Computing Parity of Combinatorial Functions

Sung-Hyuk Cha

Computer Science Department, Pace University
1 Pace Plaza, New York, NY 10038 USA
scha@pace.edu

Abstract. Finding the parity of a complex combinatorial formula is an important problem and it can be found efficiently without computing the complex formula itself. Hence, this article provides concise formulae for popular combinatorial functions such as *exponentiation*, *factorial*, *k-permutation*, n^{th} *Fibonacci*, n^{th} *Lucas* number, *summation*, and *binomial coefficient* (*k-combination*). While trivial for most functions, computing the parity of *k-combination* is quite difficult. Here, an efficient $O(\log n)$ algorithm and its $O(\log n \log\log n)$ recursive definition for computing the parity of *k-combination* using a *Fractal*, *Sierpinski's Triangle* are presented.

1 Introduction

The *parity* of a positive integer *n* is whether *n* is *even* or *odd* as defined in the eqn (1) and can be simply validated by the *mod*2 function in the eqn (2), i.e., if *n* is divisible by 2, then *n* is even.

$$odd(n) = \begin{cases} 1 & \text{if } n = odd \\ 0 & \text{if } n = even \end{cases} \tag{1}$$

$$odd(n) = n \bmod 2 \tag{2}$$

Table 1. Parity addition and multiplication rules

Addition $a + b$		Multiplication $a \times b$	
Rules	examples	rules	examples
e + e = e	4 + 2 = 6	e × e = e	4 × 6 = 24
e + o = o	2 + 3 = 5	e × o = e	2 × 5 = 10
o + e = o	5 + 2 = 7	o × e = e	3 × 4 = 12
o + o = e	3 + 5 = 8	o × o = o	3 × 5 = 15

Table 1 shows the general parity addition and multiplication rules with examples. These rules can be expressed following eqns (3) and (4). The eqn (5) is the special case of the eqn (4).

$$odd(a+b) = odd\big(odd(a) + odd(b)\big) \tag{3}$$

$$odd(a \times b) = odd(a) \times odd(b) \tag{4}$$

$$odd(a^2) = odd(a \times a) = odd(a) \times odd(a) = odd(a) \tag{5}$$

When *a* and *b* are positive integers, $c = a \times b$ can be a very big integer. What the eqn (4) means that one can compute $odd(a \times b)$ without computing $a \times b$.

Computing two to a positive integer n^{th} power takes not only $\Theta(\log n)$ computational time, but also quite space to represent the big integer output value. But if only the parity is of interest, $odd(2^n) = 0$ is trivial as 2^n is always divisible by 2 as in the eqn (6).

$$odd(2^n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Most combinatorial functions involve large integer values as their outputs, e.g.,

$$\begin{aligned}
43^{43} &= 17343773367030267519903781288812032158308062539012091953077767198995507 \\
54! &= 230843697339241380472092742683027581083278564571807941132288000000000000 \\
F_{337} &= 1200465717339148966867852201394183214700595472755636266015963789244361 7 \\
{}_{101}P_{38} &= 475431666789691804700159779803229608976439607546940471507943424000000000 0 \\
{}_{253}C_{101} &= 41854504441608411815045847357499158065915895683265618713600764368888002 55
\end{aligned}$$

Similar to the 2^n case, the parity of numerous combinatorial functions such as above cases whose output is a big positive integer can be computed very efficiently without computing the complicated functions themselves and taking the *mod* 2.

The rest of the paper is organized as follows. Section 2 provides concise formulae for trivial cases such as *exponentiation*, *factorial*, *k-permutation*, n^{th} *Fibonacci*, n^{th} *Lucas* number, and *summation*. Section 3 gives an efficient $O(\log n \log\log n)$ algorithm for computing the parity of *k-combination* a.k.a. *binomial coefficient* using the *Sierpinski's Triangle*. Finally, section 4 concludes this work.

