Journal of Theoretical and Applied Information Technology
10th November 2014. Vol. 69 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TIME COMPLEXITY OF ALGORITHMSAND ITS DIFFERENCE EQUATION REPRESENTATION

[1]M.RAJU,[2]B.SELVARAJ and [3]M.THIYAGARAJAN
[1] Department of Science and Humanities, Nehru Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India – 641105.
[2]Dean, Department of Science and Humanities, Nehru Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India – 641105.
[3] Dean Research, Nehru Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India – 641105.
E-mail: [1]rajumurugasamy@gmail.com,[2]professorselvaraj@gmail.com and[3]mthiyagarajan40@gmail.com

ABSTRACT

The fundamental problem encountered in all applications of computer science, can be classifiedintothe area of searching, merging and sorting. In the analysis of large class of algorithms, we have to discuss the solution of general recurrences based on divide and conquer algorithms. Graph algorithms are studied on the building block heap and disjoint data structure. We investigate the complexity of both time and space in the implementation of these algorithms. Results are expressed in terms of functions of number of steps along with the data structure. We give in this paper the different expressions for these complexity arguments appearingas solutions of specific types of difference equation expressions.

Keywords: *Second Order Difference Equations, Forward Difference, Asymptotic And Oscillatory Behaviors, Algorithms,Complexity, Numerical Data.*

Subject classification 2010: 39A21, 11Y16.

1. INTRODUCTION

In our investigation, asymptotic, oscillatory and bounded solutions of general second order difference equations under advanced setup, we come across specific conditions on the special type of functions studied. This has given us to think of one-one correspondence will be the particular case of reciprocal equations of first type and even degree. Already we come across the sequence formed by the n^{th}degree of the roots of these reciprocal equations to represent the time complexity of algorithms. A systematic study has been undertaken by theoretical computer science as like Alfred V.Aho, John E.Hopcroft and Jeffrey D.Ullman[3] along with Donald E.Knuth[5] on the different classifications computer algorithms. In particular, the time complexity and the space complexity of the algorithms. There is no unified approach for presenting particular variety of algorithms regarding time complexity specifications. We have proved the results using the building blocks of data structure, queues and finite graphs. Here we plays them to come under especial difference equations solutions.

Algorithms can be evaluated by a variety of criteria. Most often we shall be interested in the rate of growth of the time or space required to solve larger and larger instances of a problem. We would like to associate with a problem an integer, called the size of the problem, which is a measure of the quantity of input data. The time needed by an algorithm expressed as a function of the size of a problem is called the time complexity of the algorithm. The limiting behavior of the complexity as size increases is called the asymptotic time complexity.It is the asymptotic complexity of an algorithm which ultimately determines the size of problems that can be solved by the algorithm. If an algorithm processes inputs of size n in time cn^2 for some constant c, then we say that the time complexity of that algorithm is $O(n^2)$. A function g(n) is said to be O(f(n)) if there exists a constant c such that $g(n) \leq cf(n)$ for all nonnegative values of n. One might suspect that the tremendous increase in the speed of calculations brought about by the advent of the present generation of the digital computers would decrease the importance of the efficient algorithms. However, just the opposite is true. As computers become faster and we can handle larger problems, it is the complexity of an algorithm that determines the increase in problem size that can be achieved with an increase in computer speed.In the search of design analysis of algorithms[3, Alfred V.Aho and et al.] represented

Journal of Theoretical and Applied Information Technology
10th November 2014. Vol. 69 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

in particular the complexity of algorithms of Fibonacci sequence and Search. These give the complexity of the corresponding problems in the form $A2^n$ and $B\left(\frac{3+\sqrt{5}}{2}\right)^n$ respectively. Brief sketch of the proofs of the complexity of the algorithms along with the difference equationssatisfied by them are presented,we combine these complexities to represent in difference equations having a connection with a standard reciprocal equation in this paper.This is an extension of the paper studied by B.Selvaraj and et al.[7].The relation suggested by this study paves a way to fit even order first type reciprocal equation and their solutions. Here we consider the second order neutral delay difference equationwith new conditions. R.P.Agarwal[1], R.P.Agarwal and et al.[2]. discussed the general theory of difference equations. Many references to some applications of the difference equations discussed by Walter G.Kelley and Allan C.Peterson[4].

