Recursion and Combinatorial Mathematics in *Chandaśāstra*[*]

Amba Kulkarni
Department of Sanskrit Studies,
University of Hyderabad
Hyderabad, India
apksh@uohyd.ernet.in

January 20, 2017

Abstract

Contribution of Indian Mathematics since Vedic Period has been recognised by the historians. *Piṅgaḷa* (200 BC) in his book on *Chandaśāstra*, a text related to the description and analysis of meters in poetic work, describes algorithms which deal with the Combinatorial Mathematics. These algorithms essentially deal with the binary number system - counting using binary numbers, finding the value of a binary number, finding the value of ${}^{n}C_{r}$, evaluating 2^{n}, etc. All these algorithms are tail recursive in nature. Some of these algorithms also use the concept of stack variables to stack the intermediate results for later use. Later work by *Kedār Bhaṭṭa* (around 800 AD), however, has only iterative algorithms for the same problems. We describe both the recursive as well as iterative algorithms in this paper and also compare them with the modern works.

1 Introduction

'Without any purpose, even a fool does not get initiated.[1] Thus goes a saying in Sanskrit. If we look at the rich Sanskrit knowledge base, we find that all the branches of knowledge that exist in Sanskrit literature were originated in order to address some problems in day today life. While addressing them, we find that, there were also remarkable efforts in generalising the results and findings. For example, about the*Pāṇini*'s monumental work on *aṣṭādhyāyī* (500 BC), Paul Kiparsky says "many of the insights of *Pāṇini*'s gramar still remain to be recaptured, but those that are already understood contribute a major

[*]The earlier version of this paper is available at http://arxiv.org arxiv:math/0703658v2 dated 7th March 2008

[1]*prayojanam anuddiśya na mando api pravartate*|

theoretical contribution." (in the encyclopaedia of Language and Linguistics, ed Asher, pp 2923).

Mathematics is also no exception to it. Contribution of Indian mathematicians dates back to the Vedic period [1]. The early traces of geometry and algebra are found in *Śulvasūtra*s of Vedic period where the purpose of this geometrical and algebraic exercise was to build brick altars of different shapes to perform Vedic rituals. Fixing Luni-Solar calendar was another important task which led to the development of calculus in India. The development flourished in the classical period from Aryabhaṭṭa (500 AD) to Bhaskarachārya II (1150 AD) and further in the Kerala school of mathematics from 1350 AD to 1650 AD.

However the discovery of binary number system by Indians escaped the attention of Western scholars, may be because *Chandaśāstra* was considered as mainly a text related to description and analysis of meters in poetic literary work, totally unrelated to mathematics. B. Van Nooten [2] brought it into limelight.

*Veda*s are in poetic form. They are written in different meters (*Chanda*s). These *Chanda*s have been studied in great detail. Piṅgaḷa's *Chandaśāstra* forms a part of *Vedānga*, essential to understand the *Veda*s.

Chandaśāstra by *Piṅgaḷa* is the earliest treatise found on the Vedic Sanskrit meters. *Piṅgaḷa* defines different meters on the basis of a sequence of what are called laghu and guru (short and long) syllables and their count in the verse. The description and analysis of sequence of the laghu and guru syllables in a given verse is the major topic of *Piṅgaḷa*'s work. He has described different sequences that can be constructed with a given number of syllables and has also named them. At the end of his book on *Chandaśāstra*, *Piṅgaḷa*[3] gives rules to list all possible combinations of laghu and guru (L and G) in a verse with 'n' syllables, rules to find out the laghu-guru combinations corresponding to a given index, total number of possible combinations of 'n' L-G syllables and so on. In short *Piṅgaḷa* describes the 'combinatorial mathematics' of meters in *Chandaśāstra*. Later around eighth century AD *Kedār Bhaṭṭa*[4] wrote *Vṛttaratnākara* a work on non-vedic meters. This seems to be independent of *Piṅgaḷa*'s work, in the sense that it is not a commentary on *Piṅgaḷa*'s work, and the last chapter gives the rules related to combinatorial mathematics which are totally different from *Piṅgaḷa*'s approach. In the thirteenth century, *Halāyudha* in his *Mṛta sanjīvanī*[3] commentary on *Piṅgaḷa*'s work, has again described the *Piṅgaḷa*'s rules in great detail.

Piṅgaḷa's *Chandaśāstra* contains 8 chapters. The eighth chapter has 35 *sūtra*s of which the last 16 *sūtra*s from 8.20 to 8.35 deal with the algorithms related to combinatorial mathematics. *Kedār Bhaṭṭa*'s *Vṛttaratnākara* contains 6 chapters, of which the sixth chapter is completely devoted to algorithms related to combinatorial mathematics.

Few words on the *sūtra* style of *Piṅgaḷa* are in order. The *sūtra* style was prevalent during *Piṅgaḷa*'s period. *Aṣṭādhyāyī* of *Pāṇini* is the classic example of *sūtra* style. A *sūtra* is defined as

alpākṣaram asandigdham sāravat viśvatomukham |
astobham anavadyam ca sūtram sūtravido viduḥ ||

A *sūtra* should contain minimum number of words (*alpākṣaram*), it should be unambiguous (*asamdigdham*), it should contain essence of the topic which the *sūtra* is meant for (*sāravat*), it should be general or should have universal validity(*viśvatomukham*), it should not have any unmeaningful words (*astobham*) and finally it should be devoid of any fault (*anavadyam*).