2 Trivial Parity of Popular Functions

The *exponentiation*, a^n is a product of n factors of a positive integer a as defined in the eqn (7) and the parity of a^n can be found using the eqn (8).

$$a^n = \prod_{i=1}^{n} a = \underbrace{a \times \cdots \times a}_{n} \tag{7}$$

$$odd(a^n) = \begin{cases} 1 & \text{if } n = 0 \\ odd(a) & \text{otherwise} \end{cases} \tag{8}$$

Inductive Proof: eqn (8):

Suppose $odd(a^n) = odd(a)$.

$$\begin{aligned}
odd(a^{n+1}) &= odd(a^n) \times odd(a) && \text{by eqn (4)} \\
&= odd(a) \times odd(a) && \text{by assumption} \\
&= odd(a) && \text{by eqn (5)}
\end{aligned}$$

■

$odd(2^n)$ in the eqn (6) and $odd(n^n)$ in the eqn (9) are special cases of the eqn (8) except for 0^0, a mathematically mysterious case which we shall not discuss here.

$$odd(n^n) = \begin{cases} ? & \text{if } n = 0 \\ odd(n) & \text{otherwise} \end{cases} \tag{9}$$

The *factorial* of a positive integer n is the product of all positive integers less than or equal to n as defined in the eqn (10) and its parity can be found using the eqn (11).

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ \prod_{i=1}^{n} i = 1 \times 2 \times \cdots \times n & \text{otherwise} \end{cases} \quad (10) \qquad odd(n!) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

Proof: eqn (11): $odd(n!) = 0$ if $n > 1$

$$n! = \prod_{i=1}^{n} i = \prod_{i=3}^{n} i \times 2 \times 1$$

Since $n!$ contains 2, it is always divisible by 2. ■

The *k-Permutation* is the *k*-th falling factorial power of *n* as defined in the eqn (12) and its parity can be found using the eqn (13).

$$P_k^n = {}_nP_k = n^{\underline{k}} = \frac{n!}{(n-k)!} = \prod_{i=n-k+1}^{n} i = \underbrace{n \times (n-1) \times \cdots \times (n-k+1)}_{k} \quad (12)$$

$$odd(P_k^n) = \begin{cases} 1 & \text{if } k = 0 \\ odd(n) & \text{if } k = 1 \\ 0 & \text{if } k > 1 \end{cases} \quad (13)$$

Proof: eqn (13)

If $k = 1$, $P_k^n = n$ and thus $odd(P_k^n) = odd(n)$

If $k > 1$, i.e., $P_k^n = n \times (n-1) \times \cdots \times (n-k+1)$,

either n or $(n-1)$ has to be even and thus $odd(P_k^n) = 0$. ■

Fibonacci number is defined recursively in the eqn (14) and its parity can be computed using the eqn (15). Table 2 provides some insights where gray cells are odd and white cells are even.

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases} \quad (14) \qquad odd(F_n) = \begin{cases} 0 & \text{if } n \bmod 3 = 0 \\ 1 & \text{otherwise} \end{cases} \quad (15)$$

Inductive Proof: eqn (15)

Let $n = 3p$

Base case $p = 0$, $odd(F_0 = 0) = 0$, $odd(F_1 = 1) = 1$, and $odd(F_1 = 1) = 1$

Suppose $odd(F_{3p}) = 0$, $odd(F_{3p+1}) = 1$, and $odd(F_{3p+2}) = 1$

Show $3(p+1)$ case: Show $odd(F_{3(p+1)}) = 0$, $odd(F_{3(p+1)+1}) = 1$, and $odd(F_{3(p+1)+2}) = 1$

$odd(F_{3p+3})$	$= odd(F_{3p+1}+F_{3p+2})$	by Fibonacci definition (14)
	$= odd(odd(F_{3p+1})+ odd(F_{3p+2}))$	by eqn (3)
	$= odd(1 + 1) = 0$	by assumption
$odd(F_{3p+4})$	$= odd(F_{3p+2}+F_{3p+3})$	by Fibonacci definition (14)
	$= odd(odd(F_{3p+2})+ odd(F_{3p+3}))$	by eqn (3)
	$= odd(1 + 0) = 1$	by assumption
$odd(F_{3p+5})$	$= odd(F_{3p+3}+F_{3p+4})$	by Fibonacci definition (14)
	$= odd(odd(F_{3p+3})+ odd(F_{3p+4}))$	by eqn (3)
	$= odd(0 + 1) = 1$	by assumption ■

Lucas number is defined recursively in the eqn (16) and its parity can be computed using the eqn (17). Proof for the eqn (17) is similar to one for the eqn (15).

