will require a minimum of $2^{64} - 1 = 18446\ 744\ 073\ 709\ 551\ 615$ moves to transfer the golden disks. Even at the rate of one move per second, it will take them more than 5×10^{11} years to solve the puzzle, so the doomsday is far away!

Try the next exercise which gives an explicit expression for T_n.

E2) Using induction, show that $T_n = 2^n - 1$, $n \geq 1$.

Now let us consider the third problem we had in mind. This is the number of ways to parenthesise an expression.

Example 3: Let us derive the recurrence relation for the number of ways to parenthesise the expression $x_1 + x_2 + \cdots + x_n$ so that only two terms will be added at a time. For example, the expression $((x_1 + x_2) + x_3)$ is fully parenthesised, but $(x_1 + x_2) + x_3$ is not. Suppose the number of ways of parenthesising the expression $x_1 + x_2 + \cdots + x_n$ is a_n. If we split the expression into $x_1 + x_2 + \cdots + x_{n-1}$ and x_n, $x_1 + x_2 + \cdots + x_{n-1}$ can be parenthesised in a_{n-1} ways and x_n can be parenthesised in a_1 ways. So, in this case, we can parenthesise the expression in $a_{n-1}a_1$ ways. Similarly, if we split up the expression into $x_1 + x_2 + \cdots + x_{n-2}$ and $x_{n-1} + x_{n-2}$, we can parenthesise the expression $x_1 + x_2 + \cdots + x_{n-2}$ in a_{n-2} ways and the expression $x_{n-1} + x_{n-2}$ in a_2 ways. The total number of ways in this case is $a_{n-2}a_2$ ways. In general, if we split up the expression into two sub expressions of size $n - k$ and k, we can parenthesise the expression in $a_{n-k}a_k$ ways. We can get the **total number** of ways of parenthesising the expression by **adding** the different **ways of parenthesising the expression corresponding to different ways of splitting the expression.** This is precisely

$$a_{n-1}a_1 + a_{n-2}a_2 + \cdots + a_{n-k}a_k + \cdots + a_1 a_{n-1}.$$

Therefore, the recurrence relation satisfied by a_n is

$$a_n = a_{n-1}a_1 + a_{n-2}a_2 + \cdots + a_{n-k}a_k + \cdots + a_1a_{n-1} \text{ for } n \geq 2 \text{ with } a_1 = 1.$$

Using the fact that $a_0 = 0$, we can extend this to

$$a_0 = 0, \ a_1 = 1, \ a_n = a_n a_0 + a_{n-1} a_1 + \ldots + a_1 a_{n-1} + a_{n-1} + a_0 a_n \ (n \geq 2).$$

* * *

The numbers $a_0, a_1, a_2, \ldots$ are called Catalan numbers. Catalan numbers arise in several other contexts. We will state two other contexts without proofs.

Example 4: Suppose two candidates A and B poll the same number of votes, n each, in an election. The counting of votes is usually done in some arbitrary order and therefore, during the counting process A may lead for some period and B may lead for some period. The number of ways of counting the votes such that A never trails B is the nth catalan number. We can represent a vote for A by + and a vote for B by −. Then, the nth catalan number is the number of sequences of pluses and minuses such that there are **at least** as many pluses as there are minuses at **any stage** of the sequence. Let us call such a sequence an admissible sequence. Let us take a special case where 8 votes are cast, 4 for A and 4 for B. One voting sequence where A never trails is $(+,-,+,+,-,-,+,-)$. If we drop the last three terms, we get the sequence $(+,-,+,+,-,-)$. Here, there are 3 pluses and 3 minuses i.e. there are at least as many pluses as there are minuses. If we drop the last term, we get $(+,-,+,+,-,-,+)$. Here, there are 4 pluses and 3 minuses, i.e. there are more pluses than minuses.

***

Example 5: You would have come across the data structure called **Stack** in MCS-021. Recall that, a stack is a list which can be changed by insertions or deletions at the top of the list. An insertion is called a **push** and a deletion is called a **pop**. A sequence of pushes and pops of length 2n is called **admissible** if there are n pushes and n pops and at each stage of the sequence there are **at least** as many pushes as pops. Suppose we have the string $123\ldots n$ of the set $N = \{1, 2, 3, \ldots, n\}$ and an admissible sequence of pushes and pops of length 2n. Each push in the sequence transfers the last element in the input string to the stack and every pop transfers the element on the top of the stack to the beginning of the output string. After n pushes and pops have been performed, the output string is a permutation of N called a **stack permutation.** The number of stack permutations of $123\ldots n$ is the nth Catalan number. Again, we can represent a pop by a + and a push by a −. If you pause and think for a minute, you can convince yourself that every admissible sequence of pops and pushes corresponds to an admissible sequence of pluses and minuses. Let us look at an example. Suppose $n = 4$ and we have the following admissible sequence of pops and pushes: $(+,-,+,+,-,-,+,+,-)$. Let us find the stack permutation corresponding to this. See Table 2.