*Sūtra*s are like mathematical formulae which carried a bundle of information in few words. They were very easy to memorise. They present a unique way to communicate algorithms or procedures verbally. The *sūtra* style was adopted by Indians in almost every branch of knowledge. For example, *Piṅgaḷa*'s *sūtra*s are for combinatorics whereas Paṇini's *sūtra*s are for language analysis. Another important feature of *sūtra* style is use of *anuvṛtti.* Generally all the *sūtra*s that deal with a particular aspect are clubbed together. To avoid any duplication utmost care had been taken to factor out the common words and place them at the appropriate starting *sūtra.*

Thus for example, if the following are the expanded *sūtra*s

w_1 w_2 w_3 w_4
w_1 w_5 w_6
w_5 w_7 w_8
w_5 w_9
w_9 w_{10} w_{11} w_{12}

then the *sūtra* composer would put them as

w_1 w_2 w_3 w_4
w_5 w_6
w_7 w_8
w_9
w_{10} w_{11} w_{12}

factoring out the words that are repeated in the following *sūtra*s.

One would then reconstruct the original forms by borrowing the words from the earlier *sūtra*s. The context and the expectations provide the clues for borrowing. This process of borrowing or repeating the words from earlier *sūtra*s is known as *anuvṛtti. Piṅgaḷa* has used the *sūtra* style and also used *anuvṛtti. Kedār Bhaṭṭa*'s *Vṛttaratnākara* contains *sūtra*s which are more verbose than that of *Piṅgaḷa*'s, and does not use *anuvṛtti.*

In what follows, we take up each of the *sūtra*s from *Piṅgaḷa*'s *chandaśāstra* and explain its meaning and express it in modern mathematical language. We also examine the corresponding *sūtra* from *Kedār Bhaṭṭa*'s *Vṛttaratnākara*, and compare the two algorithms.

2 Algorithms

The algorithms that are described in *Piṅgaḷa*'s and *Kedār Bhaṭṭa*'s work are

- *Prastāraḥ*: To get all possible combinations (matrix) of n binary digits,
- *Naṣṭam*: To recover the lost/missing row in the matrix which is equivalent of getting a binary equivalent of a number,
- *ūddiṣṭam*: To get the row index of a given row in the matrix that is same as getting the value of a binary number,
- *Eka-dvi-ādi-l-g-kriyā*: To compute $^{\mathrm{n}}\mathrm{C}_{\mathrm{r}}$, n being the number of syllables and r the number of laghus (or *gurus*),
- *Samkhyā*: To get the total number of n bit combinations; equivalent to computation of 2^{n},
- *Adhva-yoga*: To compute the total combinations of *chanda* (meters) ranging from 1 syllable to n syllables that is equivalent to computation of $\sum_{i=1}^{n} 2^i$.

In addition to these algorithms, later commentators discuss an algorithm to get the positions of the r laghus in the matrix showing all possible combinations of n laghu-gurus. The corresponding structure is known as *patākā prastāra*.

2.1 *Prastāraḥ*

We shall first give the *Piṅgaḷa*'s algorithm followed by the *Kedār Bhaṭṭa*'s.

2.1.1 *Piṅgaḷa*'s algorithm for *Prastāraḥ*

Prastāraḥ literally means expansion, spreading etc. From what follows it will be clear that by *prastāraḥ*, *Piṅgaḷa* is talking about the matrix showing all possible combinations of n laghu-gurus. We know that there are 2^{n} possible combinations of n digit binary numbers. So when we write all possible combinations, it will result into a 2^{n} * n matrix.

The *sūtras* in *Piṅgaḷa*'s *Chandaśāstra* are as follows:

dvikau glau	8.20
miśrau ca	8.21
pṛthaglā miśrāḥ	8.22
vasavastrikāḥ	8.23

2.1.2 Explanation

1. `dvikau glau 8.20`

   This *sūtra* means *prastāra* of 'one syllable(akṣara)' has 2 possible elements viz. 'G or L'. So the 2^1 * 1 matrix is

$$\begin{bmatrix} G \\ L \end{bmatrix}$$

In the boolean(0 -1) notation, if we put 0 for G and 1 for L, we get

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

2. `misrau ca` `8.21`

   'To get the *prastāra* of two syllables, mix the above 1 syllable *prastāra* with itself'. So we get 'G-L' mixed with 'G' followed by 'G-L' mixed with 'L'.

Table 1: Mixed with G

G	G	0 0
L	G	1 0

Table 2: mixed with L

G	L	0 1
L	L	1 1

This will result in the $2^2 * 2$ matrix shown in table 3.

Table 3: 2 syllable combinations

G	G	0	0
L	G	1	0
G	L	0	1
L	L	1	1

But in boolean notation we represent all possible 2-digit numbers, in ascending order of their magnitude, as in table 4.

The Table (4) is obtained by elementary transformation of exchange of columns from Table (3), or simply put it is just the mirror image of Table (4). Now the question is, why the ancient Indian notation is as in Table (3) and not as in (4).

This may be because of the practice of writing from LEFT to RIGHT. The characters uttered first are written to the left of those which are uttered later.