Table 2. Parity examples of summation, Fibonacci, and Lucas sequences

n	Σi	Σi^2	Σi^3	F_n	L_n
1	1	1	1	1	1
2	3	5	9	1	3
3	6	14	36	2	4
4	10	30	100	3	7
5	15	55	225	5	11
6	21	91	441	8	18
7	28	140	784	13	29
8	36	204	1296	21	47
9	45	285	2025	34	76
10	55	385	3025	55	123
11	66	506	4356	89	199
12	78	650	6084	144	322
13	91	819	8281	233	521
14	105	1015	11025	377	843
15	120	1240	14400	610	1364
16	136	1496	18496	987	2207
17	153	1785	23409	1597	3571
18	171	2109	29241	2584	5778
19	190	2470	36100	4181	9349
20	210	2870	44100	6765	15127

$$L_n = \begin{cases} 2 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ L_{n-1} + L_{n-2} & \text{if } n > 1 \end{cases} \quad (16) \qquad odd(L_n) = \begin{cases} 0 & \text{if } n \bmod 3 = 0 \\ 1 & \text{otherwise} \end{cases} \quad (17)$$

Consider following popular *summations* with their respective polynomial expressions in eqns (18~20). Albeit their integer outputs can be easily computed in constant time, parity of them can be found even faster using the eqn (21) regardless of the positive integer exponent p.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \quad (18) \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \quad (19)$$

$$\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 \quad (20) \qquad odd\left(\sum_{i=1}^{n} i^p\right) = \begin{cases} 1 & \text{if } n \bmod 4 = 1 \text{ or } 2 \\ 0 & \text{otherwise} \end{cases} \quad (21)$$

Inductive Proof: eqn (21)
Let $n = 4q$
Base case $q = 0$,

$$odd(0^p) = 0,\ odd\left(\sum_{i=1}^{1} i^p = 1\right) = 1\ odd\left(\sum_{i=1}^{2} i^p = \sum_{i=1}^{1} i^p + 2^p\right) = odd\left(odd\left(\sum_{i=1}^{1} i^p\right) + odd(2^p)\right) = 1$$

$$odd\left(\sum_{i=1}^{3} i^p = \sum_{i=1}^{2} i^p + 3^p\right) = odd\left(odd\left(\sum_{i=1}^{2} i^p\right) + odd(3^p)\right) = 0$$

Suppose $odd\left(\sum_{i=1}^{4q} i^p\right) = 0, odd\left(\sum_{i=1}^{4q+1} i^p\right) = 1, odd\left(\sum_{i=1}^{4q+2} i^p\right) = 1, odd\left(\sum_{i=1}^{4q+3} i^p\right) = 0$

Show $4(q+1)$ case:

$$odd\left(\sum_{i=1}^{4q+4} i^p\right) = odd\left(\sum_{i=1}^{4q+3} i^p + (4q+4)^p\right) = odd\left(odd\left(\sum_{i=1}^{4q+3} i^p\right) + odd(4q+4)\right) = odd(0+0) = 0$$

$$odd\left(\sum_{i=1}^{4q+5} i^p\right) = odd\left(\sum_{i=1}^{4q+4} i^p + (4q+5)^p\right) = odd\left(odd\left(\sum_{i=1}^{4q+4} i^p\right) + odd(4q+5)\right) = odd(0+1) = 1$$

$$odd\left(\sum_{i=1}^{4q+6} i^p\right) = odd\left(\sum_{i=1}^{4q+5} i^p + (4q+6)^p\right) = odd\left(odd\left(\sum_{i=1}^{4q+5} i^p\right) + odd(4q+6)\right) = odd(1+0) = 1$$

$$odd\left(\sum_{i=1}^{4q+7} i^p\right) = odd\left(\sum_{i=1}^{4q+6} i^p + (4q+7)^p\right) = odd\left(odd\left(\sum_{i=1}^{4q+6} i^p\right) + odd(4q+7)\right) = odd(1+1) = 0$$

■

3 Parity of Binomial Coefficient

The *k-combination*, a.k.a. *binomial coefficient* is defined in the eqn (22). Unfortunately, the parity of binomial coefficient is not as trivial as ones in the previous section.

$$ {}_nC_k = C(n,k) = C_k^n = \binom{n}{k} = \frac{n!}{(n-k)!k!} \tag{22}$$

Computing the eqn (22) takes linear time, $\Theta(n)$ and one may have to compute $n!$ which is a much bigger integer than ${}_nC_k$. To minimize its computational time and space a little bit, one may use the algorithm in the eqn (23) which takes $\Theta(\min(n-k, k))$ time and $\min({}_nP_k, {}_nP_{n-k+1})$ space.

