On Parity based Divide and Conquer Recursive Functions

Sung-Hyuk Cha *†

Abstract—**The parity based divide and conquer recursion trees are introduced where the sizes of the tree do not grow monotonically as n grows. These non-monotonic recursive functions called $fog_k(n)$ and $\tilde{fog}_k(n)$ are strictly less than linear, $o(n)$ but greater than logarithm, $\Omega(\log n)$. Properties of $fog_k(n)$ such as non-monotonicity, upper and lower bounds, etc. are examined and proven. These functions are useful to analyze computational complexities of certain algorithms, especially problems of finding various properties of k-ary divide and conquer trees or size balanced k-ary trees. Several integer sequences based on the divide and conquer recursive relations are newly discovered as well.** ***Keywords: divide and conquer, Non-monotonic growth function, Analysis of Algorithms***

1 Introduction

In the analysis of algorithms, the computational time complexity functions, $T(n)$, are often limited to monotonically increasing [1] or eventually non-decreasing [2] growth functions, i.e., $T(n_1) \leq T(n_2)$ if there exist n_0 such that $n_0 < n_1 < n_2$. *Linear* and *logarithm* functions are examples of monotonically increasing functions where their *asymptotic relationships* can be discussed. Here some non-monotonic functions are introduced to analyze computational complexities of certain algorithms.

The *divide and conquer* technique, which solves problems by breaking them into two or more smaller subproblems, is one of most popular algorithm design techniques [1]. This technique produces the implicit *size balanced k-ary* tree [3] whose sizes of its children trees are the same or differ by only one. This *divide and conquer k-ary tree* gives intriguing integer sequences with regard to its input size n [3]. *Neil Sloane*'s *Online Encyclopedia of Integer Sequences* [4] contains a zoo of *divide and conquer integer sequences*. Yet, several new divide and conquer integer sequences generated from the non-monotonic recursive functions are discovered in this article.

Most classical recursive divide and conquer algorithms have their computational time complexities in a standard recursive form given in (1) where a is the number of subproblems and n/b is the size of the sub-problems.

$$T(n) = aT(\frac{n}{b}) + f(n) \tag{1}$$

These typical recursive divide and conquer algorithms form a full a-ary recursion tree and the asymptotically equivalent growth function can be determined by the *Master theorem* [1].

For example, consider the problem of finding the value of the nth power of c, c^n where n is a positive integer. Two different divide and conquer recursive algorithms given in (2) and (3) can solve the problem.

$$pow(c,n) = \begin{cases} c, & \text{if } n = 1 \\ pow(c, \lceil \frac{n}{2} \rceil) \times pow(c, \lfloor \frac{n}{2} \rfloor), & \text{if } n > 1 \end{cases} \tag{2}$$

$$pow(c,n) = \begin{cases} c, & \text{if } n = 1 \\ pow(c, \frac{n}{2})^2, & \text{if } n \text{ is even} \\ pow(c, \lfloor \frac{n}{2} \rfloor)^2 \times c, & \text{if } n \text{ is odd} \end{cases} \tag{3}$$

The former algorithm in (2) always call two half sized subproblems which result in the full binary divide and conquer recursion tree as shown in Figure 1 (a). The latter algorithm in (3) always call only one subproblem of the half size which results in the unary divide and conquer recursion tree as shown in Figure 1 (b). The algorithm in (3) is called the *"binary method"* which appeared before 200 B.C. [5, 6]. Assuming that multiplication operation takes a constant time, the computational time complexities for algorithms in (2) and (3) have the standard recursive forms, $T(n) = 2T(n/2) + 1$ and $T(n) = T(n/2) + 1$, respectively. They are asymptotically linear $\Theta(n)$ and logarithm $\Theta(\log n)$, respectively, which can be trivially shown by the *Master Theorem* [1].

Consider an algorithm in (4) which calls one subproblem if n is even but calls twice if n is odd.

$$pow(c,n) = \begin{cases} c, & \text{if } n = 1 \\ pow(c, n/2)^2, & \text{if even} \\ pow(c, \lceil \frac{n}{2} \rceil) \times pow(c, \lfloor \frac{n}{2} \rfloor), & \text{if odd} \end{cases} \tag{4}$$

The computational time complexity of the algorithm in (4) is strictly less than linear, $o(n)$ but greater than logarithm, $\Omega(\log n)$ intuitively as shown in Fig. 1 (c). Unfortunately, the *Master theorem* does not help to find the

*Manuscript received June 25, 2012; revised Auguest 1, 2012.
†S.-H. Cha is with the Computer Science Department, Pace University, New York, NY, 10038 USA Tel/Fax: 212-346-1253/1863 e-mail: scha@pace.edu (see http://csis.pace.edu/∼schа).