Table 4: 2 digit Binary numbers

0	0
0	1
1	0
1	1

3. `pruthaglā miśrāḥ 8.22`

   "To get the expansion of 3 binary numbers, again mix the G and L separately with the *prastāra* of 2-syllables". So we get a $2^3 * 3$ matrix as in table 5.

Table 5: 3 syllable *prastāra*

G-L-representation	GL-01-conversion	boolean notation
G G G	0 0 0	0 0 0
L G G	1 0 0	0 0 1
G L G	0 1 0	0 1 0
L L G	1 1 0	0 1 1
G G L	0 0 1	1 0 0
L G L	1 0 1	1 0 1
G L L	0 1 1	1 1 0
L L L	1 1 1	1 1 1

   Again note that the modern (boolean notation) and ancient Indian notations (GL-01-conversion) are mirror images of each other.

4. vasavastrikāḥ 8.23

This *sūtra* simply states that there are 8 (vasavaḥ) 3s (trikāḥ).

Thus the first of the 4 rules gives the terminating (or initial) condition. Second rule tells how to generate a matrix for 2 bits from that of 1 bit. The third rule states how to generate combinations for 3 bits, given combinations for 2 bits. Fourth rule describes the size of the matrix of 3 bits, and that's all. It is understood that this process (the 3$^{\text{rd}}$ rule) is to be repeated again and again to get matrices of higher order.

2.1.3 Recursive-ness of *prastāra*

To make it clear, we represent the matrix in step 1 as
$A^1_{2*1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.
Then the matrix in step 2 is

$$A^2_{4*2} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} A^1_{2*1} & 0_{2*1} \\ A^1_{2*1} & 1_{2*1} \end{bmatrix}.$$

where O_{m*n} is a matrix with all elements equal to 0 and 1_{m*n} is a matrix with all elements equal to 1.

Continuing further, the matrix in step 3 is

$$A^2_{4*2} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} A^2_{4*2} 0_{4*1} \\ A^2_{4*2} 1_{4*1} \end{bmatrix}.$$

The generalisation of this leads to

$$A^n_{2^n*n} = \begin{bmatrix} A^{n-1}_{2^{n-1}*(n-1)} 0_{2^{n-1}*1} \\ A^{n-1}_{2^{n-1}*(n-1)} 1_{2^{n-1}*1} \end{bmatrix}$$

We notice that the algorithm for generation of all possible combinations of n bit binary numbers is thus 'recursive'.

2.1.4 *Kedār Bhaṭṭ*'s algorithm for Prastāraḥ

Kedār Bhaṭṭ in his *Vṛttaratnākara* has given another algorithm to get the *prastāra* for a given number of bits. His algorithm goes like this:

```
pāde sarvagurau ādyāt laghu nyasya guroḥadhaḥ|
yathā-upari tathā seśam bhūyaḥkuryāt amum vidhim ||
ūne dadyāt gurūn eva yāvat sarve laghuḥbhavet |
prastāraḥayam samākhyātaḥchandoviciti vedibhiḥ||
```

"In the beginning all are gurus(G)(*pāde sarvagurau*). In the second line, place a laghu(L) below the first G of the previous line(*ādyāt laghu nyasya guroḥadhaḥ*). Copy the remaining right bits as in the above line(*yathā-upari tathā seśam*). Place Gs in all the remaining places to the left (if any) of the 1st G bit(*ūne dadyāt gurūn eva*). Repeat this till all of them become laghu(*yāvat sarve laghuḥbhavet*). This process is known as *prastāra*."

Here is an example, explaining the above algorithm:

Table 6: Kedar Bhaṭṭa's algorithm

G	G	G	start with all Gs
L	G	G	place L below 1^{st} G of above line, copy remaining right bits viz. G G as in the above line
G	**L**	G	place L below 1^{st} G of above line, copy remaining right bit viz. G as in the above line, and G in the remaining place to the left of L
L	L	G	place L below 1^{st} G of above line, copy remaining right bits viz. L G as in the above line
G	G	**L**	place L below 1^{st} G of above line and G in remaining places to the left
L	G	L	place L below 1^{st} G of above line, copy remaining right bits viz. G L as in the above line
G	**L**	L	place L below 1^{st} G of above line, copy remaining right bit viz. L as in above line and G in the remaining place to the left
L	L	L	place L below 1^{st} G of above line, copy remaining remaining right bits viz. L L as in the above line and stop the process since all are Ls.

If we compare *Piṅgaḷa*'s method with that of *Kedār Bhaṭṭa*'s, we note that the first one is a recursive, whereas the second one is an iterative one. Compare this with the well-known definitions of factorials in modern notation. We can define factorial in two different ways:

```
n! = n * (n-1)!
1! = 1

OR

n! = 1*2*3*...*n.
```

Thus if one uses the first definition to get a factorial of say 4, one needs to know how to get the factorial of 3; to get the factorial of 3, one in turn should know how to get factorial of 2, etc.
Similarly according to *Piṅgaḷa*'s algorithm, to write a *prastāra* for 4 syllables, one needs to write a *prastāra* for 3 syllables, and to do so in turn one should write a *prastāra* for 2 syllables, and so on.
On the other hand, using the second definition, one can get 4! just by multiplying 1,2,3 and 4. One need not go through the whole process of finding other factorials. Similarly *Kedār Bhaṭṭa* describes an algorithm where one can write the *prastāra* for say 4 syllables directly without knowing what the *prastāra* for 3 syllables is.
The algorithm to obtain *prastāra*, as given by *Piṅgaḷa*, is similar to the recursive definition of factorial, whereas the one given by *Kedār Bhaṭṭa*, is similar to the iterative definition of factorial.

