The Use Of The Computational Method Of Hamming Code Techniques For The Detection And Correction Of Computational Errors In A Binary Coded Data: Analysis On An Integer Sequence A119626.

Afolabi Godfrey[1], Ibrahim A.A[2] & Ibrahim Zaid[3]
[1]Department of Mathematics, Joda International School, P.M.B 1031, Kebbi State, Nigeria.
[2]Department of Mathematics, Sokoto State University, Nigeria.
[3]Department of Mathematics, Sokoto State University, Nigeria.

ABSTRACT:
This paper presents a review of the Computational method of Hamming codes techniques for detection and correction of computational errors in a binary coded data, analysis in an integer sequence A119626. This integer sequence is obtained from computing the difference in the number of inverted pairs of the first and the last cycles which in turn is obtained from a special (123)-avoiding permutation pattern. The computation in this paper was restricted to values of n = 1, 2, 3, 4, and 5 respectively. This paper simply considered the execution time (T) and rate (R) for any given time t of the Computational method of analysis based on the number of iterations (steps) involved in the general procedure of encoding, detection and correction of errors for all the values of n.

KEY WORDS: *Computational method, binary coded data, execution time (T) and rate (R), Hamming codes and integer sequence A119626.*

I. INTRODUCTION

It has been generally observed that in communication, which is the process of transmission and reception of data from one place to another at some distance or in computational analysis involving integer sequences are not without error(s). There are a number of reliable codes that can be used to encode data, that is, adding parity code to the original result (data) obtained so that any error(s) that may occur can be detected and corrected. Until this is done, communication or computational results obtained will not be efficient talk more of been reliable. According to [1], the binary coded data (binary number system) is widely used in digital communication systems. Although the binary number system has many practical advantages and it is widely used in digital computer, in many cases, it is convenient to work with the decimal number system, especially when the interaction between human beings and machine is extensive. This is because, most numerical data generated by humans is basically in terms of decimal numbers. Thus, to simplify the challenges encountered in communication (interaction) or computation between human and machine, several codes have been devised in which decimal digits are represented by sequences of binary digits that is, in binary coded data form.

Hamming code is one of such codes. It is a set of error-correction codes that can be used to detect and correct bit errors that can occur when data (information or results obtained) is transmitted, stored or computed as in [2]. Like other error-correction code, Hamming code makes use of the concept of parity and parity bits, which are bits that are added to data (say, k parity bits are added to an n-bit data to form a new word of $n + k$ bits), so that the validity and reliability of the data are being checked when it is read or after it has been transmitted or computed and received as data. Using more than one parity bit, an error-correction code can not only identify a single bit error in the data unit, but also its position in that data unit according to [2]. In a Hamming encoder, parity bits are decided so as to place a fixed parity on different combinations, which are then checked for. In the decoder, parity bits are set for the fixed parity combinations which were checked. The binary equivalence of this combination decides the position of the error. Then that particular bit is flipped (interchanged from either '0' to '1' or '1' to '0') to correct the erroneous bit. Hamming code is a single error correction code, that is, it can detect a single error and proceed to correct it. It can also detect multiple (double) errors only on the condition that no correction is attempted. Error correction may be avoided at certain cases where retransmission of the computed or sent information (data) is feasible and effective. But, error detection on the other hand, is a must in all cases. Once there is a deviation from the original information (data) computed or transmitted, the cause (error) of such deviation must be detected and corrected as the case may be. Error control therefore, is the major concern of this paper.

Hence, the analysis of the Computational method of Hamming code techniques for detection and correction of computational errors in a binary coded data in an integer sequence *A119626* shall be discussed in this paper. This special integer sequence *A119626* is obtained from computing the difference in the number of inverted pairs of the first and the last cycles which in turn is obtained from a special (123)-avoiding permutation pattern. They are: 3, 6, 12, 30, 84, 246, 732, 2790, 8364, 25086, etc generated by the formula $3 + 3^n$, with n taking values from 0, 1, 2, 3… , as in [3]. The computation in this paper shall be restricted to values of n = 1, 2, 3, 4, and 5 respectively and the results shall in no small measure contribute to the knowledge of error control.