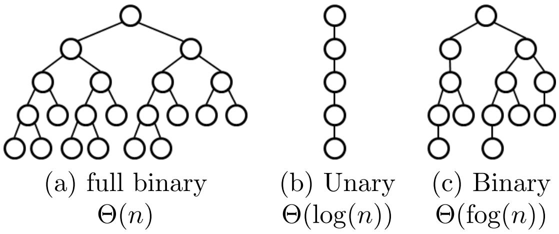


Figure 1: Three kinds of divide and conquer recursion tree.

exact asymptotic closed formula for this recursive relation in (4). Even more general divide and conquer recurrence solving theorem, such as the *Akra-Bazzi* method [7], cannot handle this simple and rather elementary divide and conquer relation, either.

Albeit the algorithms in (2) and (4) should not be used for the problem of evaluating integer powers, it is a good example to realize that there exists a recursive function that is strictly less than linear and greater than or equals to the logarithm as figuratively explained in Fig 1. This mysterious parity based divide and conquer recursive function shall be called $fog_k(n)$. In [3], the problem of finding the sum of heights of a *size balanced k-ary tree* or a *divide and conquer recursion tree* was considered. This article shall prove that the computational complexity of solving this problem in [3] is $\Theta(fog(n))$ and investigate other problems whose computation complexities are $\Theta(fog(n))$.

The subsequent sections are constructed as follows. The section 2 formally defines the $fog_k(n)$ and its properties are examined and proven. Another similar *leave-one-out divide and conquer* recursive function called $\tilde{f}og_k(n)$ is introduced as well. In section 3, various problem examples whose computational running time complexities are either $\Theta(fog_k(n))$ or $\Theta(\tilde{f}og_k(n))$ are given. Finally, the section 4 concludes this work.

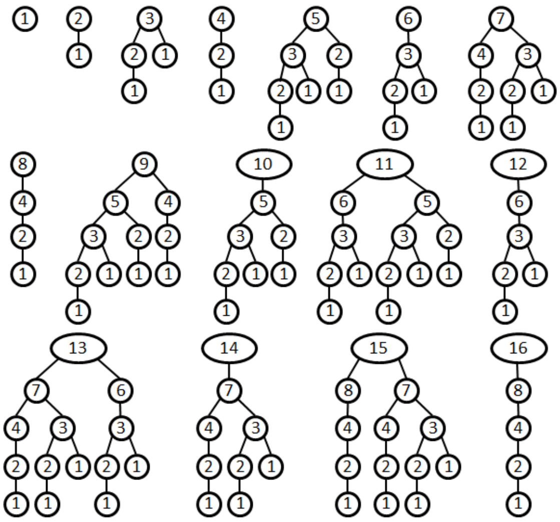
2 Divide and conquer recursive function

2.1 Definition $fog_k(n)$

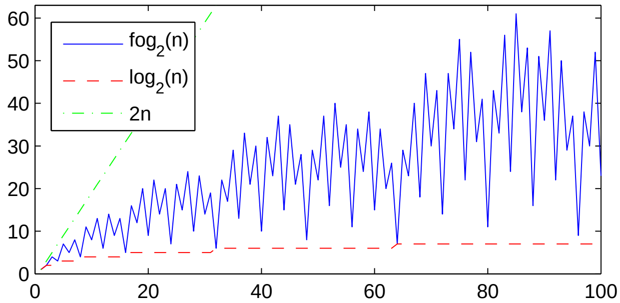
Computational running time complexity of the algorithm (4) can be represented as a recursion tree where the node has one or two children depending on its parity Fig. 2 (a) enumerates the first 16 recursion trees. Let the size of this binary recursion tree be $fog_2(n)$ as defined in (5).

$$fog_2(n) = \begin{cases} 1, & \text{if } n = 1 \\ fog_2(\frac{n}{2}) + 1, & \text{if } n\%2 = 0 \\ fog_2(\lceil \frac{n}{2} \rceil) + fog_2(\lfloor \frac{n}{2} \rfloor) + 1, & \text{if } n\%2 \neq 0 \end{cases} \tag{5}$$

$fog_2(n)$ is not monotonically growing function but fluctuates as shown in Fig 2 (b).



(a) $fog_2(n)$ recursion trees



(b) $fog_2(n)$ graph in comparison to n and $\log n$.