2.2 *Naṣṭam*

In ancient days, the *prastāra* (or matrix) used to be written on the sand, and hence there was possibility of getting a row erased. The next couple of *sūtra*s (8.24 and 8.25) are to recover the lost (or disappeared or vanished) row (naṣṭa) from the matrix. If one knows *Kedār Bhaṭṭa*'s algorithm then the lost row can be recovered easily from the previous or the next one. But *Piṅgaḷa* did not have an iterative description and hence he has given a separate algorithm to find the 'lost' row. In different words, getting a 'lost' row is conceptually equivalent to getting guru-laghu combination (i.e. the binary equivalent) of the row index.

The *sūtra*s are as follows:

l-arddhe	(8.24)
sa-eke-ga	(8.25)

"In case the given number can be halved (without any remainder), then write **L**, else add one and then halve it and write **G**". For example, suppose we want to get the 'laghu-guru' combination for the fifth row of the 3-akṣara matrix. We start with the given row-index i.e. 5. Since it is an odd number, add 1 to it and write ‘G’. After dividing 5+1(=6) by 2, we get 3. Again this is an odd number, and hence we add 1 to it, and write ‘G’. After dividing 3+1(=4) by 2 we get 2. Since it is an even number we write ‘L’. Once we get desired number of bits (in this case 3), the process ends:

Table 7: *Naṣṭam*

5	->	(5+1)/2=3	G
3	->	(3+1)/2=2	G G
2	->	2/2=1	G G L

So the fifth row in the *prastāra* (matrix) of 3 bits is G G L. The algorithm may be written as a recursive function as follows:

```
Get_Binary(n) =
Print L ; Get_Binary(n / 2), if n is even,
Print G ; Get_Binary(n+1 / 2), if n is odd,
Print G; if n=1. (terminating condition)
```

Thus this algorithm gives a method to convert a binary equivalent of a given number.

2.2.1 Difference between *Piṅgaḷa*’s method and Boolean method

Let us compare this conversion with the modern method. The boolean method is illutrated in table 8.

Table 8: Boolean conversion to binary

5	remainder	
5 / 2 = 2	1	ˆ
2 / 2 = 1	0	|
1 / 2 = 0	1	—-> |

Hence the binary equivalent of 5 is 101. If we replace G by 0 and L by 1 in ‘G G L’ we get 0 0 1. We have seen earlier that the numbers in modern and Indian method are mirror images, so after taking the mirror image of ‘0 0 1’ we get ‘1 0 0’. Thus, by *Piṅgaḷa*’s method we get the equivalent of 5 as ‘1 0 0’ whereas by modern method, we get $5=101_2$. Why is this difference? This difference is attributed to the fact that the counting in *Piṅgaḷa*’s method starts with ‘1’. In other words, 1 is represented as ‘0 0 0’ in *Piṅgaḷa*’s method, and not as ‘0 0 1’.

Thus we notice two major differences between the *Piṅgaḷa*'s method and the modern representation of binary numbers viz.
in *Piṅgaḷa*'s system,

- as has been initially observed by Nooten[2], the numbers are written with the higher place value digits to the right of lower place value digits, and
- the counting starts with 1.

2.3 *ūddiṣṭam*

The third algorithm is to obtain position of the desired (*uddiṣṭa*) row in a given matrix, without counting its position from the top, i.e. to get the row index corresponding to a given combination of G and Ls. Thus this is the inverse operation of *naṣṭam*. Both *Piṅgaḷa* as well as *Kedār Bhaṭṭa* have given algorithms for *uddiṣṭam*.

2.3.1 *Piṅgaḷa*'s algorithm for *uddiṣṭam*

Two *sūtra*s viz. (8.26) and (8.27) from *Piṅgaḷa*'s *Chandaśāstra* describe this algorithm. The *sūtra*s are as follows:

pratilomaguṇam dviḥ-l-ādyam	(8.26)
tatāḥ-gi-ekaṁ jahyāt	(8.27)

We first see the meaning of these *sūtra*s followed by an example. The first *sūtra* states that in the reverse order(*pratiloma*), starting from the 1st laghu(*l-ādyam*), multiply(*guṇam*) by 2(*dviḥ*). The second *sūtra* states that while doing so(*tataḥ*) if you come across a guru(*gi*) syllable then, subtract one (*ekaṁ jahyāt*) (after multiplying by 2). Here we also note the use of *anuvṛtti*. The word *dviḥ* is not repeated in the following *sūtra*, but should be borrowed from the previous *sūtra*. Since it is not mentioned what the starting number should be, we start with 1.

We illustrate this with an example. Let the input sequence be 'G L G'. Table 9 describes the application of the above sutras.

Thus the row 'G L G' is in the 3rd position in the *prastāra* of 3 bits. It is clear that this set of rules thus gives the row index of a row in the *prastāra* matrix.