Table 2 : Finding the slack permutation corresponding to the Admissible sequence
$(+,-,+,+,-,-,+,+,-)$

Sequence	Input String	Stack	Output String
+ (Push 4)	123	[4]	Empty
− (Pop 4)	123	[]	4
+ (Push 3)	12	[3]	4
+ (Push 2)	1	[23]	4
− (Pop 2)	1	[3]	24
− (Pop 3)	1	[]	324
+ (Push 1)	Empty	[1]	324
− (Pop 1)	Empty	[]	1324

So, the permutation obtained is 1324. This is an example of a stack permutation of size 4.

* * *

Try the following exercises now to check if you have understood our discussion so far.

E3)   Which of the following sequences are admissible?

    i) $(+,-,+,-,+,+,-,-)$

    ii) $(+,-,+,-,+,-,+,-)$

    iii) $(+,+,-,-,-,+,-,+)$

E4)   Is the statement 'An admissible sequence has to start with a plus and end with a minus' true? Explain your answer.

E5)   Consider the problem discussed in the introduction. Let a_n be the number of binary sequences of length n that do not contain consecutive zeros. Show that $a_n = a_{n-1} + a_{n-2}$ for $n \geq 3$.

You will have noticed that in each of the three problems, we have been able to express the nth term of a sequence in terms of one or more previous terms and a function of n. This gives you a method to compute the terms of the sequence accurately, given enough time. At times, if the relation between the terms is in a reasonably nice form, we can even "solve" the recurrence, that is, express the nth term as a function of n. You will learn how to solve these three recurrences in the next 2 units.

Let us now consider some more recurrent problems.

1.3   MORE RECURRENCES

You have been exposed to some famous recurrent problems in the previous section. In this section, we shall take another look at setting up recurrence relations for combinational problems of the kind you would have encountered in MCS-013. You will find that in trying to determine recurrence, we are really attempting to describe the counting inductively. In most cases, you will see that the recurrence relation leads to an alternate method of solution, although the methods themselves will be dealt with in Unit 3.

Example 6: Let C_n be the number of comparisons needed to sort a list of n integers. Let us find a recurrence relation for C_n. We first find the minimum of the n elements. This will be the first element of the list. We compare the first two elements and find the minimum among the two. We compare this minimum with the third element, and so on. For finding the minimum of n elements we have to make n−1 comparisons. For example, if we want to find the minimum of the list 3, 2, 4, 1, 5, we will have to make 4 comparisons as shown in Fig.4.
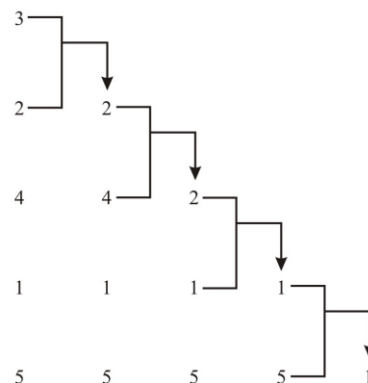

Fig.4 : Comparisons for finding the minimum of 3,2,4,1,5

After making the minimum element the first element of the list we can sort the remaining n−1 elements with C_{n-1} comparisons and append it after first element. So, n−1 + C_{n-1} comparisons are required to sort a list of n elements.

(i.e.) $C_n = C_{n-1} + n-1$

* * *

Try the following exercise which asks you to prove an expression C_n.

E6) In Example 6, using the recurrence relation for C_n, show that

$$C_n = \frac{1}{2}n(n-1), n \geq 1.$$

Example 7: You may recall that the set of all subsets of any non-empty set, S, is called its **power set,** and denoted by $p(S)$. Let us determine a recurrence relation satisfied by $s_n = |p(S)|$, where $|S| = n$. Let us take $S = \{1, 2, …, n\}$. Now, any subset, A, of S either contains the number n or does not contain the number n. Let us consider these two mutually exclusive cases separately and count the number of such subsets, A. If $n \in A$, then $A = A' \cup \{n\}$, where A' is a subset of $\{1, 2, …, n-1\}$, there are s_{n-1} such subsets A. So, the number of subsets A that contain n is the same as the number of subsets of $\{1, 2, …, n-1\}$, which is s_{n-1}. On the other hand if $n \notin A$, then, in fact, A is a subset of $\{1, 2, …, n-1\}$, and there are s_{n-1} of these too. Combining these, we see that $s_n = s_{n-1} + s_{n-1} = 2s_{n-1}, n \geq 1,$ with $s_0 = 1$.

* * *

We ask you to prove that $s_n = 2^n$ in the next exercise.

E7) In Example 5, using the recurrence relation for s_n, show that $s_n = 2^n, n \geq 0$.