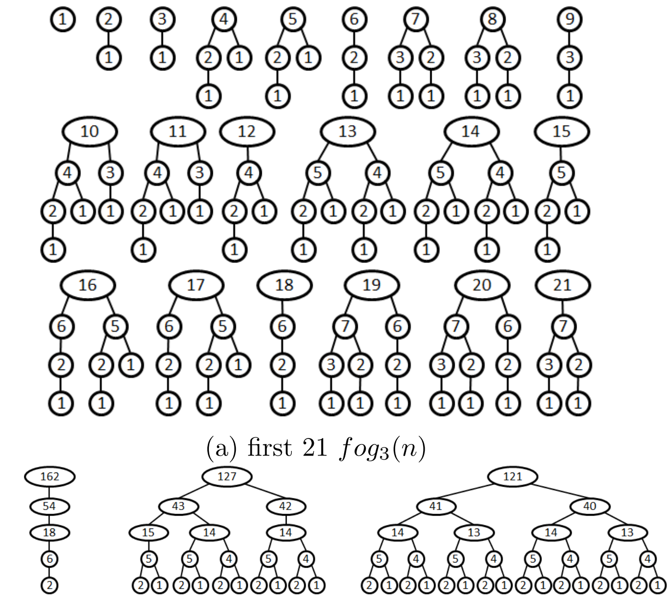
Figure 2: *$fog_2(n)$ recursion trees and graph.*

The concept in (5) can be generalized for the k-ary divide and conquer cases where each node has up to k children. The sizes of k-sub trees follow the integer partition into k balanced parts defined in (6).

$$\text{BIP}(n,k) = \Big(\overbrace{\underbrace{\left\lceil \frac{n}{k} \right\rceil, \dots, \left\lceil \frac{n}{k} \right\rceil}_{\tilde{k}=n\%k}, \left\lfloor \frac{n}{k} \right\rfloor, \dots, \left\lfloor \frac{n}{k} \right\rfloor}^{k}\Big) \tag{6}$$

For examples, $\text{BIP}(17,3) = (6,6,5)$ and $\text{BIP}(22,4) = (6,6,5,5)$. If n is divisible by k, all children have the unique size $\frac{n}{k}$. If n is not divisible by k, there are exactly two kinds of children, i.e., $\lceil \frac{n}{k} \rceil$ and $\lfloor \frac{n}{k} \rfloor$ as defined in (6).

The binary divide and conquer algorithms in (2) and (3) can be generalized to k-divide and conquer algorithms which have $\Theta(\log_k n)$ unary recursion tree and $\Theta(n)$ k-ary recursion tree, respectively. The algorithm in (4) can be also generalized to k-divide and conquer where it only calls one sub-problem if the size n is divisible by k or calls two sub-problems if not. This algorithm has a binary recursion tree regardless of k. The computational time



(a) first 21 $fog_3(n)$

(b) best - unary (c) typical - binary (d) worst case - full binary

Figure 3: *$fog_3(n)$ recursion trees.*

complexity of this algorithm can be defined as in (7).

$$fog_k(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2, & \text{if } n \leq k \\ fog_k(\frac{n}{k}) + 1, & \text{if } n\%k = 0 \\ fog_k(\lceil \frac{n}{k} \rceil) + fog_k(\lfloor \frac{n}{k} \rfloor) + 1, & \text{if } n\%k \neq 0 \end{cases} \tag{7}$$

For the example of ternary ($k = 3$) case, Fig 3 (a) shows the first 21 recursion trees. In the worst case, it has a full binary tree as shown in Fig 3 (d) while it has a unary tree in the base case as given in Fig 3 (b). The Table 1 lists first 100 integer sequences for $fog_2(n)$, $fog_3(n)$, and $fog_4(n)$.

2.2 Properties of $fog_k(n)$

The function in (7) is named as *fog* for two reasons. The first one is to be consistant with logarithm function introduced by Napier and the second reason is depicted in Fig 4 where $fog_k(n)$ integer values are ploted as dots instead of lines. This non-monotonic function looks like fogs.