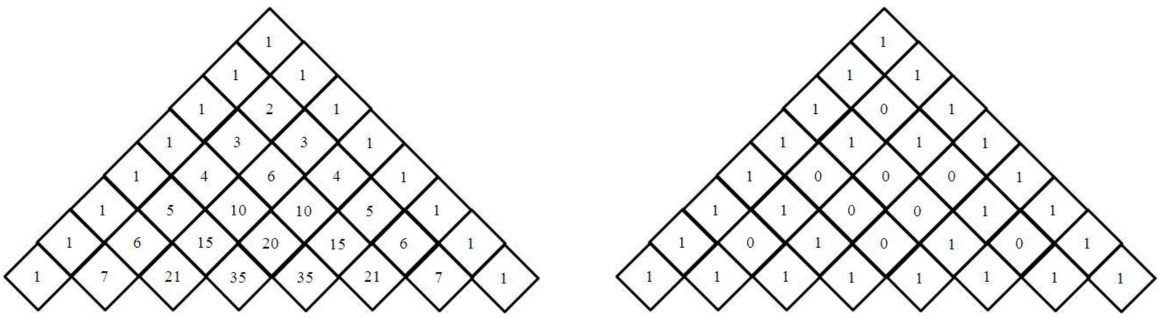
$$\binom{n}{k} = \begin{cases} \dfrac{P_k^n}{k!} = \dfrac{\overbrace{(n\times\cdots\times(n-k+1))}^{k}}{\underbrace{(k\times\cdots\times 1)}_{k}} & \text{if } k \le n-k \\[2ex] \dfrac{P_{n-k}^n}{(n-k)!} = \dfrac{\overbrace{(n\times\cdots\times(k+1))}^{n-k}}{\underbrace{((n-k)\times\cdots\times 1)}_{n-k}} & \text{if } n-k < k \end{cases} \tag{23}$$

If only the space is of concern, one can use the *Pascal's rule* in the eqn (24) which takes $O(n^2)$ or $\Theta(\min(n(n-k), nk))$ time but only ${}_nC_k$ space. Yet, an efficient algorithm for $odd({}_nC_k)$ without computing ${}_nC_k$ is of great interest here. One quick naïve algorithm is to extend the *Pascal's rule* [1] in the eqn (24) to its parity in the eqn (25).

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \tag{24}$$

$$odd\left(\binom{n}{k}\right) = odd\left(odd\left(\binom{n-1}{k-1}\right) + odd\left(\binom{n-1}{k}\right)\right) \tag{25}$$

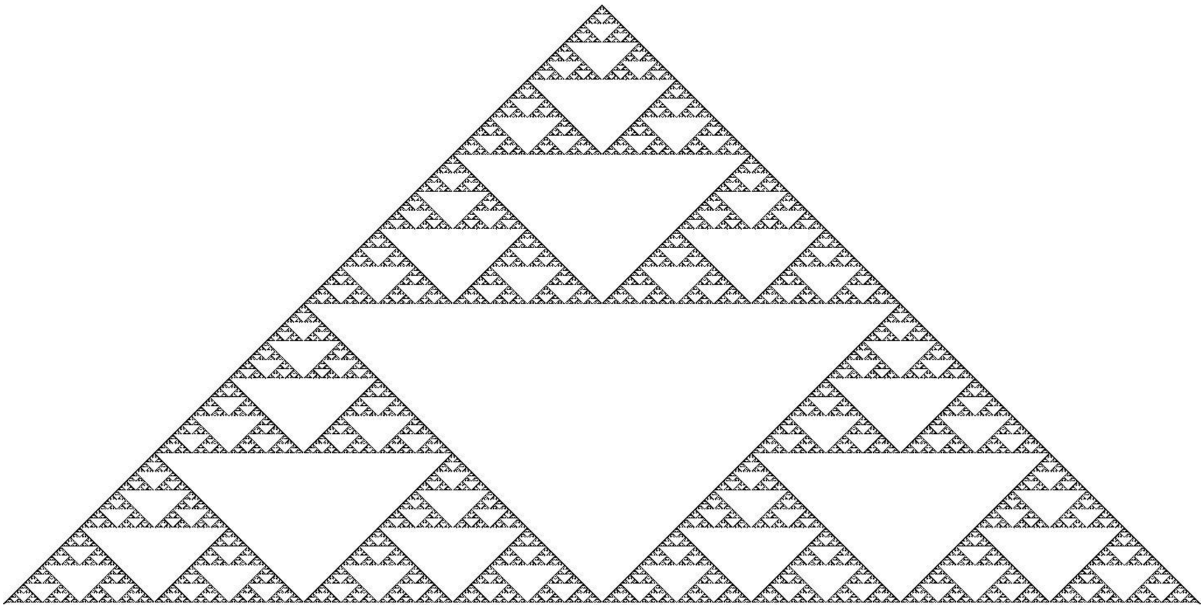
The parity triangle of the Pascal's triangle in Figure 1 (a) is given in Figure 1 (b). This naïve algorithm takes $O(n^2)$ or $\Theta(\min(n(n-k), nk))$ time.