Example 8: Recall that a **bijection** is a **one-one, onto** mapping of a set onto itself. It is quite easy to determine directly the number of bijections of an n-set (a set with n elements). We will, however, look at a recurrence relation satisfied by the number of bijections, b_n, of any n-set, say $\{1, 2, …, n\}$. To begin with, if f is any such bijection, $f(n)$ could be any one of the n elements of the set $\{1, 2, …, n\}$; but now we must map $\{1, 2, …, n-1\}$ bijectively to $\{1, 2, …, n\} \setminus \{f(n)\}$; this can be done in b_{n-1} ways. $f(n)$ can be chosen in n ways. In all then, $b_n = nb_{n-1}, n \geq 2$, with $b_1 = 1$.

* * *

Again, we ask you to prove an expression for b_n.

E8) In Example 8 using the recurrence relation for b_n, show that $b_n = n!, n \geq 1$.

We end this section by looking again at the problem of derangements, discussed in MCS-013.

Example 9: You may recall that the problem is to determine the number of derangements of n objects, d_n, and that we had employed the method of **Inclusion-Exclusion** to solve it. Let us find a recurrence relation for d_n. Recall that d_n counts the number of permutations of n objects that leave **no** object fixed. Any such permutation is called a **derangement**. Let us begin by labeling the objects serially: 1, 2, …, n. In any such derangement of n objects, 1 gets sent to some i, where $i \neq 1$. Two cases arise: For the same derangement, either i gets sent back to 1 or it does not. In the first case, we can leave out 1 and i from the original set and obtain a derangement of n−2 objects; there are d_{n-2} such possibilities. Suppose i is not sent back to 1. Then, we can get a derangement of $\{2, 3, …, n\}$ as follows. Some number must be mapped back to 1, suppose j is mapped to 1. We can define a derangement of $\{2, 3, …, n\}$ as follows:

1) Map j to i
2) Every other element is mapped according to the original derangement

Thus there are d_{n-1} such possibilities to obtain a derangement of $n-1$ objects. Therefore, assuming 1 gets sent to i, there is a total of $d_{n-1} + d_{n-2}$ possibilities. Observing that i could have been any number between 2 and n, we conclude that $d_n = (n-1)(d_{n-1} + d_{n-2})$ for $n \geq 3$. To complete the recurrence relation, we note that $d_1 = 0$ and $d_2 = 1$. You will have noticed that to compute d_n one needs to know the values of the two preceding terms. Can we get to compute d_n on the basis of the value of only one preceding term, d_{n-1}? To explore this, let us write the recurrence in the form

$$d_n - nd_{n-1} = -[d_{n-1} - (n-1)d_{n-2}].$$

We now observe that the expression on the right hand side within the brackets is got from the expression on the left hand side by merely replacing n by $n-1$. If we write $D_n = d_n - nd_{n-1}$, we have the simplified expression $D_n = -D_{n-1}$. But then $D_{n-1} = -D_{n-2}$, and so $D_n = D_{n-2}$. Continuing this procedure, we arrive at

$$D_n = (-1)^{n-2} D_2 = (-1)^n [d_2 - 2d_1] = (-1)^n.$$

Therefore, we have $d_n = nd_{n-1} + (-1)^n$ if $n \geq 2$, with $d_1 = 0$.

$$* * *$$

In the next exercise, we ask you to prove the expression for d_n we derived in Block 2, Unit 2, but using the recurrence for d_n this time.

E9)   Using induction and either of the two recurrence relations for d_n, show that

$$d_n = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}, n \geq 1.$$

We end this section with a few problems in which you are required to set up the recurrence equations.

E10)   Set up a recurrence relation for the determinant of the $n \times n$ matrix with 1

along the main diagonal and with 1 on either side of the main diagonal in each

row and zero elsewhere. For example, the 3×3 determinent is $\begin{vmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}$.

E11)   Set up a recurrence relation for the number of n digit sequences on integers {0, 1, 2, 3} having an even number of 0's.

E12)   Show that the number of r-permutations of n distinct objects, P(n,r), satisfies the recurrence relation

   $P(n, r) = P(n-1, r) + rP(n-1, r-1), n \geq 1, r \geq 1.$

E13)   Let S_r^n denote the **Stirling numbers of the second kind,** that is, the number of ways to distribute r distinct objects into n nondistinct boxes with no box left empty. Show that S_r^n satisfies the recurrence relation

   $S_{r+1}^n = S_r^{n-1} + nS_r^n, 1 < n < r.$

In the next section, we give all relevant definitions and introduce the notation for studying recurrence relations.

1.4   DEFINITIONS

We hope you have got a fairly good idea of what a "recurrence relation" is, as well as how to set it up by now. It is time to formalise the procedure and set up a more

rigorous mathematical pedestal for it. Let us now define a recurrence relation formally.

Definition: Let $\{a_n : n \geq 0\}$ be a sequence of real or complex numbers. A **recurrence relation (or a recurrence equation)** is an expression of the form

$$a_n = F(a_{n-1}, a_{n-2}, \ldots, n)$$

where F is a function of some of the variables a_{n-1}, a_{n-2}, …, a_0, n. Note that all the a_is need not occur in the expression.