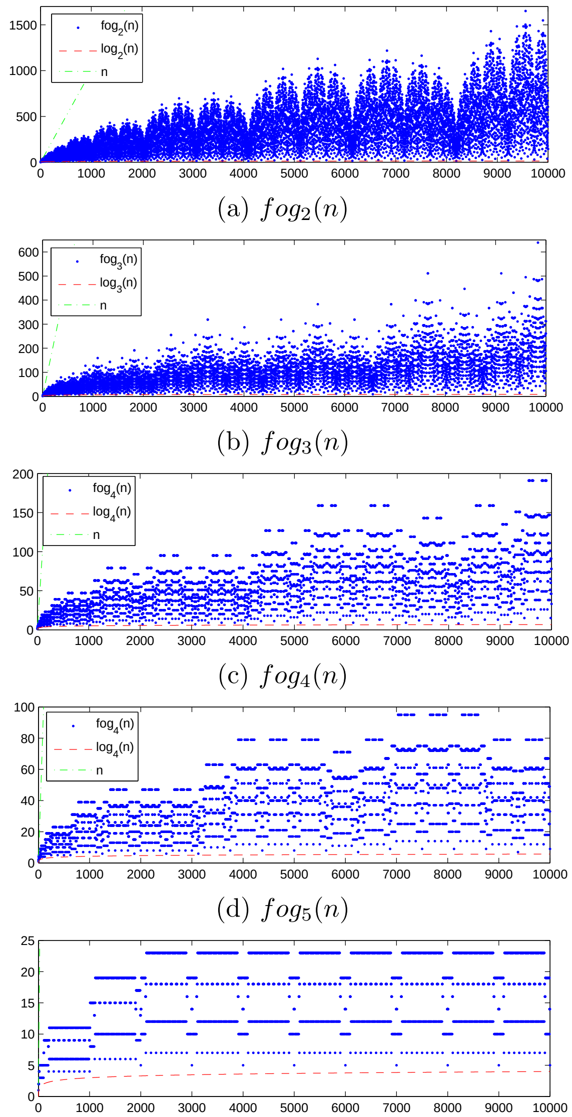
Property 1 *Non-monotonicity: $fog_k(n_1) \not\leq fog_k(n_2)$ if $n_1 < n_2$.*

The first property of $fog_k(n)$ is straightforward as shown in Figs 2∼4.

Another obvious property of $fog_k(n)$ is its lower bound.

Property 2 *The lower bound: $fog_k(n) = \Omega(\log_k(n))$.*

In the best case, the input size n is divisible by k and its sub-problem's size is also divisible by k all the way to the base case. This case is $\Theta(\log_k n)$. This best case scenario occurs at $n = k^m$ where m is a positive integer as depicted in Fig. 4.



(a) $fog_2(n)$

(b) $fog_3(n)$

(c) $fog_4(n)$

(d) $fog_5(n)$

(e) $fog_{10}(n)$

Figure 4: *various $fog_k(n)$ plots.*

The worst case or the upper bound of $fog_k(n)$ can be derived using the *Master theorem*.

Theorem 1 *The upper bound: $fog_k(n) = O(N^{\log_k 2})$.*

Proof: In the worst case, the input size n and its all sub-children's sizes are not divisible by k. As depicted in Fig 3 (d), it forms a full binary tree and thus $T(n) = 2T(n/k) + 1$. Using the *Master theorem* case 2, $T(n) = \Theta(n^{\log_k 2})$ ∎

Table 1: $\Theta(fog_k(n))$ Integer Sequences.

k	Integer sequence for $n = 1, \cdots, 100$
2	1, 2, 4, 3, 7, 5, 8, 4, 11, 8, 13, 6, 14, 9, 13, 5, 16, 12, 20, 9, 22, 14, 20, 7, 21, 15, 24, 10, 23, 14, 19, 6, 22, 17, 29, 13, 33, 21, 30, 10, 32, 23, 37, 15, 35, 21, 28, 8, 29, 22, 37, 16, 40, 25, 35, 11, 34, 24, 38, 15, 34, 20, 26, 7, 29, 23, 40, 18, 47, 30, 43, 14, 47, 34, 55, 22, 52, 31, 41, 11, 43, 33, 56, 24, 61, 38, 53, 16, 51, 36, 57, 22, 50, 29, 37, 9, 38, 30, 52, 23, $\cdots$
3	1, 2, 2, 4, 4, 3, 5, 5, 3, 7, 7, 5, 9, 9, 5, 8, 8, 4, 9, 9, 6, 11, 11, 6, 9, 9, 4, 11, 11, 8, 15, 15, 8, 13, 13, 6, 15, 15, 10, 19, 19, 10, 15, 15, 6, 14, 14, 9, 17, 17, 9, 13, 13, 5, 14, 14, 10, 19, 19, 10, 16, 16, 7, 18, 18, 12, 23, 23, 12, 18, 18, 7, 16, 16, 10, 19, 19, 10, 14, 14, 5, 16, 16, 12, 23, 23, 12, 20, 20, 9, 24, 24, 16, 31, 31, 16, 24, 24, 9, 22, $\cdots$
4	1 ,2 ,2 ,2 ,4 ,4 ,4 ,3 ,5 ,5 ,5 ,3 ,5 ,5 ,5 ,3 ,7 ,7 ,7 ,5 ,9 ,9 ,9 ,5 ,9 ,9 ,9 ,5 ,8 ,8 ,8 ,4 ,9 ,9 ,9 ,6 ,11 ,11 ,11 ,6 ,11 ,11 ,11 ,6 ,9 ,9 ,9 ,4 ,9 ,9 ,9 ,6 ,11 ,11 ,11 ,6 ,11 ,11 ,11 ,6 ,9 ,9 ,9 ,4 ,11 ,11 ,11 ,8 ,15 ,15 ,15 ,8 ,15 ,15 ,15 ,8 ,13 ,13 ,13 ,6 ,15 ,15 ,15 ,10 ,19 ,19 ,19 ,10 ,19 ,19 ,19 ,10 ,15 ,15 ,15 ,6 ,15 ,15 ,15 ,10, $\cdots$