II. DEFINITION OF TERMS USED

Throughout this paper, the following terms as used are defined as follows:

i. Binary coded data: According to [4], in most digital and communication systems, information is transmitted in form of binary coded data that is, messages are in the form of the symbols '0' and '1'. Communication is the process of transmitting information by [4]. This transmission can either be made between two distinct places, say a telephone call or between two points in time. As an example, the writing of this paper so that it could be read latter is a form of communication.

ii. Binary coded decimal: Binary coded decimals are those codes in which error detection and correction is done in binary information (bits). Hence, after the error is detected or located, correction means only flipping the bit found erroneous as in [4]. E.G: The decimal number in the complete four-bit reverse (that is, parity position) binary code is shown in the Table below: (DN= Decimal number)

DN	$2^0 = 1$	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

iii. Computational method: In the computational method, the following procedures are involved: (a) the bit positions are numbered in sequence from 1 to $n + k$ in a tabular form. Those positions numbered with powers of two are reserved for the parity bits. The remaining bits are the data bits. (b) Identify the data positions with '1', obtain their respective binary equivalences and set parity to '1' if odd numbers of 1's and to '0' if even numbers of 1's along their respective columns, the result from the table gives the parity code.

iv. Parity operator: This is the process of reversing parity bit set. E.G, the even parity (XOR) operator is the reversing of parity bit sets of the comparison between the parity code of the encoded (sent) data and that of the received (re-calculated) data when detecting an even data error location. While the odd parity (XNOR) operator is the reversing of both the parity bit sets of the received data positions having '1' and the comparison of the encoded (sent) parity code with that of the received (re-calculated) parity code when detecting an odd data error location as in [5].

v. DATA PRESENTATION AND ANALYSIS

The results obtained from the computation of the integer sequence *A119626* generated by the formula $3 + 3^n$, with *n* taking values from 1-5, along with their respective binary coded data (binary equivalences), parity codes and encoded data are shown in the table below:

Table 3.0 (Results obtained from the computation of the integer sequence)

n	$3 + 3^n$	Binary coded data	Parity code	Encoded data
1	6	110	**011**	**011**10
2	12	1100	**011**	**0111**100
3	30	11110	**1110**	**1111111**0
4	84	1010100	**0011**	**00**110**1**0**1**100
5	246	11110110	**0110**	**01111110**0110
…	…	…	**…**	…

Source: Researcher's calculation

From TABLE 3.0 above, the computational results of the integer sequence *A119626* using the formula $3 + 3^n$, with n = 1-5, were obtained as follows:

When $n = 1$; $3 + 3^1 = 6$, when $n = 2$; $3 + 3^2 = 12$, when $n = 3$; $3 + 3^3 = 30$, when $n = 4$; $3 + 3^4 = 84$, when $n = 5$; $3 + 3^5 = 246$.

Their respective parity codes and encoded data were obtained by the analysis of the Computational method as follows: (where: DP = Data position, ES = Encoded sequence, PBS = Parity bit set and DEP = Data error position)

[6]: 1 1 0 arrange the given bytes data in their respective positions in a table, leaving space for parity bits, that is positions 1, 2 and 4 respectively. This is shown in the table below:

Table 3.1: (Data and parity bit position)

DP	1	2	3	4	5	6
ES	**X**	**X**	1	**X**	1	0

Source: Researcher's calculation

We observed that data positions 3 and 5 have 1's respectively, using our previous binary conversion table, their respective binary equivalences are obtained and parity bit is set to a '**1**' for odd numbers of 1's and to a '0' for even numbers of 1's. This is shown in the table below:

Table 3.2: (Binary equivalence of data position with 1's)

DP	Binary Equivalence
3	0 1 1
5	1 0 1
PBS	**1 1 0**

Source: Researcher's calculation

The parity bit set obtained are reversed to form the parity code then placed back in their proper location in the original table 3.1 above. This is shown in the table below:

Table 3.3 (Data and parity bit position)

DP	1	2	3	4	5	6
ES	**0**	**1**	1	**1**	1	0

Source: Researcher's calculation

Thus the encoded data is: **0 1** 1 **1** 1 0.