In other words, it allows us to compute the nth term of a sequence from one or more of the preceding terms. The symbol "F" merely denotes a (any) function and the variables are (some or all of) the preceding terms in the sequence as also n. For our purposes, we shall only deal with such functions F which are **polynomials and depend** on only **finitely** many variables,

$$a_n = F(a_{n-1}, a_{n-2}, \ldots, a_{n-k}, n)$$

Definition: The **order** of the recurrence relation defined by

$$a_n = F(a_{n-1}, a_{n-2}, \ldots, a_{n-k}, n)$$

is k, where a_n depends on one or more of the previous k terms and k is the **smallest** such integer. We do not define an order for recurrence relations of the form $a_n = F(a_{n-1}, a_{n-2}, \ldots, a_o, n)$ that depend on each of its previous terms.

Therefore, if we can compute the nth term of a sequence from the preceding k terms, but not from the preceding k− 1 terms, we define the order to be k.

Definition: The **degree** of the recurrence relation is the degree of F treated as a polynomial in its variables **excluding** n. If F is not a polynomial in its variables, no degree is assigned to the recurrence relation.

A recurrence relation of degree **one** is also called **linear**, one of degree **two quardratic**, and so on, just like we have in the case of polynomials. After all, the notion of "degree" is tied up with the degree of the defining polynomial F.

Definition: A recurrence relation is called **homogeneous** if it contains no term that depends only on the variable n. A recurrence relation that is not homogeneous is said to be **non-homogeneous** or **inhomogeneous**.

Thus, for a recurrence to be called homogeneous, every term defining the recurrence must contain at least one of the preceding terms of the sequence. Usually, the term **homogeneous** is used for linear recurrences regardless of its order.

Examples:

1) $a_n = 3a_{n-1} + n^2$ is **nonhomogeneous** of order 1 and degree 1.
2) $a_n = na_{n-2} + 2^n$ is **nonhomogeneous** of order 2 and degree 1.
3) $a_n = \sqrt{a_{n-1}} + a_{n-2}^2$ is **homogeneous** of order 2, but has no degree.
4) $a_n = a_{n-1} + a_{n-2} + \ldots + a_0$ is **homogeneous,** has no order, but has degree 1.
5) $a_n = a_{n-1}^2 + a_{n-2} a_{n-3} a_{n-4}$ is **homogeneous** of order 4 and degree 3.
6) $a_n = \sin a_{n-1} + \cos a_{n-2} + \sin a_{n-3} + \ldots + e^n$ is **nonhomogeneous,** has no order and no degree.
7) $a_n = f_1(n)a_{n-1} + f_2(n)a_{n-2} + \ldots + f_{n-k}(n) a_{n-k} + g(n)$ represents the general form of a linear k th order recurrence relation $(f_{n-k}(n) \neq 0)$. It is **homogeneous** if $g(n) = 0$ for each n, and **nonhomogeneous** otherwise.

8) $a_n = a_0 a_{n-1} + a_1 a_{n-2} + \ldots + a_{n-1} a_0$ $(n \geq 2)$ with $a_0 = 0$, $a_1 = 1$ is a nonlinear recurrence relation.

9) $a_{n,k} = a_{n-1,k} + a_{n-1,k-1}$ is a recurrence relation in two variables n and k. Taking $a_{n,k} = C(n,k)$, the given relation is nothing but Pascal's identity with initial conditions $a_{n,0} = C(n,0) = a_{n,n} = C(n, n) = 1$ for all $n \geq 0$ and $a_{n,k} = 0$, $k \geq n$

10) $a_{n,k} = a_{n-2,k-1} + a_{n-3,k-1} + a_{n-4,k-1}$, with initial conditions $a_{2,1} = a_{3,1} = a_{4,1} = 1$ and $a_{k,1} = 0$ otherwise, is a recurrence relation in two variables (This is the recurrence relation for the ways to distribute n identical balls into k distinct boxes with between two and four balls in each box).

11) $a_n = a_{n/2} + 1$ with $a_1 = 0$ (n a power of 2) is a nonlinear recurrence relation.

Check your understanding of the concepts of order, degree and homogeneity with respect to recurrence relations by trying the following exercises.

E14) Find the order and degree of each of the following recurrences. Also, state whether they are homogeneous or nonhomogeneous.

  a) $a_n = a_{n-1} + a_{n-2}$

  b) $L_n = L_{n-1} + n$

  c) $d_n = n d_{n-1} + (-1)^n$

  d) $a_n = a_n a_0 + a_{n-1} + \cdots + a_0 a_n$ $(n \geq 2)$.

You must have observed while looking at the various examples above that the recurrence relation alone will not define for you the terms of the sequence. To be able to do this, one needs to know where to begin the sequence. For example, the nth Fibonacci number and the number of binary sequences of length n that do not contain consecutive zeros satisfy the same recurrence relation. If a_n is defined in terms of a_{n-1} alone, deciding the value for a_0 (or, a_1, or wherever you wish to begin the sequence) uniquely describes the sequence for you. More generally, in case of a kth order recurrence, one needs to know the first k terms, typically $a_0, \ldots, a_{k-1}$ of the sequence in order to uniquely define the sequence. A well-defined linear recurrence relation of degree k consists of a recurrence part and initial conditions for k consecutive values.