For $k = 3$ and $k = 4$, $n^{\log_3 2} = n^{0.6309}$ and $n^{\log_4 2} = n^{0.5}$, respectively. The tigher upper bound for the binary case is given as follows.

Theorem 2 *The upper bound for $k = 2$: $fog_2(n) = O(N^{\log_2(\varphi)})$.*

Proof: An odd number n is always divided into odd and even parts. An even number n is either a sum of two smaller even numbers in the best case or two smaller odd numbers in the worst case. In the worst case, we have a *Fibonacci* tree as shown in Fig 5. In the standard divide and conquer form, $a = \frac{1f_h+2f_{h+1}}{f_h+f_{h+1}} = 1+\frac{f_{h+1}}{f_{h+2}} = 1+\frac{1}{\varphi} = \varphi$. Since $T(n) = \varphi T(n/2) + 1$ belongs to the case 1 in the *Master Theorem*, $T(n) = \Theta(N^{\log_2(\varphi)}) \approx \Theta(N^{0.6942})$ ▮

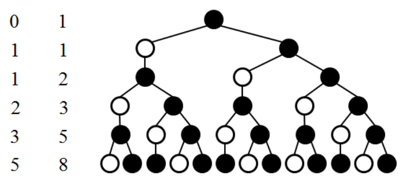


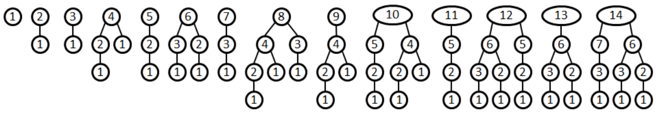
Figure 5: Worst case Fibonacci tree.

The following obvious inequality for the logarithm function, i.e., $\log_{k_1}(n) \leq \log_{k_2}(n)$ for $k_1 > k_2$, does not apply to the fog functions.

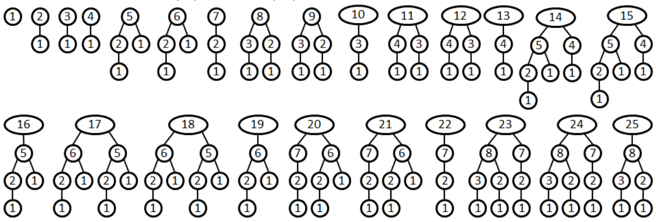
Fallacy 1 *$fog_{k_1}(n) \leq fog_{k_2}(n)$ for $k_1 > k_2$.*

Proof: While there are cases that the claim is true, e.g., $(fog_3(64) = 18) < (fog_2(64) = 7)$, there are also numerous counter examples like $(fog_3(96) = 16) \not\leq (fog_2(96) = 9)$, $(fog_4(63) = 9) \not\leq (fog_3(63) = 7)$, etc. Hence, $fog_{k_1}(n) \not\leq fog_{k_2}(n)$ for $k_1 > k_2$. ▮

Despite the Fallacy 1, fog is getting lower and fading as k increases, i.e., the lower and upper bounds of $fog_{k_1}(n)$ are lower than those of $fog_{k_2}(n)$ for $k_1 > k_2$ as shown in Fig 4. The following corollary 1 is an exceptional case of Fallacy 1.



(a) $\tilde{fog}_2(n)$ recursion trees.



(b) $\tilde{fog}_3(n)$ recursion trees.

Figure 6: *$\tilde{fog}_k(n)$ recursion trees.*

Corollary 1 *$fog_{k^p}(n) \leq fog_k(n)$ for a positive integer, p.*

Proof omitted.