(a) *Pascal's Triangle* (b) Parity of *Pascal's Triangle*

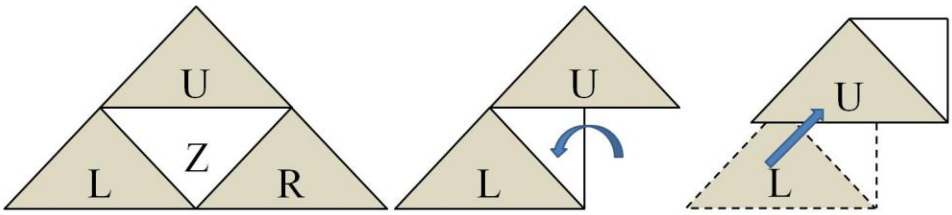
Figure 1. *Pascal's Triangle*

However, a careful observation of parity of Pascal's triangle gives a much efficient algorithm for $odd({}_nC_k)$. If all ones and zeros are colored with black and white, respectively, it becomes the beautiful fractal known as the *Sierpinski's triangle* [2] as shown in Figure 2. The *Sierpinski's triangle* presents a pattern of nested triangles. With this fractal recurring triangle patterns, the parity of binomial coefficient can be computed very efficiently.



n		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
2	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
4	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0
5	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0
6	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
7	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
8	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0
9	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0
10	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
11	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
12	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
13	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
16	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0
17	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0
18	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0

Figure 2. *Sierpinski's Triangle*.



(a) four triangles (b) symmetric folding (c) shift up

Figure 3. Figurative steps for the fast $odd({}_nC_k)$ algorithm.

The sketch of the recursive algorithm is given in Figure 3. Every $odd({}_nC_k)$ lies in an isosceles right triangle T with hypotenuse as its base. T consists of four isosceles right triangles, U, L, Z, and R as shown in Figure 3 (a). If $odd({}_nC_k)$ lies in U, U becomes T. Notice that Z is the upside-down triangle and contains all zeros. If $odd({}_nC_k)$ lies in Z (Zero zone), it should be solved in constant time. Notice also that $U = L = R$, and thus, if $odd({}_nC_k)$ lies in L or R, we can map its corresponding point in U and solve it recursively.

The *row-symmetry property* of binomial coefficient in the eqn (26) [1] extends to their parities as in the eqn (27) as well.

$$\binom{n}{k} = \binom{n}{n-k} \qquad (26) \qquad odd\left(\binom{n}{k}\right) = odd\left(\binom{n}{n-k}\right) \qquad (27)$$

If $odd({}_nC_k)$ lies in R, the eqn (27) can be utilized to map $odd({}_nC_k)$ to its symmetric part $odd({}_nC_{n-k})$ in L as shown in Figure 3 (b).

Notice that the base line of T contains all ones and it occurs at every $n = 2^b - 1$ where $b > 0$ as defined in the eqn (28).

$$odd\left(\binom{2^b-1}{x}\right) = 1 \text{ for any } x = 0 \sim 2^n - 1 \,\&\, b > 0 \qquad (28)$$

Similarly, the base line of the upside down U contains all zeros embraced by ones at both ends and it occurs at every $n = 2^b$ where $b > 0$ as defined in the eqn (29).

$$odd\left(\binom{2^b}{x}\right) = \begin{cases} 0 & \text{for any } x = 1 \sim 2^b - 1 \\ 1 & \text{if } x = 1 \text{ or } 2^b \end{cases} \qquad (29)$$

For any point at the n^{th} row, its base for the smallest T is $2^{\lceil \log n \rceil}$ and its base for the upside down triangle, Z is $2^{\lfloor \log n \rfloor}$. The log function is the *base-two-log* here. Any point in L can be shifted up to U by subtracting $2^{\lfloor \log n \rfloor}$ from n. as shown in Figure 3 (c).