The algorithm may be written formally as in table 10.

This set of rules further can be extended to get the decimal value of a number in any base B as shown in table 11.

According to this algorithm, value of the decimal number 789 can be calculated as shown in the table 12. Thus the position of 789 in the decimal place value system (where 0 is the 1st number 1 is the 2nd and so on) is 790. Further

Table 9: *uddiṣṭam*

G	L	G	remark
	1		(start with 1^{st} L from the right, starting number 1)
	2		(multiply by 2)
2			(continue with the previous result i.e. 2)
4			(multiply by 2)
3			(subtract 1, since it is guru).

Table 10: algorithm for Base 2

S_i	= 1 where 1^{st} laghu occurs in the $i+1^{th}$ position from right.
S_{i+1}	= 2 * S_i if $i+1^{th}$ position has L, = 2 * S_i - 1 if $i+1^{th}$ position has G, where S_i denotes sum till i^{th} digits from the right.

Table 11: algorithm:for Base B

S_i	= 1 where 1^{st} non-zero digit occurs in the $i+1^{th}$ position. (The counting for i starts with 1, and goes from the right digit with highest place value to the lowest place value)
S_{i+1}	= B * S_i if $i+1^{th}$ position has B-1, = B * S_i - D'_{i+1}, otherwise, where D'_{i+1} stands for the B-1's complements of $i+1^{th}$ digit, and S_i denotes sum of i digits from the right.

Table 12: Example: base 10

S0	= 1
S1	= 10 * 1 - 2 (9's complement of 7)
	= 8
S2	= 8 * 10 - 1 (9's complement of 8)
	= 79
S3	= 79 * 10
	= 790.

note that S1(=8) gives the value of single digit 7, S2(=79) gives the index of 2 digits 78.

2.3.2 ***Kedār Bhaṭṭa*'s algorithm for *uddiṣṭam***

The *Kedār Bhaṭṭa*'s version of *uddiṣṭam* differs from that of *Piṅgaḷa*. *Kedār Bhaṭṭa*'s version goes like this:

uddiṣṭam dvigunān ādyān upari ankān samālikhet |
laghusthā ye tu tatra ankān taiḥ sa-ekaiḥ miśritaiḥ bhavet ||

"To get the row number corresponding to the given laghu guru combination, starting from the first, write double (of the previous one) on the top of each laghu-guru. Then all the numbers on top of laghu are added with 1. (Since the starting number is not mentioned, by default, we start with 1)."

We illustrate this with an example.

Let the row be 'G L L'.

We start with 1, write it on top of G. Then multiply it by 2, and write 2 on top of the next L, and similarly (2*2=) 4 on top of the last L.

Table 13: *uddisṫam*

1	2	4
G	L	L

Then we add all the numbers which are on top of L viz. 2 + 4. To this we add 1. So the row index of the given L-G combination is 7.

In boolean mathematical notation, 'G L L' stands for 1 1 0 (after mirror image). This is equivalent to decimal 6. The difference of 1 is attributed to the fact that row index is counted from 1, as pointed out earlier.

2.4 eka-dvi-ādi l-g kriyā

Kedār Bhaṭṭa's work describes explicit rules to get the number of combinations of 1L, 2L, etc. (1G, 2G, etc.) among all possible combinations of n L-Gs. In other words, it gives a procedure to calculate $^{n}C_{r}$. *Piṅgaḷa*'s *sūtra* is very cryptic and it is only through *Halāyudha*'s commentary on it, one can interpret the *sūtra*as a *meru* which resembles the Pascal's triangle. We first give *Kedār* Bhaṭṭa's algorithm followed by *Piṅgaḷa*'s.

2.4.1 *Kedār Bhaṭṭa*'s algorithm

The procedure for *eka-dvi-adi-l-g-kriyā* in *Vṛttaratnākara* is described as follows:

varṇān vṛtta bhavān sa-ekān auttarārdhayataḥ sthitān |
ekādikramataḥ ca etān upari-upari nikṣipet ||
upāntyataḥ nivartet tyajatan ekaikam ūrdhavataḥ|
upari ādyāt guroḥ evaṁ eka-dvi-ādi-l-g-kriyā ||

Whatever the given number of syllables is, write those many 1s from the left to right as well as from top to bottom. Then in the 1^{st} row, add the number in the top(previous) row to its left occupant, and continue this process leaving the last number. Continue this process for the remaining rows. The last number in the 1^{st} column stands for all gurus. The last number in the second column stands for one laghu, the one in the third column for two laghus, and so on.

We explain this algorithm by an example.

Let the number be 6. Write 6 1s horizontally as well as vertically as below. Elements are populated rowwise by writing the sum of numbers in immediately preceeding row and column.

Table 14: *Meru* aka Pascal's Triangle

	1	1	1	1	1	**1**
1	2	3	4	5	**6**	
1	3	6	10	**15**		
1	4	10	**20**			
1	5	**15**				
1	**6**					
1						

The numbers 1, 6, 15, 20, 15, 6, 1 give number of combinations with all gurus, one laghu, two laghus, three laghus, four laghus, five laghus, and finally all laghus.

We see the striking similarity of this expansion with the Pascal's triangle. This process describes the method of getting the number of combinations of r from n viz. $^{n}C_{r}$. This triangle is termed as *meru* (literal: hill) in the Indian literature.