[12]: 1 1 0 0 arrange the given bytes data in their respective positions in a table, leaving space for parity bits, that is positions 1, 2 and 4 respectively. This is shown in the table below:

Table 3.4 (Data and parity bit position)

DP	1	2	3	4	5	6	7
ES	**X**	**X**	1	**X**	1	0	0

Source: Researcher's calculation

We observed that data positions 3 and 5 have 1's respectively, using our previous binary conversion table, their respective binary equivalences are obtained and parity bit is set to a '**1**' for odd numbers of 1's and to a '0' for even numbers of 1's. This is shown in the table below:

Table 3.5 (Binary equivalence of data position with 1's)

DP	Binary Equivalence
3	0 1 1
5	1 0 1
PBS	**1 1 0**

Source: Researcher's calculation

The parity bit set obtained are reversed to form the parity code then placed back in their proper location in the original table 3.4 above. This is shown in the table below:

Table 3.6 (Data and parity bit position)

DP	1	2	3	4	5	6	7
ES	**0**	**1**	1	**1**	1	0	0

Source: Researcher's calculation

Thus the encoded data is: **0 1** 1 **1** 1 0 0.

[30]: 1 1 1 1 0 arrange the given bytes data in their respective positions in a table, leaving space for parity bits, that is positions 1, 2, 4 and 8 respectively. This is shown in the table below:

Table 3.7 (Data and parity bit position)

DP	1	2	3	4	5	6	7	8	9
ES	**X**	**x**	1	**X**	1	1	1	X	0

We observed that data positions 3, 5, 6 and 7 have 1's respectively, using our previous binary conversion table, their respective binary equivalences are obtained and parity bit is set to a '**1**' for odd numbers of 1's and to a '0' for even numbers of 1's. This is shown in the table below:

Table 3.8 (Binary equivalence of data position with 1's)

DP	Binary Equivalence
3	0 0 1 1
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
PBS	**0 1 1 1**

Source: Researcher's calculation

The parity bit set obtained are reversed to form the parity code then placed back in their proper location in the original table 3.7 above. This is shown in the table below:

Table 3.9 (Data and parity bit position)

DP	1	2	3	4	5	6	7	8	9
ES	**1**	**1**	1	**1**	1	1	1	0	0

Source: Researcher's calculation

Thus the encoded data is: **1 1** 1 **1** 1 1 **0** 0.

[84]:1 0 1 0 1 0 0 arrange the given bytes data in their respective positions in a table, leaving space for parity bits, that is positions 1, 2, 4 and 8 respectively. This is shown in the table below:

Table 3.10 (Data and parity bit position)

DP	1	2	3	4	5	6	7	8	9	10	11
ES	**X**	**x**	1	**X**	0	1	0	**X**	1	0	0

We observed that data positions 3, 6 and 9 have 1's respectively, using our previous binary conversion table, their respective binary equivalences are obtained and parity bit is set to a '**1**' for odd numbers of 1's and to a '0' for even numbers of 1's. This is shown in the table below:

Table 3.11 (Binary equivalence of data position with 1's)

DP	Binary Equivalence
3	0 0 1 1
6	0 1 1 0
9	1 0 0 1
PBS	**1 1 0 0**

Source: Researcher's calculation

The parity bit set obtained are reversed to form the parity code then placed back in their proper location in the original table 3.10 above. This is shown in the table below:

Table 3.12 (Data and parity bit position)

DP	1	2	3	4	5	6	7	8	9	10	11
ES	**0**	**0**	1	**1**	0	1	0	**1**	1	0	0

Thus the encoded data is: **0 0** 1 **1** 0 1 0 **1** 1 0 0.
[246]: 1 1 1 1 0 1 1 0 arrange the given bytes data in their respective positions in a table, leaving space for parity bits, that is positions 1, 2, 4 and 8 respectively. This is shown in the table below:

Table 3.13 (Data and parity bit position)

DP	1	2	3	4	5	6	7	8	9	10	11	12
ES	**X**	**x**	1	**X**	1	1	1	**X**	0	1	1	0

Source: Researcher's calculation

We observed that data positions 3, 5, 6, 7, 10 and 11 have 1's respectively, using our previous binary conversion table, their respective binary equivalences are obtained and parity bit is set to a '**1**' for odd numbers of 1's and to a '0' for even numbers of 1's. This is shown in the table below:

Table 3.14 (Binary equivalence of data position with 1's)

DP	Binary Equivalence
3	0 0 1 1
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
10	1 0 1 0
11	1 0 1 1
PBS	**0 1 1 0**

Source: Researcher's calculation

The parity bit set obtained are reversed to form the parity code then placed back in their proper location in the original table 3.13 above. This is shown in the table below:

Table 3.15 (Data and parity bit position)

DP	1	2	3	4	5	6	7	8	9	10	11	12
ES	**0**	**1**	1	**1**	1	1	1	**0**	0	1	1	0

Source: Researcher's calculation

Thus the encoded data is: **0 1** 1 **1** 1 1 1 **0** 0 1 1 0.

VI. Analysis of the Computational method of Hamming code techniques for detection and correction of computational errors in a binary coded data of an integer sequence A119626

The results obtained from the computation of integer sequence *A119626* generated by the formula $3 + 3^n$, with n taking values from 1-5, their respective encoded (computed) data, erroneous results (data) obtained instead with their binary coded data (binary equivalences) and the positions of the erroneous data (DEP) are shown in the table below:

Table 4.1 (Results of error detection and correction obtained from the computation of the integer sequence *A119626*)

n	$3 + 3^n$	Binary Equivalence	Parity Code	Encoded Data	$3 + 3^n$ error	Binary Equivalence	Parity Code	Encoded Data	DEP
1	6	110	011	011110	7	111	000	001011	6
2	12	1100	011	0111100	13	1101	100	1010101	7
3	30	11110	1110	111111100	22	10110	0100	011001100	5
4	84	1010100	0011	00110101100	85	1010101	1110	11110100101	11
5	246	11110110	0110	011111100110	118	01110110	1010	100111100110	3
…	…	…	…	…	…	…	…	…	…

Source: Researcher's calculation

From the TABLE above, the parity code, encoded data and the data error positions of the respective erroneous results (data) obtained in the process of computation are shown using the analysis of the Computational method as follows:

[7]: Suppose 0 1 1 1 1 0 was computed but 0 1 1 1 1 1 was obtained instead. Since error data is in position 6, we use the even parity operator (XOR) that is, reverse the binary equivalence at the comparison of encoded data and re-calculated data parity codes respectively (parity bit set). At the receiver's end, the following encoded sequence was seen as shown in the table below:

Table 4.2 (Data and parity bit position)

DP	1	2	3	4	5	6
ES	**0**	**1**	1	**1**	1	1

Source: Researcher's calculation

We observed that data positions 3, 5 and 6 have 1's respectively, using our previous binary conversion table, their respective binary equivalences are obtained and parity bit is set to a '**1**' for odd numbers of 1's and to a '0' for even numbers of 1's. This is shown in the table below:

Table 4.3 (Binary equivalence of data position with 1's)

DP	Binary Equivalence
3	0 1 1
5	1 0 1
6	1 1 0
PBS	**0 0 0**

Source: Researcher's calculation

The re-calculated parity information is then compared with the encoded (computed) parity. If they are both the same, the results will be all 0's. But if a single bit was corrupted, the resulting parity set will be the position of the corrupted or erroneous data. This is shown in the table below:

Table 4.4 (Re-calculated parity code compared with the encoded parity)