2.3 Leave-one-out divide and conquer, $\tilde{fog}_k(n)$

There are two kinds of divide and conquer recursion trees. One is the standard one where n is the number of leaf nodes. The other is the *leave-one-out divide and conquer* where n is the total number of both internal and leaf nodes. In the *leave-one-out divide and conquer* tree, k number of subtrees have their sizes of either $\lceil (n-1)/k \rceil$ or $\lfloor (n-1)/k \rfloor$. The *median split tree* [8] is an example of the binary *leave-one-out divide and conquer* tree. In [3], k-ary *leave-one-out divide and conquer* are categorized as simply *k-ary size-balanced tree.*

The parity based divide and conquer function defined in (7) can be altered to analyze the leave-one-out divide and conquer algorithms. Let's denote this altered func-

Table 2: $\tilde{f}og_k(n)$ Integer Sequences.

k	Integer sequence for $n = 1, \cdots, 100$
2	1, 2, 2, 4, 3, 5, 3, 7, 5, 8, 4, 9, 6, 9, 4, 11, 8, 13, 6, 14, 9, 13, 5, 14, 10, 16, 7, 16, 10, 14, 5, 16, 12, 20, 9, 22, 14, 20, 7, 21, 15, 24, 10, 23, 14, 19, 6, 20, 15, 25, 11, 27, 17, 24, 8, 24, 17, 27, 11, 25, 15, 20, 6, 22, 17, 29, 13, 33, 21, 30, 10, 32, 23, 37, 15, 35, 21, 28, 8, 29, 22, 37, 16, 40, 25, 35, 11, 34, 24, 38, 15, 34, 20, 26, 7, 27, 21, 36, 16, 41, $\cdots$
3	1, 2, 2, 2, 4, 4, 3, 5, 5, 3, 5, 5, 3, 7, 7, 5, 9, 9, 5, 8, 8, 4, 9, 9, 6, 11, 11, 6, 9, 9, 4, 9, 9, 6, 11, 11, 6, 9, 9, 4, 11, 11, 8, 15, 15, 8, 13, 13, 6, 15, 15, 10, 19, 19, 10, 15, 15, 6, 14, 14, 9, 17, 17, 9, 13, 13, 5, 14, 14, 10, 19, 19, 10, 16, 16, 7, 18, 18, 12, 23, 23, 12, 18, 18, 7, 16, 16, 10, 19, 19, 10, 14, 14, 5, 14, 14, 10, 19, 19, 10, $\cdots$
4	1, 2, 2, 2, 2, 4, 4, 4, 3, 5, 5, 5, 3, 5, 5, 5, 3, 5, 5, 5, 3, 7, 7, 7, 5, 9, 9, 9, 5, 9, 9, 9, 5, 8, 8, 8, 4, 9, 9, 9, 6, 11, 11, 11, 6, 11, 11, 11, 6, 9, 9, 9, 4, 9, 9, 9, 6, 11, 11, 11, 6, 11, 11, 11, 6, 9, 9, 9, 4, 9, 9, 9, 6, 11, 11, 11, 6, 11, 11, 11, 6, 9, 9, 9, 4, 11, 11, 11, 8, 15, 15, 15, 8, 15, 15, 15, 8, 13, 13, 13, $\cdots$

tion as $\tilde{f}og_k(n)$.

$$\tilde{f}og_k(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2, & \text{if } n \le k+1 \\ \tilde{f}og_k(\frac{n-1}{k}) + 1, & \text{if } (n-1)\%k = 0 \\ \tilde{f}og_k(\lceil \frac{n-1}{k} \rceil) + \tilde{f}og_k(\lfloor \frac{n-1}{k} \rfloor) + 1, & \text{if } (n-1)\%k \neq 0 \end{cases} \tag{8}$$

Figure 6 shows the first 14 $\tilde{f}og_2(n)$ and 25 $\tilde{f}og_3(n)$ recursion trees and the the table 2 lists first 100 integer sequences for $\tilde{f}og_2(n)$, $\tilde{f}og_3(n)$, and $\tilde{f}og_4(n)$.

3 Applications

In [3], the problem of finding the sum of heights of a *size balanced k-ary tree*, $Z'_k(n)$ or a *leave-one-out divide and conquer recursion tree* was considered. Although the value can be computed in linear time by traversing the tree in the depth first order manner, it can be computed faster in $\Theta(\tilde{f}og_k(n))$ because we need to traverse only one sub-tree if all k sub-trees' sizes are the same or only two sub-trees otherwise. Here some other problems whose computational complexities are either $\Theta(fog_k(n))$ or $\Theta(\tilde{f}og_k(n))$ are examined.