2.4.2 ***Piṅgaḷa*'s algorithm**

Piṅgaḷa's *sūtra*s are

pare pūrṇam	(8.34)
pare pūrṇam iti	(8.35)

The sutra 8.34 literally means, "complete it using the two far ends *pare*". Only from the *Halāyudha*'s commentary it becomes clear that this *sūtra* means: Start with '1' in a cell. Below this cell draw two cells, and so on. Then fill all the cells which are at the far ends, in each row, by 1s. This results in figure 15.

Table 15: *Meru* construction step-1

							1							
						1		1						
					1				1					
				1						1				
			1								1			
		1										1		
	1												1	
1														1

Next *sūtra* says, complete a cell using above two cells, again filling the far end cells. Thus resulting in table 16. Repeating this process we get table 17, and we see that the repeatition leads to the building of meru, or pascal's triangle.

Table 16: *Meru* construction step-2

							1							
						1		1						
					1		2		1					
				1		3		3		1				
			1		4				4		1			
		1		5						5		1		
	1		6								6		1	
1		7										7		1

Table 17: *Meru* construction step-3

							1							
						1		1						
					1		2		1					
				1		3		3		1				
			1		4		6		4		1			
		1		5		10		10		5		1		
	1		6		15				15		6		1	
1		7		21						21		7		1

2.4.3 Bhaskarācharya's method of obtaining *meru*

There are other ways of obtaining this *meru* described in Indian literature. For example Bhaskarācharya-II - the twelfth century Indian mathematician - in his Līlāvati[5] gives following procedure for obtaining n$^{\text{th}}$ row of the meru.

ekādi-ekottarā aṅkā vyastā bhājyāḥ kramasthiteḥ|
paraḥ pūrveṇa saṅguṇyastatparastena tena ca ||112 ||

The numbers 1, 2, etc. placed in reverse order be divided by 1, 2, etc. in this order. The quotient be multiplied by the previous one, the next by previous one. These shall be the combinations of 1,2,3 ... (from a group of n things.)

Reverse	n	n-1	n-2	...	2	1
Direct	1	2	3	...	n-1	n
Quotient	$\frac{n}{1}$	$\frac{(n-1)}{2}$	$\frac{(n-2)}{3}$	...	$\frac{2}{(n-1)}$	$\frac{1}{n}$
Product	$\frac{n}{1}$	$\frac{n(n-1)}{1.2}$	$\frac{n.(n-1)(n-2)}{1.2.3}$	...	$\frac{n.(n-1)..3.2}{1.2..(n-2).(n-1)}$	$\frac{n.(n-1)...2.1}{1.2...(n-1).n}$

These are the combinations of *n* things taken 1,2,3 ... at a time.

Reverse	6	5	4	3	2	1
Direct	1	2	3	4	5	6
Quotient	$\frac{6}{1}$	$\frac{5}{2}$	$\frac{4}{3}$	$\frac{3}{4}$	$\frac{2}{5}$	$\frac{1}{6}$
Product	$\frac{6}{1}$	$\frac{6.5}{1.2}$	$\frac{6.5.4}{1.2.3}$	$\frac{6.5.4.3}{1.2.3.4}$	$\frac{6.5.4.3.2}{1.2.3.4.5}$	$\frac{6.5.4.3.2.1}{1.2.3.4.5.6}$

$$C_0 = 1$$
$$A_i = n - i$$
$$B_i = i + 1$$
$$C_{i+1} = C_i * A_i / B_i.$$

Or, in other words,

Table 18: Binary Coefficients

i	6	5	4	3	2	1	0
C	1	6	15	20	15	6	1
B		6	5	4	3	2	1
A		1	2	3	4	5	6

$$^{n}C_{0} = 1$$
$$^{n}C_{r+1} = {}^{n}C_{r} * (n-r)/(r+1).$$

This is another instance of recursive definition.

2.5 *Sankhyā*

Sankhyā stands for the number of possible combinations of n bits. *Piṅgaḷa* and *Kedār* give an algorithm to calculate 2^n, given n. The algorithms differ as in earlier cases. *Kedār Bhaṭṭa* uses the results of previous operations (*uddiṣṭam* and *eka-dvi-ādi-l-g-kriya*), whereas *Piṅgaḷa* describes a totally independent algorithm.

2.5.1 *Kedār Bhaṭṭa*'s algorithm for finding the value of *Sankhyā*

Kedār Bhaṭṭ gives the following *sūtra* in his sixth chapter of the book *Vṛttaratnākara*

`l-g-kriyānka sandohe bhavet sankhyā vimiśrite |`

`uddiṣṭa-anka samāhārāḥ saḥ ekaḥ vā janayed imām ||`

This *sūtra* says, one can get the total combinations in two different ways:
a) by adding the numbers of *eka-dvi-ādi-l-g-kriyā*, or
b) by adding the numbers at the top in the *uddiṣṭa kriyā* and then adding 1 to it.

So for example, to get the possible combinations of 6 bits,

- the numbers in the *eka-dvi-ādi-l-g-kriyā* are 1,6,15,20,15,6,1 (see table 14). Adding these we get
  1 + 6 + 15 + 20 + 15 + 6 + 1 = 64.
  Therefore, there are 64 combinations of 6 bits.
- The *uddiṣṭa* numbers in case of 6 bits are
  1,2,4,8,16,32
  and adding all these and then 1 to it, we get
  1+2+4+8+16+32+1 = 64.