The eqn (30) shows the recursive formula for $odd({}_nC_k)$.

$$odd\left(\binom{n}{k}\right) = \begin{cases} 0 & \text{if } k > n & \text{(zero zone)} \\ 1 & \text{if } k \le n \le 1 & \text{(base case)} \\ odd\left(\binom{n - 2^{\lfloor \log n \rfloor}}{k}\right) & \text{if } k \le \frac{n}{2} \,\&\, n > 1 & \text{(shift up)} \\ odd\left(\binom{n - 2^{\lfloor \log n \rfloor}}{n-k}\right) & \text{if } \frac{n}{2} < k \le n \,\&\, n > 1 & \text{(fold \& shiftup)} \end{cases} \qquad (30)$$

For example, ${}_{254}C_{239}$, k needs to be folded to $k = 254 - 239 = 15$. To shift up to a simpler problem, the floor of log254 needs to be computed first which is 7. And then $2^7 = 128$ must be subtracted from the original $n = 254$ resulting in 126. And thus a new simpler problem ${}_{126}C_{15}$ needs to be solved recursively. The recursive procedure is called only five times as illustrated in Figure 4 (a).

$$\underbrace{odd\left(\binom{254}{239}\right)}_{\substack{\text{fold \& shiftup}\\ \lfloor\log 254\rfloor=7\\ 2^7=128\\ n=254-128\\ k=254-239}} = \underbrace{odd\left(\binom{126}{15}\right)}_{\substack{\text{shiftup}\\ \lfloor\log 126\rfloor=6\\ 2^6=64\\ n=126-64}} = \underbrace{odd\left(\binom{62}{15}\right)}_{\substack{\text{shiftup}\\ \lfloor\log 62\rfloor=5\\ 2^5=32\\ n=62-32}} = \underbrace{odd\left(\binom{30}{15}\right)}_{\substack{\text{shiftup}\\ \lfloor\log 30\rfloor=4\\ 2^4=16\\ n=30-16}} = \underbrace{odd\left(\binom{14}{15}\right)}_{k>n} = 0$$

(a)

$$\underbrace{odd\left(\binom{1026}{875}\right)}_{\text{fold \& shiftup}} = \underbrace{odd\left(\binom{2}{151}\right)}_{k>n} = 0$$

(b) best case

$$\underbrace{odd\left(\binom{65537}{21511}\right)}_{\text{shiftup}} = \underbrace{odd\left(\binom{1}{21511}\right)}_{k>n} = 0$$

(c) best case

$$\underbrace{odd\left(\binom{253}{100}\right)}_{\text{shiftup}} = \underbrace{odd\left(\binom{125}{100}\right)}_{\text{fold \& shiftup}} = \underbrace{odd\left(\binom{61}{25}\right)}_{\text{shiftup}} = \underbrace{odd\left(\binom{29}{25}\right)}_{\text{fold \& shiftup}} = \underbrace{odd\left(\binom{13}{4}\right)}_{\text{shiftup}} = \underbrace{odd\left(\binom{5}{4}\right)}_{\text{fold \& shiftup}} = \underbrace{odd\left(\binom{1}{1}\right)}_{n\le 1} = 1$$

(d) worst case

Figure 4. Illustration of the fast $odd({}_nC_k)$ eqn (30) algorithm

The best case scenario is when the recursive procedure is called only once or twice as illustrated in Figure 4 (b) and (c). Zero zone cases belong to the best case. Notice that $x = \lfloor \log n \rfloor$ and 2^x must be computed within the procedure though. Both take Θ(log n) if efficient *divide and conquer* algorithms are used as given in eqns (31) and (32), respectively. Hence the best case computational complexity is Θ(log n).

$$\lfloor \log n \rfloor = \begin{cases} 0 & \text{if } n < 2 \\ \lfloor \log \frac{n}{2} \rfloor + 1 & \text{if } n \ge 2 \end{cases} \qquad (31) \qquad\qquad a^n = \begin{cases} a & \text{if } n = 1 \\ a^{\lfloor \frac{n}{2} \rfloor} \times a^{\lceil \frac{n}{2} \rceil} & \text{if } n > 1 \end{cases} \qquad (32)$$

In the worst case, the recursive procedure is called up to Θ(log n) times, e.g., Figure 4 (d).