Definition: We say that a kth order recurrence relation is a **recurrence relation with initial conditions** provided one or more of the k values $a_0, a_1, \ldots, a_{k-1}$ are known.

Definition: A function $f(n)$ is said to be a **general solution** to the recurrence relation if it satisfies the recurrence equation.
A function $g(n)$ is said to be the **particular solution** to a recurrence relation if it satisfies the recurrence equation, together with the initial conditions.

Please note that there are infinitely many "general solutions" to any recurrence relation without initial condition(s), one for each set of values for the initial terms, but only one "solution" once the first k initial terms are fixed for recurrence relations of order k. You have been verifying that given functions are indeed solutions to the recurrences of the previous two sections. We give a few more examples of a simpler nature. The solution of recurrence relations will be discussed in unit 3.

Examples:

1) The general solution to $a_n = a_{n-1}$ is $a_n = c$, where c is any constant, but if in addition $a_0 = 1$, then the solution is $a_n = 1$, $n \geq 0$.

2) The general solution to $a_n = a_{n-1} + 1$ is $a_n = c + n$, where c is any constant; if $a_0 = 0$, then the solution is $a_n = n$, $n \geq 0$.

3) The general solution to $a_n = k a_{n-1}$ is $a_n = ck^n$, where c is any constant; if $a_0 = 1$, then the solution is $a_n = k^n$, $n \geq 0$.

4) The general solution to $a_n = a_{n-1} + a_{n-2}$ is

5) $a_n = c_1 \left(\dfrac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\dfrac{1-\sqrt{5}}{2}\right)^n$, where c_1, c_2 are any constants.

6) If $a_1 = 1$, $a_2 = 3$, then the particular solution is

7) $a_n = \left(\dfrac{1+\sqrt{5}}{2}\right)^n + \left(\dfrac{1-\sqrt{5}}{2}\right)^n$, $n \geq 1$.

8) The particular solution to $a_n - \dfrac{n}{n-1} a_{n-1} = n^3$ with $a_1 = 1$, is

9) $a_n = \dfrac{n^2(n+1)(2n+1)}{6}$

$$* * *$$

In the concluding section, we discuss some common types of recurrence relations that result from divide and conquer algorithms.

1.5 DIVIDE AND CONQUER RELATIONS

Often, to solve a problem, we can partition the problem into smaller problems, find solutions to the smaller problems and then combine the solutions to find a solution for the whole. We can repeatedly partition the problem till we reach a stage where we can solve the problem quite easily. This approach is called the **divide-and-conquer** approach. Let us start with an example. **Throughout this section we will assume that** $n = 2^k$.

Example 10: Consider the problem of finding the maximum and minimum of a list of n numbers. Here let us assume that $n = 2^k$ for some k to make things simpler for us. Let a_n be the number of comparisons required for finding the maximum and minimum of a list of n numbers. We can partition the list into two lists of size $\dfrac{n}{2}$ each. The maximum and minimum of each of the list can be found using $a_{\frac{n}{2}}$ comparisons. We can then compare minimum (resp. maximum) of the lists to get the minimum (resp. maximum) of the list, i.e., we will need two more comparisons. So, $a_n = 2a_{\frac{n}{2}} + 2$ for $n \geq 2$. $a_2 = 2$. We leave it as an exercise for you to check that $a_n = \dfrac{3}{2}n - 2$, when n is a power of 2.

E15) Check that $a_n = \dfrac{3}{2}n - 2$ is a solution to the recurrence $a_n = 2a_{\frac{n}{2}} + 2$ where n is a power of 2 and $a_2 = 1$.

Definition: We call a recurrence a divide-and-conquer recurrence if it has the form
$$a_n = ba_{\frac{n}{a}} + d(n)$$
where $a > 1, b > 1$ are integers and $d(n)$ is a function of n.

Such a relation results when we split a problem of size n into b sub problems of size $\dfrac{n}{a}$. After solving each of the sub problems we may need $d(n)$ steps to get the solution to the original problem of size n. In Example 10, we splitted a problem of size n into 2

sub problems of size $\frac{n}{2}$ each and we needed 2 more steps to get the answer to the original problem. So, a = b = 2 and d(n) = 2 in this example.