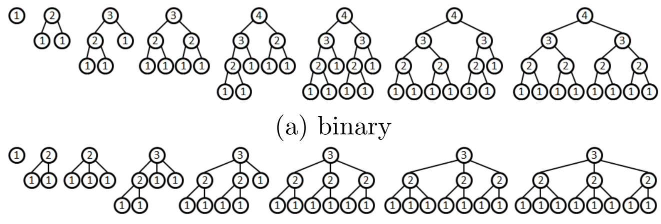
Let $Z_k(n)$ be a *standard divide and conquer recursion tree*. Let $H(Z_k(n))$ be the sum of each node's height in $Z_k(n)$ and it is defined recursively as in (9).

$$H(Z_k(n)) = \begin{cases} 0, & \text{if } n \le 1 \\ \lceil \log_k n \rceil + \tilde{k} \times H\left(Z_k(\lceil \frac{n}{k} \rceil)\right) \\ \quad +(k - \tilde{k}) \times H\left(Z_k(\lfloor \frac{n}{k} \rfloor)\right) & \text{otherwise} \end{cases} \tag{9}$$

Note that $\tilde{k} = n\%k$ as defined in (6). Table 3 shows the first 100 integer sequences of $H(Z_2(n))$ and $H(Z_3(n))$. The direct definition based recursive algorthm in (9) would take $\Theta(n^{\log_k 2})$. However, if the following condition in (10) is added to (9), the value can be computed in $\Theta(fog_k(n))$.

$$H(Z_k(n)) = \lceil \log_k n \rceil + k \times H\left(Z_k(\frac{n}{k})\right) \text{ if } n\%k = 0 \tag{10}$$

In *Neil Sloane*'s *Online Encyclopedia of Integer Sequences* [4], the sum of *inclusive heights* [3] of various ex-



(a) binary

(b) ternary

Figure 7: *inclusive heights of standard divide and conquer tree.*

plicit tree data structures are more popular than the sum of *exclusive heights* or simply heights. Hence, Fig 7 enumerates the first eight standard binary and ternary divide and conquer trees with each node containing the inclusive height. With a little modification to (9) and (10), the sum of the inclusive heights can be computed in $\Theta(fog_k(n))$ as well. Table 4 shows the first one hundred integer sequences of the sum of inclusive heights of binary and ternary trees.

Similarly, computing the several other properties of standard or leave-one-out divide and conquer trees would have their computational complexities of $\Theta(fog_k(n))$ or $\Theta(\tilde{f}og_k(n))$ naturally. For example, the path length of a rooted tree [9], $P(T)$ is another important property and $P(Z_k(n))$ and $P(Z'_k(n))$ can be computed in $\Theta(fog_2(n))$ and $\Theta(\tilde{f}og_2(n))$, respectively.

Strahler numbering of a binary tree, T, $S(T)$ is another important property of a binary tree. $S(T)$ in (11) can be computed in linear time if one uses the *postorder traversal* [10].

$$S(T) = \begin{cases} 0, & \text{if } T \text{ is empty} \\ \max(S(T_L), S(T_R)), & \text{if } S(T_L) \neq S(T_R) \\ S(T_L) + 1, & \text{if } S(T_L) = S(T_R) \end{cases} \tag{11}$$

Yet, $S(Z_2(n))$ and $S(Z'_2(n))$ can be computed much faster

Table 3: Sum of heights of standard divide and conquer tree integer sequences.

k	Integer sequence for $n = 1, \cdots, 100$
2	0, 1, 3, 4, 7, 9, 10, 11, 15, 18, 20, 22, 23, 24, 25, 26, 31, 35, 38, 41, 43, 45, 47, 49, 50, 51, 52, 53, 54, 55, 56, 57, 63, 68, 72, 76, 79, 82, 85, 88, 90, 92, 94, 96, 98, 100, 102, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 127, 133, 138, 143, 147, 151, 155, 159, 162, 165, 168, 171, 174, 177, 180, 183, 185, 187, 189, 191, 193, 195, 197, 199, 201, 203, 205, 207, 209, 211, 213, 215, 216, 217, 218, 219, $\cdots$
3	0, 1, 1, 3, 4, 5, 5, 5, 5, 8, 10, 12, 13, 14, 15, 16, 17, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 22, 25, 28, 30, 32, 34, 36, 38, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 63, 67, 71, 74, 77, 80, 83, 86, 89, 91, 93, 95, 97, 99, 101, 103, 105, 107, 109, $\cdots$