Theorem 1: The worst case running time for the eqn (30) recursive algorithm is Θ(log$n$ loglog$n$).
Proof: Let $b = \lfloor \log n \rfloor \approx \log n$.
For each iteration, b and 2^b must be computed which takes Θ(log$b$).
There are up to b number of recursive calls in the worst case: <2^b, 2^{b-1},…, 2, 1>.
The total running time is log(b) + log(b–1) + … + log(1)

$$= \sum_{i=1}^{b} \log(i) = \Theta(b \log b) = \Theta(\log n \log\log n)$$

■

Therefore, the computational running time for the eqn (30) recursive algorithm is O(log$n$ loglog$n$).

The recursive formula is an excellent way to define the concept but a naïve direct implementation of a certain recursive definition as an algorithm often result in expensive computational time. Most famous examples include Fibonacci in the eqn (14) and Pascal’s rule in the eqn (24); direct implementation of the formulae result in exponential time complexity while Θ(n) and O(n^2) iterative algorithms are known, respectively.

Similarly, the eqn (30) gives an excellent and concise recursive formula for the parity of binomial coefficient but the direct implementation takes O(log$n$ loglog$n$) while a Θ(log$n$) algorithm is possible based on the eqn (30). The eqns (31) and (32) do not need to be computed in every recursive call but only once in the beginning. Consider the pseudo code for an iterative version for the eqn (30) whose computational running time is Θ(log$n$).

Algorithm I: *oddC*(*n*,*k*)
if $k = 0$,   return 1
$b \leftarrow 2^{\lfloor \log n \rfloor}$
while $n > 1$ & $k < n$
if $k > n/2$
$k \leftarrow n - k$
while $b > n$
$b \leftarrow b/2$
$n \leftarrow n - b$
if $k > n$, return 0
else return 1

Note that eqns (31) and (32) are executed only once in the line two of the pseudo code and thus Algorithm I takes $\Theta(\log n)$.

Corollary 1. Two leg sides of T are odd. Proof in the eqn (33)

Proof:

$$odd\left(\binom{n}{k}\right) = \binom{n}{k} = 1 \text{ if } k = 0 \text{ or } n \tag{33}$$

■

$$odd\left(\binom{276}{256}\right) = \underbrace{odd\left(\binom{20}{20}\right)}_{\text{fold \& shiftup}} = \underbrace{odd\left(\binom{4}{0}\right)}_{\text{fold \& shiftup}} = \underbrace{odd\left(\binom{0}{0}\right)}_{\text{shiftup}} = 1$$

(a) without right leg side termination

$$odd\left(\binom{276}{256}\right) = \underbrace{odd\left(\binom{20}{20}\right)}_{\text{fold \& shiftup}} = 1$$

(b) with right leg side termination

Figure 5. Illustration of the two leg side termination

For example in Figure 5, a naïve direct implementation algorithm of the eqn (30) may call some unnecessary recursive calls while the iterative Algorithm I terminates immediately as it contains the two leg side corollary 1. One may add Corollary 1 to the eqn (30) to improve it slightly but once again, the purpose of the eqn (30) is to provide as concise formula as possible. The author strongly recommends the Algorithm I rather than any recursive version.

Here are some further facts regarding the Pascal's parity triangle T. Notice that $C(2,1)$, $C(4,2)$, $C(6,3)$, $C(8,4)$, $C(10,5)$, $C(12,6)$, etc, are all even numbers.