Example 11: In a tennis tournament, each entrant plays a match in the first round. Next, all winners from the first round play a second-round match. Winners continue to move on to the next round, until finally only one player is left as the tournament winner. Assuming that tournaments always involve $n = 2^k$ players, for some k, find the recurrence relation for the number rounds in a tournaments of n players.
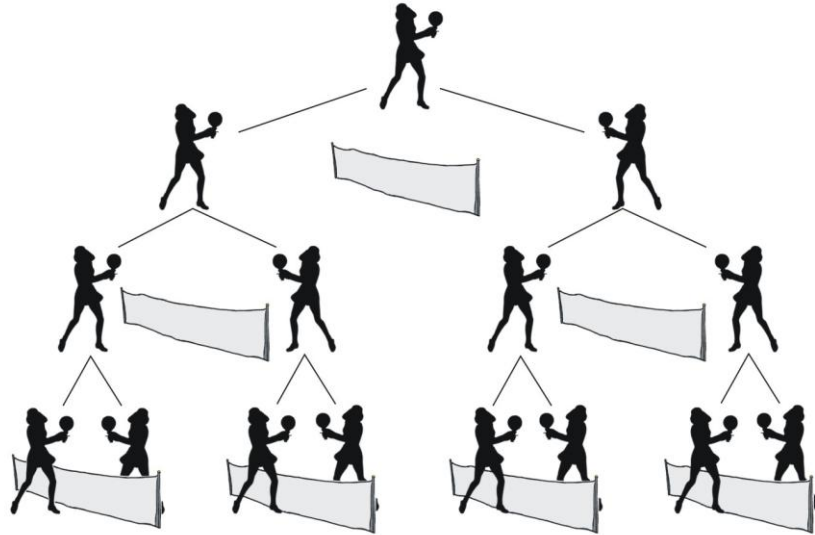

Fig.5: A tournament involving 8 players

Solution: The tournament is subdivided into 2 sub tournaments of n/2 players each. Here the first round of both the tournaments are held together so that they are considered as a single first round. Similarly, all the other rounds are held together. So, after $a_{n/2}$ rounds, one player will be left in both the sub tournaments. The winner of the match between the two is the winner of the tournament. So, $a_n = 2a_{n/2} + 1$. See Fig.5 for a tournament involving 8 players.

* * *

Let us now discuss some more problems where divide and conquer approach can be used.

Example 12: (Binary search) Suppose we have a **sorted** list of $n=2^k$ numbers and we want to check whether a particular number is in the list or not. If we compare the number with all the elements of the list we need n comparisons in the worst case. However, we will describe a better algorithm called the binary search algorithm which uses the divide- and-conquer paradigm. Let us look at a particular example. Suppose we want to check whether 12 is in the list 1,2,3,5,6,7,12,13. We split up the list into two parts. 1,2,3,5 and 6,7,12,13. We compare 12 with the last element of the first list which is 5. We have $5 < 12$ and all the elements in the first list are less than 5. So, we can discard the first list. The second list is again split into two parts, 6,7 and 12,13. Since $7 < 12$, we discard the first part. We split 12,13 into two lists of one element each. The first of these two lists, 12, contains 12. Let a_n denote the number of comparisons required while searching for an element in a sorted list of size n. Then $a_1=1$. In general, we can split up the list into two equal parts of n/2 elements each. We can compare our element with the last element, which is the largest element, of the first list. This will tell us whether we have to search in the first list or in the second list. In either case we have to search in a list of size n/2 only. So, $a_n=a_{n/2}+1$.

* * *

Example 13: (Merge sort) Suppose we want to sort a list of $n=2^k$ elements in ascending order. We can divide the list into parts of size n/2 each, sort them and merge

them into a sorted list. Suppose a_n is the number of comparison required to sort a list of n elements then, the two lists can be sorted using $a_{n/2}$ comparisons each. Let us call the lists L_1 and L_2. We will merge them into a new sorted list L. We compare the first (smallest) elements of the list, add the smallest element of the two to the new list L and remove the smallest element from the list from which we selected it. We repeat this till one of the lists, say L_1 is empty. Then we append L_2 to the list L to get a sorted list. See Table 4, where we show how to merge two sorted lists 1,2,5,6 and 3,4.

Step	L_1	L_2	L	Action taken
1	1, 2, 5, 6	3, 4	--	1<3. Remove 1 from L_2 and add it to L.
2	2, 5, 6	3, 4	1	2<3. Remove 2 from L_1 and add it to L.
3	5, 6	3, 4	1, 2	5>3. Remove 3 from L_2 and add it to L.
4	5, 6	4	1, 2, 3	5>4 Remove 4 from L_2 and add it to L.
5	5, 6	--	1, 2, 3, 4	L_2 is empty. No more comparisons required. Add the remaining elements of L_1 to L to get the sorted list 1,2,3,4,5,6

Here L_1 and L_2 are not of the same size. We needed 4+2−1=5 comparisons to sort lists of sizes 4 and 2. In general, to merge two sorted lists of size m and n under this method at most m+n−1 comparisons are required. So, to merge the two lists of size n/2 each, at most n−1 comparisons are required. So, $a_n = 2a_{n/2} + n - 1$.

E16) Finding the nth power of an integer i, by successive multiplications by i requires $n - 1$ multiplications. Assuming $n = 2^k$, describe a divide and conquer algorithm such that if a_n is the number of multiplications to find the nth power, then $a_n = a_{n/2} + 1$. Is the algorithm desirable given that the solution is given by $a_n = \log_2(n)$?