Table 4: Sum of inclusive heights of standard divide and conquer tree integer sequences.

k	Integer sequence for $n = 1, \cdots, 100$
2	1, 4, 8, 11, 16, 20, 23, 26, 32, 37, 41, 45, 48, 51, 54, 57, 64, 70, 75, 80, 84, 88, 92, 96, 99, 102, 105, 108, 111, 114, 117, 120, 128, 135, 141, 147, 152, 157, 162, 167, 171, 175, 179, 183, 187, 191, 195, 199, 202, 205, 208, 211, 214, 217, 220, 223, 226, 229, 232, 235, 238, 241, 244, 247, 256, 264, 271, 278, 284, 290, 296, 302, 307, 312, 317, 322, 327, 332, 337, 342, 346, 350, 354, 358, 362, 366, 370, 374, 378, 382, 386, 390, 394, 398, 402, 406, 409, 412, 415, 418, $\cdots$
3	1, 4, 5, 9, 12, 15, 16, 17, 18, 23, 27, 31, 34, 37, 40, 43, 46, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 64, 69, 74, 78, 82, 86, 90, 94, 98, 101, 104, 107, 110, 113, 116, 119, 122, 125, 128, 131, 134, 137, 140, 143, 146, 149, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 186, 192, 198, 203, 208, 213, 218, 223, 228, 232, 236, 240, 244, 248, 252, 256, 260, 264, 268, $\cdots$

using (12) in $\Theta(fog_2(n))$ and $\Theta(\tilde{f}og_2(n))$, respectively.

$$S(Z(n)) = \begin{cases} 0, & \text{if } Z(n) \text{ is empty} \\ \max(S(Z(\lceil \frac{n}{2} \rceil)), S(Z(\lfloor \frac{n}{2} \rfloor))), & \text{if } n\%2 \neq 0 \\ S(Z(\frac{n}{2})) + 1, & \text{if } n\%2 = 0 \end{cases} \tag{12}$$

4 Conclusions

Both standard and leave-one-out divide-and-conquer recursion trees are pervasive in computer science. This article introduced the parity based divide and conquer recursion trees. The functions of sizes of standard and leave-one-out divide and conquer trees were denoted as $fog_k(n)$ and $\tilde{f}og_k(n)$, respectively. Properties and some applications of these functions were discussed. We strongly believe that there are a plethora of applications of these functions in various problems which use the divide and conquer algorithms.

Another contribution of this article is discovering new integer sequences. All integer sequences in Tables 1∼4 are surprisingly not currently in *Neil Sloane*'s *Online Encyclopedia of Integer Sequences* [4]. These are important and pervasive integer sequences which involve divide and conquer algorithms.

The divide and conquer recursion is one of the widely studied areas and *Master theorem* and *Akra-Bazzi* method attempt to generalize these recursive formulae. However, they cannot handle $fog_k(n)$ and $\tilde{f}og_k(n)$ functions. Studying the more generalized parity based recursive forms is one of the future works.

References

[1] T.–H. Cormen, C.–E. Leiserson, and R.–L. Rivest, *Algorithm,* MIT Press, Cambridge, Massachusetts, 1993

[2] D.–S. Malik and M.–K. Sen, *Discrete Mathematic Structures: Theory and Applications,* Thomson Course Technology, 2004

[3] S.-H. Cha, "On Integer Sequences Derived from Balanced k-ary trees," in *Proceedings of American Conference on Applied Mathematics*, Cambridge, MA, pp. 377-381, Jan 2012.

[4] N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. http://oeis.org/

[5] D.–E. Knuth, *The Art of Computer Programming, vol 2: Seminumerical Algorithms,* 2nd ed. Addison-Wesley, 1981

[6] B. Datta and A.–N. Singh, *History of Hindu Mathematics,* vol 1, Bombay, 1935

[7] M. Akra and L. Bazzi,"On the solution of linear recurrence equations," *Computational Optimization and Applications*, vol. 10(2), pp. 195–210, 1998.

[8] B.–A. Sheil, "Median Split Trees: A Fast Lookup Technique for Frequently Occurring Keys," *Comm. ACM*, vol. 21, n. 11, pp. 947–958, 1978.

[9] T.–C. Hu and K.–C. Tan, "Path Length of Binary Search Trees," *SIAM Journal on Applied Mathematics*, vol. 22, n. 2, pp. 225234, 1972.

[10] P. Kruszewski, "A note on the Horton-Strahler number for random binary search trees," *Information Processing Letters*, vol. 69, n. 1, pp. 4751, 1999.