Encoded (computed) Parity code	0 1 1
Re-calculated parity code	0 0 0
Parity Bit Set	**0 1 1**

Source: Researcher's calculation

The parity bit set **1 1 0** forms the parity code and is equivalent to 6 in our binary conversion table. Therefore, data position 6 of the obtained result was corrupted. To correct the result obtained, data position 6 is flipped from '1' to '0'. Thus, 0 1 1 1 1 0 would eventually be obtained as final result.[13]: Suppose 0 1 1 1 1 0 0 was computed but 0 1 1 1 1 0 1 was obtained instead. Since error data is in position 7, we use the odd parity operator (XNOR),that is reverse binary equivalence both at the parity bit set of data positions 1's and the

comparison of encoded data and re-calculated data parity codes respectively. At the receiver end, the following encoded sequence was seen as shown in the table below:

Table 4.5 (Data and parity bit position)

DP	1	2	3	4	5	6	7
ES	0	1	1	1	1	0	1

Source: Researcher's calculation

We observed that data positions 3, 5 and 7 have 1's, using our previous binary conversion table, their binary equivalence are obtained and parity bit is set to a '**1**' for odd numbers of 1's and to a '0' for even numbers of 1's. This is shown in the table below:

Table 4.6 (Binary equivalence of data position with 1's)

DP	Binary Equivalence
3	0 1 1
5	1 0 1
7	1 1 1
PBS	**0 0 1**

Source: Researcher's calculation

The re-calculated parity information is then compared with the encoded (computed) parity. If they are both the same, the results will be all 0's. But if a single bit was corrupted, the resulting parity set will be the position of the corrupted or erroneous data. This is shown in the table below:

Table 4.7 (Re-calculated parity code compared with the encoded parity)

Encoded (computed) Parity	**0 1 1**
Recalculated Parity	**1 0 0**
Parity Bit Set	**1 1 1**

Source: Researcher's calculation

The reversed parity bit **1 1 1** form the parity code and is equivalent to 7 in our binary conversion table. Therefore, data position 7 of the obtained result was corrupted. To correct the result obtained, data position 7 is flipped from '1' to '0'. Thus, 0 1 1 1 1 0 0 would eventually be obtained as final result.[30]: Suppose 1 1 1 1 1 1 0 0 is computed but 1 1 1 1 0 1 1 0 0 was obtained instead. Since error data is in position 5, we use the odd parity operator (XNOR),that is reverse binary equivalence both at the parity bit set of data positions 1's and the comparison of encoded data and re-calculated data parity codes respectively. At the receiver end, the following encoded sequence was seen as shown in the table below:

Table 4.8 (Data and parity bit position)

DP	1	2	3	4	5	6	7	8	9
ES	**1**	**1**	1	**1**	0	1	1	**0**	0

Source: Researcher's calculation

We observed that data positions 3, 6, and 7 have 1's, using our previous binary conversion table, their binary equivalence are obtained and parity bit is set to a '**1**' for odd numbers of 1's and to a '0' for even numbers of 1's along their respective columns. This is shown in the table below:

Table 4.9 (Binary equivalence of data position with 1's)

DP	Binary Equivalence
3	0 1 1
6	1 1 0
7	1 1 1
PBS	**0 1 0**

Source: Researcher's calculation

The re-calculated parity information is then compared with the encoded (computed) parity. If they are both the same, the results will be all 0's. But if a single bit was corrupted, the resulting parity set will be the position of the corrupted or erroneous data. This is shown in the table below:

Table 4.10 (Re-calculated parity code compared with the encoded parity)

Encoded (computed) Parity	1 1 1 0
Recalculated Parity	0 1 0 0
Parity Bit Set	**1 0 1 0**

Source: Researcher's calculation

The reversed parity bit set **0 1 0 1** forms the parity code and is equivalent to 5 in our binary conversion table. Therefore, data position 5 of the obtained result was corrupted. To correct the result obtained, data position 5 is flipped from '0' to '1'. Thus, 1 1 1 1 1 1 1 0 0 would eventually be obtained as final result.[85]:

Suppose 0 0 1 1 0 1 0 1 1 0 0 is computed, but 0 0 1 1 0 1 0 1 1 0 1 was obtained instead. Since error data is in position 11, we use the odd parity operator (XNOR),that is reverse binary equivalence both at the parity bit set of data positions 1's and the comparison of encoded data and re-calculated data parity codes respectively. At the receiver end, the following encoded sequence was seen as shown in the table below:

Table 4.11 (Data and parity bit position)

DP	1	2	3	4	5	6	7	8	9	10	11
ES	**0**	**0**	1	**1**	0	1	0	**1**	1	0	1

Source: Researcher's calculation

We observed that data positions 3, 6, 9 and 11 have 1's, using our previous binary conversion table, their respective binary equivalence are obtained and parity bit is set to a '**1**' for odd numbers of 1's and to a '0' for even numbers of 1's. This is shown in the table below:

Table 4.12(Binary equivalence of data position with 1's)

DP	Binary Equivalence
3	0 0 1 1
6	0 1 1 0
9	1 0 0 1
11	1 0 1 1
PBS	**0 1 1 1**

Source: Researcher's calculation

The recalculated parity information is then compared with the encoded (computed) parity. If they are both the same, the results will be all 0's. But if a single bit was corrupted, the resulting parity set will be the position of the corrupted or erroneous data. This is shown in the table below:

Table 4.13 (Re-calculated parity code compared with the encoded parity)

Encoded (computed) Parity	**0 0 1 1**
Recalculated Parity	**1 1 1 0**
Parity Bit Set	**1 1 0 1**

Source: Researcher's calculation

The reversed parity bit set **1 0 1 1** forms the parity code and is equivalent to 11 in our binary conversion table. Therefore, data position 11 of the obtained result was corrupted. To correct the result obtained, data position 11 is flipped from '1' to '0'. Thus, 0 0 1 1 0 1 0 1 1 0 0 would eventually be obtained as final result.

[118]: Suppose 0 1 1 1 1 1 1 0 0 1 1 0 was computed but 0 1 0 1 1 1 1 0 0 1 1 0 was obtained instead. Since error data is in position 3, we use the odd parity operator (XNOR),that is reverse binary equivalence both at the parity bit set of data positions 1's and the comparison of encoded data and re-calculated data parity codes respectively. At the receiver end, the following encoded sequence was seen as shown in the table below:

Table 4.14 (Data and parity bit position)

DP	1	2	3	4	5	6	7	8	9	10	11	12
ES	**0**	**1**	0	**1**	1	1	1	**0**	0	1	1	0

Source: Researcher's calculation

We observed that data positions 5, 6, 7, 10, and 11 have 1's, using our previous binary conversion table, their respective binary equivalence are obtained and parity bit is set to a '**1**' for odd numbers of 1's and to a '0' for even numbers of 1's along their respective columns. This is shown in the table below:

Table 4.15 (Binary equivalence of data position with 1's)

DP	Binary Equivalence
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
10	1 0 1 0
11	1 0 1 1
PBS	**0 1 0 1**

Source: Researcher's calculation

The re-calculated parity information is then compared with the encoded (computed) parity. If they are both the same, the results will be all 0's. But if a single bit was corrupted, the resulting parity set will be the position of the corrupted or erroneous data. This is shown in the table below:

Table 4.16 (Re-calculated parity code compared with the encoded parity)

Encoded (computed) Parity	**0 1 1 0**
Recalculated Parity	**1 0 1 0**
Parity Bit Set	**1 1 0 0**

Source: Researcher's calculation

The reversed parity bit set **0 0 1 1** forms the parity code and is equivalent to 3 in our binary conversion table. Therefore, data position 3 of the obtained result was corrupted. To correct the result obtained, data position 3 is flipped from '0' to '1'. Thus, 0 1 1 1 1 1 1 0 0 1 1 0 would eventually be obtained as final result.From the analysis above, the total number of steps (iterations) and their respective total execution time (*T*) and rate (*R*), for any given time *t*, involved in the general analysis for encoding, detection and correction of computational error(s) using the Computational method of Hamming code techniques is summarized in the table below. Where the time of execution (*T*) and rate (*R*) are given as follows: $T = i \times t$ (that is, number of iteration *i* multiplied by the time *t* taken) and $R = i / t$ (that is the number of iteration *i* divided by the time *t* taken).