E17) Finding the product of a list of n integers by successive multiplication requires $n - 1$ multiplications. Assuming $n = 2^k$, describe a divide and conquer algorithm for which the number of multiplications a_n satisfies the recurrence relation $a_n = 2a_{n/2} + 1$

E18) To multiply two n-digit numbers, one must do normally n^2 digit-times-digit multiplications. Use a divide and conquer algorithm to do better when n is a power of 2.

With this we have come to the end of this unit. Next two units will deal with the methods of solving recurrence relations. Now let us take a quick look at what we have discussed in this unit.

1.6 SUMMARY

In this Unit **recurrence relations** we:
1) discussed several examples of recurrence relations, drawn from well-known problems and from routine exercises in combinatorics.
2) discussed how to set up recurrence relations after having read this unit.
3) defined the homogeneous, non-homogeneous recurrence relations.
4) defined the order and degree of a recurrence relation.

5) discussed setting up of recurrence relations with the help of divide and conquer paradigm.

1.7 SOLUTIONS/ANSWERS

E1) It is easy to check that $f_1 = 1 = f_2$. With $\alpha = \dfrac{1+\sqrt{5}}{2}$, $\beta = \dfrac{1-\sqrt{5}}{2}$, we observe that $\alpha\beta$ are solutions to the equation $x^2 - x - 1 = 0$. $\therefore \alpha^2 = \alpha + 1$, $\beta^2 = \beta + 1$. If $n \geq 3$,

$$\sqrt{5}\left(f_{n-1} + f_{n-2}\right) = \left(\alpha^{n-1} - \beta^{n-1}\right) + \left(\alpha^{n-2} - \beta^{n-2}\right)$$
$$= \alpha^{n-2}\left(\alpha + 1\right) - \beta^{n-2}\left(\beta + 1\right)$$
$$= \alpha^{n-2}\alpha^2 - \beta^{n-2}\beta^2$$
$$= \alpha^n - \beta^n = \sqrt{5}f_n,$$

as desired.

E2) Observe that $T_1 = 1$. If $n \geq 2$, $2T_{n-1} + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 1 = T_n$, verifying the formula.

E3) 1) and 2) are admissible. If we drop the last three terms from 3), we get $(+, +, -, -, -)$ which has two pluses and three minuses. So, 3) is not admissible.

E4) Yes, this is true. If an admissible sequence starts with a $-$, the sequence got by removing all the terms except the first one has more minuses (1 minus) than pluses (0 Plus). Suppose an admissible sequence ends in a plus. The sequence we get by removing a plus has more pluses than minuses. If we put back the plus, the sequence will still have more pluses than minuses. But, an admissible sequence must have equal number of pluses and minuses.

E5) Consider any string of length n. The last bit of the sequence is either 0 or 1. If the last bit is 0, the previous bit must be 1. So, it can be obtained by adding '10' to a string of length $n - 2$. (Note that 2 consecutive 1s are allowed!)
If the last bit is 1, this can be obtained by adding 1 to a string of length $n-1$.
$\therefore a_n = a_{n-1} + a_{n-2}$
Note that, a_n satisfies the same recurrence satisfied by the Fibonacci sequence. We have $f_1 = 1, f_2 = 1$, but $a_1 = 2, a_2 = 3$.

E6) It is easy to see that $C_1 = 0$. If $n \geq 2$,

$$C_{n-1} + n - 1 = \frac{1}{2}(n-1)(n-2) + n - 1 = \frac{1}{2}n(n-1) = C_n, \text{ as desired.}$$

E7) Observe that $s_0 = 1$. If $n \geq 1$, $2s_{n-1} = 2.2^{n-1} = 2^n = s_n$, verifying the formula.

E8) We note that $b_1 = 1$. If $n \geq 2$, $nb_{n-1} = n.(n-1)! = n! = b_n$, as required.

E9) We check that $d_1 = 0$, $d_2 = 1$. To verify the first order recurrence relation, note that if $n \geq 2$,

$$nd_{n-1} + (-1)^n = n\left[(n-1)!\sum_{i=0}^{n-1}\frac{(-1)^i}{i!}\right] + (-1)^n$$
$$= n!\left(\sum_{i=0}^{n}\frac{(-1)^i}{i!} - \frac{(-1)^n}{n!}\right) + (-1)^n$$

$$= n! \sum_{i=0}^{n-1} \frac{(-1)^i}{i!} = d_n$$

as desired.

In case of the second order recurrence relation, if $n \geq 1$,

$$(n-1)(d_{n-1} + d_{n-2}) = (n-1)\left\{ \left[(n-1)! \sum_{i=0}^{n-1} \frac{(-1)^i}{i!} \right] + \left((n-2)! \sum_{i=0}^{n-2} \frac{(-1)^i}{i!} \right) \right\}$$

$$= n.(n-1)! \sum_{i=0}^{n-1} \frac{(-1)^i}{i!} - (n-1)! \sum_{i=0}^{n-1} \frac{(-1)^i}{i!} + (n-1)! \sum_{i=0}^{n-2} \frac{(-1)^i}{i!}$$

$$= n! \sum_{i=0}^{n-1} \frac{(-1)^i}{i!} - (n-1)! \frac{(-1)^{n-1}}{(n-1)!} = n! \sum_{i=0}^{n-1} \frac{(-1)^i}{i!} + (-1)^n$$

$$= \sum_{i=0}^{n} \frac{(-1)^i}{i!} = d_n,$$

as desired.