From this it is obvious that *Kedār Bhaṭṭa* was aware of the following two well-known formulae.

$$2^{\mathrm{n}} = \sum_{r=0}^{n} {}^{\mathrm{n}}\mathrm{C}_{\mathrm{r}} \text{ (Sum of the numbers in } \textit{eka-dv-ādi-l-g-kriyā}\text{)}$$

and

$$2^{\mathrm{n}} = \sum_{i=0}^{n-1} 2^{i} + 1 \text{ (sum of } \textit{uddiṣṭa} \text{ numbers +1).}$$

2.5.2 *Piṅgaḷa*'s algorithm for finding the value of *sankhyā*

Piṅgaḷa's description goes like this:

dviḥarddhe	(8.28)
rūpe śūnyam	(8.29)
dviḥśūnye	(8.30)
tāvadardhe tadguṇitam	(8.31)

If the number is divisible by 2{*arddhe*}, divide by 2 and write 2{*dviḥ*}. If not, subtract 1{*rūpe*}, and write 0{*śūnyam*}. If the answer were 0{*śūnya*}, multiply by 2{*dviḥ*}, and if the answer were 2{*arddhe*}, multiply {*tad guṇitam*} by itself {*tāvad*}.

So for example, consider 8.

```
8
4 2  (if even, divide by 2 and write 2)
2 2  (if even, divide by 2 and write 2)
1 2  (if even, divide by 2 and write 2)
0 0 (if odd, subtract 1 and write 0).
```

Now start with the 2^{nd} column, from bottom to top.

```
0 1*2 = 2          (if 0, multiply by 2)
2 2^2 = 4          (if 2, multiply by itself)
2 4^2 = 16        (if 2, multiply by itself)
2 16^16 = 256   (if 2, multiply by itself).
```

This algorithm may be expressed formally as

```
power2(n)   =   [power2(n/2)] ^  2  if n is even,
            =   power2(n-1/2) * 2, if n is odd,
            =   1, if n = 0.
```

Note that the results after each call of the function are ‘stacked’ and may also be treated as ‘tokens’ carrying the information for the next action (whether to multiply by 2 or to square). It still remains unclear to the author which part of the *sūtra* codes information about 'stack'. Or, in other words, how does one know that the operation is to be done in reverse order? There is no information about this in the *sūtra*s anywhere either explicit or implicit. This algorithm of calculating n^{th} power of 2 is a recursive algorithm and its complexity is $O(\log_2 n)$, whereas the complexity of calculating power by normal multiplication is O(n). Knuth[6] has referred to this algorithm as a 'binary method' (Knuth, pp 399).

2.6 Adhvayoga

Piṅgaḷa's *sūtra* is

dviḥ dviḥ-ūnam tat antānām	8.32

This algorithm gives the sum (*yoga*) of all the chandas (*adhva*) with number of syllables less than or equal to n. The sutra literally means to get *adhvayoga*, multiply the last one (*tat antānām*) by 2 (*dviḥ*) and then subtract 2 (*dviḥūnam*). That is

$$\sum_{i=1}^{n} 2^i = 2^n * 2 - 2 = 2^{n+1} - 2$$

2.7 Finding the position of all combinations of r guru (laghu) in a *prastāra* of n bits

This is an interesting algorithm found only in commentaries on *Kedār Bhaṭṭa*'s work[4]. The algorithm has been given in Bhaskarācārya's Līlāvati. This algorithm tells us the the positions of combinations involving 1 laghu, 2 laghu, etc. in the n bit *prastāra*. For example, in the 2 bit *prastāra* shown in table (2), we see that there is only one combination with both Gs, and it occurs in the 1^{st} position. There are 2 combinations of 1G (or 1 L), and they occur at the 2^{nd} and the 3^{rd} positions. Finally there is only one combination of 2 Ls, and it occurs at the fourth position. The following algorithm describes a way to get these positions without writing down the *prastāra*.

2.7.1 Algorithm to get positions of r laghu(guru) in a *prastāra* of n laghu-gurus

We will give an algorithm to populate the matrix A such that the j^{th} column of A will have positions of the rows in *prastāra* with j laghus. It follows that the total number of elements in j^{th} column will be ${}^{n}C_{j}$.

1. Write down 1,2,4,8,...,2^n in the 1^{st} row.

   $A[0,i] = 2^i$, $0 \leq i \leq n$.

2. The 2^{nd} column of elements is obtained by the following operation:
   $A[1,i] = A[0,0] + A[0,i]$, $1 \leq i \leq n$, and $A[i,j] < 2^n$.

3. The remaining columns (3^{rd} onwards) are obtained as follows:
   For each of the elements A[k-1,j] in the k^{th} column, do the following:

```
A[k,m] = A[k-1,j] + A[0,i],  k ≤ i ≤  n+1,

          if A[k,m] does not occur in the so-far-populated matrix, and

          A[k,m] < 2^n,  and

          0 ≤ j ≤ nCj, and

          0  ≤ m ≤ nCl.
```

The 1^{st} column gives positions of rows with all gurus;
2^{nd} column gives position of rows with 1 laghu, and remaining gurus;
3^{rd} column gives position of rows with 2 laghu and remaining gurus, and so on.
The last column gives the position of rows with all laghus.