Table 4.32 (Results of the total number of steps, execution time and rate)

n	$3+3^n$	CME	CMEDC	T	CMET	CMER
1	6	2	4	6	6 x *t*	6 \ *t*
2	12	2	4	6	6 x *t*	6 \ *t*
3	30	2	4	6	6 x *t*	6 \ *t*
4	84	2	4	6	6 x *t*	6 \ *t*
5	246	2	4	6	6 x *t*	6 \ *t*
	Total	10	20	30	30 x *t*	30 \ *t*

Source: Researcher's calculation

(Where: CME = computational method of encoding, CMEDC = computational method of error detection and correction, T = total, CMET = computational method of execution time, CMER = computational method of execution rate):From the analysis of the Computational method of Hamming code techniques for encoding, detection and correction of computational error(s) in a binary coded data of an integer sequence *A119626* discussed in this paper, it is observed that for each value of *n* computed all have a total of six (6) number of steps (iterations) respectively. Thus, the total number of steps (iterations) for values of *n* = 1-5, is thirty (30). Therefore, the execution time (*T*) and rate (*R*) for any given time *t*, would be obtained by multiplying and dividing *T* and *R* by *t* respectively.

III. CONCLUSION

According to [6] the fewer the number of steps (iterations) involved in the analysis of any method of computation, the faster the execution time (*T*) and rate (*R*) and the more efficient is that method. From the analysis in this paper, the computational method has a fewer total number of steps (iterations) which is thirty (30) involved in the process of computation and lesser execution time (*T*) and rate (*R)* for any given time *t* when compared to that of the Algorithmic method which was found to be forty six (46) in my previous analysis [7]. Since the analysis of the Computational method has a fewer total number of steps (iterations) which is thirty (30) involved in the process of computation and lesser execution time (*T*) and rate (*R)*for any given time *t* when compared with that of the Algorithmic method which is forty six (46).Therefore, the Computational method of Hamming code techniques has a faster execution time (*T*) and rate (*R*) for any given time *t*, and thus, is a more efficient analysis for encoding, detection and correction of computational error(s) in a binary coded data of an integer sequence *A119626* when compared with the Algorithmic method. Although the computation in this paper was restricted to values of n = 1-5 of the integer sequence *A119626*, however, the results obtained can be applied generally on computation involving binary coded data.

REFERENCES

[1.] I. Koren, *Computer arithmetic algorithms* (Natrick MA): A. K. Peters, 2002.

[2.] R. W. Hamming, Bell system Technology, *Journal of Error Detecting and Correcting Codes vol. 29, April, 1950, pp. 147-160.*

[3.] A. A. Ibrahim, Mathematics Association of Nigeria, *the journal On the Combinatorics of A Five-element sample Abacus of vol. 32, 2005, No. 2B: 410-415.*

[4.] B. Sklar, Digital communication: *fundamentals and applications* (Second Edition: Prentice-Hall, 2001).

[5.] Moon & K. Todd, *Error correction coding* (http://www.neng.usu.edu/ece/faculty/tmoon/eccbook/book.html). (New Jersey: John Wiley, 2005 and sons ISBN 978-0-471-64800-0).

[6.] Nirandi, Computational complexity of a longest path algorithm with a recursive method, 2010.

[7.] A. Godfrey & A. A. Ibrahim, IOSR Journals of Mathematics (IOSR- JM) e – ISSN: 2278, p – ISSN: 2319 – 7676. Volume 9, Issue 2 (Nov. – Dec. 2013) pp 33 -37. www.iosrjournals.org.