E10) Let Δ_n denote the required $n \times n$ determinant. Expanding about the **first** row, we get Δ_{n-1} minus the determinant which when expanded about its first row yields Δ_{n-2}. The corresponding recurrence relation is $\Delta_n = \Delta_{n-1} - \Delta_{n-2}$, $n \geq 3$, with $\Delta_1 = 1$, $\Delta_2 = 0$.

E11) Let a_n denote the number of n-digit sequences containing an even number of 0's. Then there are a_{n-1} (n − 1)-digit sequences that have an even number of 0's and $4^{n-1} - a_{n-1}$ (n −1)-digit sequences that have an odd number of 0's. To each of the a_{n-1} sequences that have an even number of 0's, the digit 1,2 or 3 can be appended to yield sequences of length n that contain an even number of 0's. To each of the $4^{n-1} - a_{n-1}$ sequences that have an odd number of 0's, the digit 0 must be appended to yield sequences of length n that contain an even number of 0's. Therefore, for $n \geq 2$, $a_n = 3a_{n-1} + 4^{n-1} - a_{n-1} = 2a_{n-1} + 4^{n-1}$, with $a_1 = 3$.

E12) Of the n distinct objects, pick any **one** object and call it "special". Then, the number of r-permutations in which this "special" object does not appear is $P(n - 1, r)$ because this is the number of r-permutations of the remaining (n − 1) objects. On the other hand, if the "special" object does appear, the number of r-permutations is $rP(n - 1, r - 1)$ because the "special" object could be in any one of r positions between objects or at either end, and we have then to determine the number of (r − 1)-permutations of n − 1 objects. Combining the two, we get the required recurrence.

E13) This is somewhat similar to the previous one; choose a "special" object first. The box containing this object either contains no other object or contains at least one more. In the first case, we need to distribute r distinct objects into n−1 nondistinct boxes, with no empty box; the number of ways in which to do this is S_r^{n-1}. Otherwise, the special "object" may be placed into any one of the n (nondistinct) boxes (there are n choices), and we still need to distribute r objects into n nondistinct boxes, with no empty box; there are S_r^n such choices for each choice of the box the "special" object is placed in. Combining the two cases gives the recurrence relation.

E14) a)   order 2, degree 1
       b)   order 1, degree 1
       c)   order 1, degree 1
       d)   order not defined. Degree is 2.

E15) When n = 2, $\frac{3}{2}$ n − 2 = 1. Let us suppose n = 2^k. Let us apply induction on k.

Suppose it is true for k, say m = 2^k, then

$$a_m = 2a_{m/2} + 2$$

$$= 2\left(\frac{2m}{4} - 2\right) + 2$$

$$= \frac{3m}{2} = 4 + 2$$

$$= \frac{3}{2} m - 2$$

Thus it is true for k+1.

E16) Divide n by 2, find $i^{\frac{n}{2}}$, and square it. Thus $a_n = a_{\frac{n}{2}} + 1$. The algorithm is

desirable since $\log_2(n)$ grows more slowly than n.

E17) Divide n by 2. Find the product of $\frac{n}{2}$ integers and the product of last $\frac{n}{2}$ integers.

Multiply two products obtained. Thus $a_n = 2 a_{\frac{n}{2}} + 1$.

E18) Let n be a power of 2. Let the two n-digit numbers b A and B. We split each of

these numbers into two $\frac{n}{2}$ digit parts:

$A = A_1 10^{\frac{n}{2}} + A_2$ and

$B = B_1 10^{\frac{n}{2}} + B_2$ (like 1235 = 12 × 100 + 35)

Then A.B = $A_1 B_1 10^n + A_1 B_2 \ 10^{\frac{n}{2}} + A_2 B_1 10^{\frac{n}{2}} + A_2 B_1 10^{\frac{n}{2}} + A_2 B_2$

We need only to make three $\frac{n}{2}$ - digit multiplications, $A_1.B_1$, $A_2.B_2$ and

$(A_1 + A_2) . (B_1 + B_2)$ to determine A . B since

$A_1 . B_2 + B_2 . A_1 = (A_1 + A_2). (B_1 + B_2) - A_1 . B_1 - A_2 . B_2$

Actually $(A_1 + A_2)$ or $(B_1 + B_2)$ may be $\left(\frac{n}{2} + 1\right)$-digit numbers but this slight

variation does not effect the general magnitude of our solution (like 1295 = 12 × 10^2 + 95). If a_n represents the number of digit-times multiplications needed to multiply two n-digit numbers by the above procedure, this gives the recurrence relation $a_n = 3a_{\frac{n}{2}}$

a_n is proportional to $n^{\log_2 3} = n^{1.6}$ — a substantial improvement over n^2.