This paper is organized as follows: In section 2, we consider the general form of the difference equation of order two and reciprocal equation of even degree,connection between asecond order difference equationand the corresponding reciprocal equation are given in section 3. Section 4deals with complexity of depth first-search algorithm. Last section gives the conclusion of our contribution.

2. THE GENERAL FORM OF THE DIFFERENCE EQUATION OF ORDER TWO AND RECIPROCAL EQUATION OF EVEN DEGREE

We consider second order neutral delay difference equations of the form

$$\Delta^2\left(a_n x_n - b_n x_{n-\tau} + c_n x_{n-\sigma}\right) + f\left(x_{n-l-2}, x_{n-l-3}, \Delta x_{n-l-4} + \Delta x_{n+l+1}\right) = 0, \quad (1)$$

where $a_n>0, b_n>0, c_n>0$, $\tau \geq 3$, $\sigma \geq \tau$, for n ∈N = {0, 1, 2, ...}, l ∈{−s, ..., 0}, s = max {τ, σ},Δ is the forward difference operator defined by $\Delta x_n = x_{n+1} - x_n$ and the continuous function $f: R^4 \rightarrow R$ is defined by

$$f(\alpha,\beta,\gamma,\delta) = p_n F(\alpha) + q_n G(\beta) + r_n H(\gamma) + s_n I(\delta),$$

where $p_n>0$, $q_n<0$, $r_n<0$, $s_n>0$, F, G, H and I are continuous functions F: R → R, G: R → R, H: R → R, I: R → Rsuch that $y_1F(y_1)>0$, for $y_1 \neq 0$, $y_2G(y_2)>0$, for $y_2 \neq 0$, $y_3H(y_3)>0$, for $y_3 \neq 0$, $y_4I(y_4)>0$, for $y_4 \neq 0$, respectively.

We use the following notations throughout, N = {0, 1, 2, ...}, the set of natural numbers including zero;

N (a) = {a, a + 1, a + 2, ...}, where a ∈N.

We consider general reciprocal equation of even degree of the form

$$x^n + \lambda_1 x^{n-1} + \lambda_2 x^{n-2} + ... + \lambda_{n-1} x + \lambda_n = 0, \quad (2)$$

where n=2m, m∈N-{0}.

Many authors [9, 10] have studied the cases of $p_i \equiv 0$ and *f*is increasing, the author [8] has studied the cases of $q_j \equiv 0$. Few authors [11] have studied the cases of $p_i \neq 0$ in the first order difference equations.

Definition 2.1: By a solution of equation (1), we mean a real sequence $\{x_n\}$ which is defined for all $k \geq \min_{k\in N(1)}\{\tau_k, \sigma_k\}$ and satisfies equation (1) for sufficiently large k ∈N (a), a ∈N. A nontrivial solution $\{x_n\}$ of equation (1) is saidto be nonoscillatory if it is either eventually positive or eventually negative, and otherwise it is oscillatory.An equation is oscillatory if all its solutions are oscillatory.

Definition 2.2: If α is a root of the equation (2), $\frac{1}{\alpha}$ must also be a root. Hence the roots of the reciprocal equation(2) occur in pairs.

3. CONNECTION BETWEEN A SECOND ORDER DIFFERENCE EQUATION AND THE CORRESPONDING RECIPROCAL EQUATION

Consider the second order neutral delay difference equation

$$\Delta^2\left(10x_n - 3x_{n-3} + 2x_{n-4}\right) + x_{n+2} - 2x_{n+1} - 9\Delta x_n + 2\Delta x_{n-3} = 0. \quad (3)$$

Here $a_n= 10, b_n= 3, c_n=2$, τ=3, σ=4, l = -4,

$$f\left(x_{n+2} - 2x_{n+1} - 9\Delta x_n + 2\Delta x_{n-3}\right) = x_{n+2} - 2x_{n+1} - 9\Delta x_n + 2\Delta x_{n-3},$$