Corollary 2. The *altitude* of T contain all even except for $n = 0$ as defined in the eqn (34).

$$odd\left(\binom{2m}{m}\right) = \begin{cases} 1 & \text{if } m = 0 \\ 0 & \text{otherwise} \end{cases} \tag{34}$$

Proof:

$$\binom{2m}{m} = \binom{2m-1}{m-1} + \binom{2m-1}{m} \quad \text{by Pascal definition (24)}$$

$$\binom{2m-1}{m-1} = \binom{2m-1}{m} \quad \text{by row symmetry property (27)}$$

$$\therefore\ odd\binom{2m}{m} = odd\left(2 \times \binom{2m-1}{m-1}\right) = 0$$

■

Corollary 3. The parity of the sum of *n*th row in *T* is always even as defined in the eqn (35).

$$odd\left(\sum_{k=0}^{n} odd\binom{n}{k}\right) = 0 \tag{35}$$

Proof: There are two cases.
Case 1: If n is odd, i.e., $n = 2m - 1$, the n^{th} row in T contains even number of possible values for k, i.e., $k = \{0,1,\ldots,n\}$. There are exactly $2m$ possible values for k. Suppose that the left half of the row contains x number of ones, then the other right half of the row must contain x number of ones because of the row symmetry property (27). Hence, the sum of the row is an even number.

Case 2: If n is even, i.e., $n = 2m$, the n^{th} row in T contains odd number of possible values for k, and the middle of the row is $k = m$. The middle one is even because of Corollary 2. Suppose that the left side of the row ($k = 0 \sim m - 1$) contains x number of ones, then the right half of the row ($k = m+1 \sim n$) must contain x number of ones as well because of the row symmetry property (27). Hence, the sum of the row is an even number. ■

Corollary 4. The parity of the row sum of bionomial coefficient is even as defined in the eqn (36).

$$odd\left(\sum_{k=0}^{n} \binom{n}{k}\right) = 0 \tag{36}$$

Proof: The *row-sum property* of binomial coefficient [1] is given in the eqn (37).

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n \tag{37}$$

$$\therefore odd\left(\sum_{k=0}^{n} \binom{n}{k}\right) = odd\left(2^n\right) = 0$$ ■

4 Conclusions

In this article, concise parity formulae for *exponentiation*, *factorial*, *k-permutation*, n^{th} *Fibonacci*, n^{th} *Lucas* number, *summation*, and *binomial coefficient* (*k-combination*) are studied and summarized in Table 3. While computing their parities takes constant time for most of combinatorial functions, no constant time formula is known for the binomial coefficient. Here, an efficient and concise recursive formula is presented.

It was also shown that the naïve direct implementation of the recursive formula for the binomial coefficient takes $O(\log n \log\log n)$. An $O(\log n)$ iterative version is also presented. Note that loglog$n$ grows extremely slow, e.g.,

loglog(256) = 3.
loglog(65535) = 4.
loglog(68719476736) = 5.
loglog(18446744073709551616) = 6
loglog(340282366920938463463374607431768211456) = 7
loglog(115792089237316195423570985008687907853269984665640564039457584007913129639936) = 8

Hence, the difference between two implementations should not be notable.

Table 3. Summary of formulae for combinatorial functions

$odd(a^n)=\begin{cases}1 & \text{if } n=0\\ odd(a) & \text{otherwise}\end{cases}$	$odd(n!)=\begin{cases}1 & \text{if } n=0 \text{ or } 1\\ 0 & \text{otherwise}\end{cases}$
$odd(P_k^n)=\begin{cases}1 & \text{if } k=0\\ odd(n) & \text{if } k=1\\ 0 & \text{if } k>1\end{cases}$	$odd\left(\sum_{i=1}^{n} i^p\right)=\begin{cases}1 & \text{if } n \bmod 4=1 \text{ or } 2\\ 0 & \text{otherwise}\end{cases}$
$odd(F_n)=\begin{cases}0 & \text{if } n \bmod 3=0\\ 1 & \text{otherwise}\end{cases}$	$odd(L_n)=\begin{cases}0 & \text{if } n \bmod 3=0\\ 1 & \text{otherwise}\end{cases}$
$odd\left(\binom{n}{k}\right)=\begin{cases}0 & \text{if } k>n\\ 1 & \text{if } k\le n\le 1\\ odd\left(\binom{n-2^{\lfloor \log n\rfloor}}{k}\right) & \text{if } k\le \frac{n}{2} \,\&\, n>1\\ odd\left(\binom{n-2^{\lfloor \log n\rfloor}}{n-k}\right) & \text{if } \frac{n}{2}<k\le n \,\&\, n>1\end{cases}$	

References

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms*, 2nd ed., MIT Press, 2001
2. Michael F. Barnsley, *SuperFractals*, Cambridge University Press 2006