Following example will illustrate the procedure.
Suppose we are interested in the positions of different combinations of laghus and gurus in the *prastāra* of 5 bits. We start with the powers of 2 starting from 0 till 5 as the 1^{st} row.

```
1 2 4 8 16 32.
```

To get the 2^{nd} column, we add 1 (A[0,0]) to the remaining elements in the 1^{st} row (see table 19).

1	2	4	8	16	32	
	3					(1 + 2)
	5					(1 + 4)
	9					(1 + 8)
	17					(1 + 16).

Thus, the 2^{nd} column gives the positions of rows in the *prastāra* of 5 bits with 1 laghu and remaining (4) gurus.

To get the 3^{rd} column, we add 2 (A[1,0]) to the remaining elements(A[0,j]; j > 1) of the 1^{st} row (see table 20).

1	2	4	8	16	32	
	3	6				(2 + 4)
	5	10				(2 + 8)
	9	18				(2 + 16)
	17					.

We repeat this for other elements in the 2^{nd} column (A[i,1]; i>0) as in Table 21.

1	2	4	8	16	32	
	3	6				(2 + 4)
	5	10				(2 + 8)
	9	18				(2 + 16)
	17	7				(3 + 4)
		11				(3 + 8)
		19				(3 + 16)
		13				(5 + 8) [5+4=9 already exists in the matrix, and hence ignored]
		21				(5+16)
		25				(9+16) [9+4, 9+8 are ignored].

The 3^{rd} column gives positions of rows with 2 laghus in the 5 bit expansion. We repeat this procedure till all the columns are exhausted. The final matrix will be as in table 22.

1	2	4	8	16	32
	3	6	12	24	
	5	10	20	28	
	9	18	14	30	
	17	7	22	31	
		11	26		
		19	15		
		13	23		
		21	27		
		25	29		

Since this is in the form of *patākā*(which literally means a flag[2]), it is also called as *patākā prastāra*. Thus the Indian mathematicians have gone one step

[2]Indian flags used to be triangular in shape

ahead of the modern mathematicians and not only gave the algorithms to find $^{n}C_{r}$, but also have given an algorithm to find the exact positions of these combinations in the matrix of all possible combinations(*prastāra*) of n G-Ls.

3 Conclusion

The use of mathematical algorithms and of recursion dates back to around 200 B.C.. *Piṅgaḷa* used recursion extensively to describe the algorithms. Further, the use of stack to store the information of intermediate operations, in *Piṅgaḷa*'s algorithms is also worth mentioning. All these algorithms use a terminating condition also, ensuring that the recursion terminates. Recursive algorithms are easy to conceptualise, and implement mechanically. We notice the use of method of recursion and also the binomial expansion in the later works on mathematics such as *brahmasphoṭasiddhanta*[7] with commentary by *pruthudaka* on summing a geometric series, or *Bhaṭṭotpala*'s commentary on *bruhatsamhitā* etc. They present a mathematical model corresponding to the algorithm. However, the iterative algorithms are easy from user's point of view. They are directly executable for a given value of inputs, without requirement of any stacking of variables. Hence the later commentators such as *Kedār Bhaṭṭa* might have used only iterative algorithms. The *sūtra* style was prevalent in India, and unlike modern mathematics, the Indian mathematics was passed from generations to generations verbally, through *sūtra*s. Sutras being very brief, and compact, were easy to memorise and also communicate orally. However, it is still unexplored what features of Natural language like Sanskrit have Indian mathematicians used for mathematics as opposed to a specially designed language of modern mathematics that made the Indian mathematicians communicate mathematics orally effortlessly.

4 Acknowledgment

The material in this paper was evolved while teaching a course on "Glimpses of Indian Mathematics" to the first year students of the Integrated Masters course at the University of Hyderabad. The author acknowledges the counsel of her father A. B. Padmanabha Rao, Subhash Kak, and Gèrard Huet who gave useful pointers and suggestions.

References

[1] Seidenberg, A., *The Origin of Mathematics* in Archive for History of Exact Sciences, 1978

[2] Nooten, B. Van., Binary Numbers in Indian Antiquity, *Journal of Indian Philosophy* 21:31-50, 1993.

[3] Sharma, Anantkrishna, *Pingalāchārya praītam Chandah Śāstram*, Parimal Publications, Delhi, 2001.

[4] Sharma, Kedār Nath, *Vṛttaratnākara, nārāyanī, manimayī vyākhyādyayopetaḥ*, Chaukhamba Sanskrit Sansthan, Varanasi, 1986.

[5] Jha, Lakhanlal, *Līlavati of Srīmadbhāskarachārya*, Chaukhamba Vidyabhavan, Varanasi, 1979.

[6] Knuth, D. E., seminumerical algorithms, *The Art of Computer Programming* Vol. 2, 2nd ed.,Addison-Wesley, Reading, MA, 1981.

[7] Coolbrook, H. T., *Algebra with Arithmetic Mensuration from the Sanskrit of Brahmagupta and Bhaskara*, Motilal Banarasidas, 1817.