$p_n=1$, $q_n=-2$, $r_n=-9$, $s_n=2$, $F(x_{n+2})=x_{n+2}$, $G(x_{n+1})= x_{n+1}$, $H(\Delta x_n)=\Delta x_n$, $I(\Delta x_{n-3})=\Delta x_{n-3}$.The corresponding reciprocal equation is given as

$$2x^6 - 9x^5 + 10x^4 - 3x^3 + 10x^2 - 9x + 2 = 0. \quad (4)$$

By solving the equation (4), we get the roots are as

$$x = 2, \frac{1}{2}, \frac{3\pm\sqrt{5}}{2}, \frac{-1\pm i\sqrt{3}}{2}.$$

Therefore, the solution of equation (3) is given as

Journal of Theoretical and Applied Information Technology
10th November 2014. Vol. 69 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$x_n = A_1 2^n + A_2 \frac{1}{2^n} + A_3\left(\frac{3+\sqrt{5}}{2}\right)^n + A_4\left(\frac{3-\sqrt{5}}{2}\right)^n + A_5 \cos n\theta + A_6 \sin n\theta,$$

where

$$A_5 = C_1 + C_2, A_6 = i(C_1 - C_2),$$

$$r = \sqrt{\left(\frac{-1}{2}\right)^2 + \left(\frac{\sqrt{3}}{2}\right)^2} = 1, \theta = \tan^{-1}\left(\frac{-1}{\sqrt{3}}\right) = -\frac{\pi}{3}.$$

4. COMPLEXITY OF DEPTH FIRST-SEARCH ALGORITHM

Section 1:

The Fibonacci sequence,defined by, F_0=0, F_1=1, $F_{n+2}=F_{n+1}+F_n$, n≥0 satisfy the condition that $\phi^{n-2} \le F_n \le \phi^{n-1}$ if n is a positive integer and if $\phi = \frac{1}{2}(1+\sqrt{5})$. We will see shortly that the quantity, ϕ, is intimately connected with the Fibonacci numbers.

Theorem 4.1.1. A number divides both F_m and F_n if and only if it is a divisor of F_d, where $d=gcd(m,n)$; in particular, $gcd(F_m, F_n)=F_{gcd(m,n)}$.

We define the generating function G(z) as follows:

$G(z)=F_0+F_1z+F_2z^2+F_3z^3\ldots.$

By the theorem 4.1.1., we have F_0=1, $F_1=F_2$=1, $F_i=F_{i-1}$, i≥3.

Therefore, $G(z)=z+z^2+2z^3+\ldots.$

$\Rightarrow \quad (1-z-z^2)G(z)=F_0+(F_1-F_0)z+(F_2-F_1-F_0)z^2+\ldots.$

$\Rightarrow G(z)=\frac{z}{1-z-z^2}.$

We can now manipulate G(z) and find out more about the Fibonacci sequence. The denominator $1-z-z^2$ is a quadratic equation with the two roots $\phi = \frac{1}{2}(1\pm\sqrt{5})$; after a little calculation we find that G(z) can be expanded by the method of partial fraction into the form

$$G(z) = \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z} + \frac{1}{1-\hat{\phi} z}\right), \text{ where}$$

$$\hat{\phi} = 1-\phi = \frac{1}{2}(1-\sqrt{5}).$$

The quantity $\frac{1}{1-\phi z}$ is the sum of the infinite geometric series $1+\phi z+\phi^2 z^2+\ldots$, we have

$$G(z) = \frac{1}{\sqrt{5}}\left(1+\phi z+\phi^2 z^2+\ldots-1-\hat{\phi} z-\hat{\phi}^2 z^2-\ldots.\right).$$

We now look at the coefficient of z^n, which must be equal to F_n, and we find that

$$F_n = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right).$$

Section 2

Procedure SEARCH(v)

```
Begin
1.  Mark v "old";
2.  For each vertex w on L[v] do
3.  If w is marked "new" then
        Begin
4.      Add (v,w) to T;
5.      SEARCH(w)
        End
End
```

Fig.4.2.1. Depth-first search.

Algorithm 4.2.1.Depth-first search of an undirected graph.

Input: A graph G=(V,E) represented by adjacency lists L[v], for v ∈ V.

Output: A partition of E into T, a set of tree edges, and B, a set of back edges.

Method: The recursive procedure SEARCH(v) in Fig.4.2.1. adds edges (v,w) to T if vertex w is first reached during the search by an edge from v. We assume all vertices are initially marked "new". The entire algorithm is as follows:

```
    Begin
6.  T ← φ:
7.  For all v in V do mark v "new";
8.  While there exists a vertex v in V marked
    "new" do
9.  SEARCH(v)
    end
```

All edges in E not placed in T are considered to be in B. Note that if edge (v,w) is in E, then w will be on L[v] and v will be on L[w]. Thus we cannot simply place edge (v,w) in B if we are at vertex v and vertex w is marked "old" since w might be the father of v.

Theorem 4.2.1. Algorithm 4.2.1.requires O(MAX(n,e)) steps on a graph with n vertices and e edges.

Proof: Line 7 and search for "new" vertices at line 8 require O(n) steps if a list of vertices is made and scanned once. The time spent in SEARCH(v), exclusive of recursive calls to itself, is proportional to the number of vertices adjacent to v. SEARCH(v) is called only once for a given v, since v is marked "old" the first time SEARCH(v) is

Journal of Theoretical and Applied Information Technology
10th November 2014. Vol. 69 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

called. Thus the total time spent in SEARCH is O(MAX(n,e)), and we have the theorem.

Lemma 4.2.1. If (v,w) is a back edge, then in the spanning forest v is an ancestor of w or vice versa.
Proof. Suppose without loss of generality that v is visited before w, in the sense that SEARCH(v) is called before SEARCH(w). Thus when v is reached,w is still labeled "new". All "new" vertices visited by SEARCH(v) will become descendants of v in the spanning forest. But SEARCH(v) cannot end until w is reached, since w is on the list L[v].

Lemma 4.2.2. Let G=(V,E) be a connected, undirected graph, and let S=(V,T) be a depth-first spanning tree for G. Vertex *a* is an articulation point of G if and only if either

1. *a* is the root and *a* has more than one son,or
2. *a* is not the root, and for some son s of *a* there is no back edge between any descendent of s (including s itself) and a proper ancestor of *a*.

Proof: It is easy to show that the root is an articulation point if and only if it has more than one son.

Suppose condition 2 is true. Let f be the father of *a*. By lemma 4.2.1.each back edge goes from a vertex to an ancestor of the vertex. Thus any back edge goes from a descendent v of s goes to an ancestor of v. By the hypothesis of the lemma the back edge cannot go to a proper ancestor of *a*. Hence it goes to *a* or to a descendant of s. Thus every path from s to f contains *a*, implying that *a* is an articulation point.

To prove the converse, suppose that *a* is an articulation point but not the root. Let x and y be the distinct vertices other than *a* such that every path in G between x and y contains *a*. At least one of x and y, say x, is a proper descendant of *a* in S, else there is a path in G between x and y using edges in T and avoiding. Let s be the son of *a* such that x is a descendant of s(perhaps x=s). Either there is no back edge between a descendant v of s and a proper ancestor w of *a*, in which case condition 2 is immediately true, or there is such an edge (v,w). In the latter situation we must consider two cases.
Case 1. Suppose that y is not a descendant of *a*. Then there is a path from x to v to w to y that avoids *a*, a contradiction.
Case 2. Suppose y is a descendant of *a*. Surely y is not a descendant of s, else there is a path from x to y that avoids *a*. Let s′ be the son of *a* such that y is a descendant of s′. Either there is no back edge between descendant v′ of s′ and a proper ancestor w′ of *a*, in which case condition 2 is immediately true, or there is such an edge (v′,w′). In the latter case there is a path from x to v to w to w′ to v′ to y that avoids *a*, a contradiction. We conclude that the condition 2 is true.

Algorithm 4.2.2. Finding biconnected components.
Input: A connected, undirected graph G=(V,E).
Output: A list of the edges of each biconnected component of G.
Method:

1. Initially set T to θ and COUNT to 1. Also, mark each vertex in V as being "new". Then select an arbitrary vertex v_0 in V and calSEARCHB(v_0) to build a depth-first spanning tree S=(V,T) and to compute LOW(v) for each v in V.
2. When vertex w is encountered at line 5 of SEARCHB, put edge (v,w) on STACK, a pushdown store of edges, if it is not already there. After discovering a pair (v,w) at line 10 such that w is a son of v and LOW[w]≥v, pop from STACK all edges up to and including (v,w). These edges form a biconnected component of G.

Theorem 4.2.2. Algorithm 4.2.2.correctly finds the biconnected components of G and requires O(e) times if G has e edges.
Proof: The proof that step1 requires O(e) times is a simple extension of that observation for SEARCH(Theorem 4.2.1.). Step2 examines each degree once, places it on a pushdown store, and subsequently pops it. Thus step2 is O(e).

For the correctness of the algorithm, Lemma 4.2.2.assures us that the articulation points are correctly identified. Even if the root is not an articulation point, it is treated as one in order to emit the biconnected component containing the root.

We must prove that if LOW[w]≥v, then when SEARCHB(w) is completed the edges above (v,w) on STACK will be exactly those edges in the biconnected component containing (v,w).This is done by induction on the number b of biconnected components of G. The basis, b=1, is trivial since in this case v is the root of the tree, (v,w) is the only tree edge out of v, and on completion of SEARCHB(w) all edges of G are on STACK.

Now, assume the induction hypothesis is true for all graphs with b biconnected components, and let G be a graph with b+1 biconnected components. Let SEARCHB(w) be the first call of SEARCHB to end with LOW(W)≥v, for (v,w) a

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

tree edge.Since no edges have been removed from STACK, the set of edges above (v,w) on STACK is the set of all edges incident upon descendants of w. It is easily shown that these edges are exactly the edges of the biconnected component containing (v,w). On removal of these edges from STACK, the algorithm behaves exactly as it would on the graph G′ that is obtained from G by deleting the biconnected component with edge (v,w). The induction step now follows since G′ has b biconnected components.

Note: When $e=2^n$, our result of complexity apply.

5. CONCLUTION AND FUTURE WORK

The time complexities of major types of algorithms are given usually in difference equation representation. Here a special type of difference equation satisfied by three asymptotic sequences may be takes as a representation of a complexity issues in the computation of complexity algorithms. In the future work, we give a generalize difference equation which will answer complexity issues in major types of algorithms studied under various topics of research in computational methods.

REFRENCES:

[1]. R.P.Agarwal, Difference Equations and Inequalities, Second edition, Marcel Dekker,New York, 2000.
[2]. R.P.Agarwal, Martin Bohner, Said R. Grace, DonalO'Regan, Discrete Oscillation Theory, Hindawi, New York, 2005.
[3]. Alfred V.Aho, John E.Hopcroft and Jeffrey D.Ullman, The design and analysis of computer algorithms, Third impression, Pearson education, India, 2008.
[4]. Walter G.Kelley and Allan C.Peterson, Difference Equations - An Introduction withApplications, 2nd edition, Academic Press, San Diego, 2001.
[5]. Donald E.Knuth, The art of computer programming-Fundamental Algorithms, Third edition, Pearson education, India, 2005.
[6]. T.K.ManicavachagomPillay, T.Natarajan and K.S.Ganapathy, Algebra, Eleventh edition, S.Viswanathan(Printers & Publishers) pvt. Ltd. India, 2002.
[7]. B.Selvaraj, M.Raju and M.Thiyagarajan, A novel difference equation representation for autoregressive time series, Journal of Theoretical and Applied Information Technology(JATIT), Accepted for publication in Vol. 67, SEP. 2014.
[8]. EwaSchmeidel, An application of measures of noncompactness in the investigation of boundedness of solutions of second-order neutral difference equations,April 4, 2013, Vol. 91, No. 1.http://www.advancesindifferenceequations.com/content/2013/1/91.
[9]. E.Thandapani and B.Selvaraj, Existence and Asymptotic Behavior of Non Oscillatory Solutions of Certain Nonlinear Difference Equations, Far East Journal of Mathematical Sciences (FJMS), 14 (1) 2004, pp. 9 – 25.
[10]. E.Thandapani and P.Sundaram, On the asymptotic and oscillatory behavior of solutions of second order nonlinear neutral difference equations, Indian Journal of pure and applied Mathematics, Vol. 26, No. 12, 1995, pp. 1149–1160.
[11]. Yu-Ping Zhao and Xi-Lan Liu, Asymptotic behavior for nonoscillatory solutions of nonlinear delay difference equations, International Journal of Difference Equations, Vol. 5, No. 2, 2010, pp. 266–271.