Algorithm to Compute Cubes of 1st "N" Natural Numbers using Single Multiplication per Iteration

Rajat Tandon
Samsung R&D Institute*
Bagmane Constellation Park,
Doddanekundi Circle, Bangalore, India

Rajika Tandon
AT&T*
575 Morosgo Dr NE,
Atlanta, Georgia, USA

ABSTRACT
Different processors work with disparate speeds. For any given processor, elementary operations differ in terms of their speeds and computational complexities. The paper presents an algorithm to compute cubes of 1st "N" Natural Numbers using one multiplication by constant, two additions on variables and one addition by constant, per iteration. Theoretically, computational complexity of multiplication is $O(n^2)$ while that of addition is $\Theta(n)$, where n is the number of bits used to represent that number. So, keeping the number of iterations same in both, in the traditional approach, the overall computational complexity per iteration is expressed in the order of $O(n^2)$ while in the current approach the overall computational complexity per iteration is of the order of $O(n)$. For small values of "N", the difference in complexities may not be huge. But, given any large value of "N", difference will be noticeable.

General Terms
Algorithms, Sum of Cubes

Keywords
Computational complexity, Cube

1. INTRODUCTION
Computational complexity of an algorithm is a measure of how many steps the algorithm will require in the worst case for an instance or input of a given size. It is commonly estimated by counting the number of elementary operations performed by the algorithm, where an elementary operation takes a fixed amount of time to perform. Addition requires lesser time than multiplication [3] as in case of serial processing.

If the value of "N" is small, then there will not be much time difference. However, for a large value of "N", the difference cannot be ignored. Table 1 shows some of the elementary operations and their time complexities.

2. METHODOLOGY
As the number of iterations involved in both the cases is same, the comparison is based on the operation computational complexity per iteration. The complexity mentioned in Table 1 is expressed in terms of the number of digits in the numbers. In the traditional approach, the cubes of the 1st "N" Natural Numbers is computed by the following equation:

$$cube \leftarrow i * i * i$$

where, i varies from 1 to N, and the computational complexity is computed by considering he complexity of 2 multiplications, which is $2*O(n^2)$. So it can be approximated to the order of $O(n^2)$ [1, 2, 4, 5].

2.1 Traditional Algorithm
The following algorithm presents the traditional approach to compute the cubes of first N natural numbers. The cube in this way is computed by multiplying the number by itself, and multiplying the result obtained, by the number again. This method is applied for all the N numbers.

Algorithm: Cubes of first N natural numbers – traditional way
Input: A positive integer "N"
Output: Cubes of 1st "N" natural numbers

```
for i ← 1 to N do
  cube ← i * i * i // Two multiplication operations
  print cube
end for
```

2.2 Proposed Algorithm
The following algorithm presents the approach proposed in this paper. In order to compute the cube of first N natural numbers, we have identified a series {(6*a) + 1}, where a is computed by adding natural numbers to the previous value of 'a', and initial value of 'a' being 0. The cube is computed by adding cube of previous number, (6 * a), and 1.

Algorithm: Cubes of first N natural numbers – proposed way
Input: A positive integer "N"
Output: Cubes of 1st "N" natural numbers

```
a ← 0
cube ← 0
for i ← 1 to N do
  cube ← cube + (6*a) +1 // One multiplication by constant
  a ← a + i
  print cube
end for
```

3. EXPLANATION
The logic behind the proposed algorithm is that all perfect cubes of natural numbers differ by the series {(6*a) + 1}, where, a is given by the series {0, 1, 3, 6, 10, 15, 21…} which can be computed by adding natural numbers from the series {1, 2, 3, 4, 5, 6, 7…} to the previous value of 'a'.

For example, if we start from 1 and add 7 to it, we will obtain the cube of 2. Similarly, if we add 19 to the previous value of the variable "cube" then we get 27, which is the cube of 3. Similarly, if we add 37 to the previous value of the variable "cube" then we get 64, which is the cube of 4. Likewise, we'll obtain cube of N as: cube of (N-1) + (6 * a) + 1.

4. COMPUTATIONAL COMPLEXITY
Different elementary operations have different computational complexities based on the number of digits present in the two numbers. Table 1 depicts the rudimentary arithmetic operations along with their algorithmic computational complexities.

Schoolbook addition with carry algorithm has its complexity of the order $\Theta(n)$. While the complexity of schoolbook long multiplication is of the order $O(n^2)$. The asymptotic analysis or growth of function [5] of "n", i.e. $\Theta(n)$, is linear, while that of a function of "n^2", i.e. $O(n^2)$, is quadratic. The graph presented in figure 1 provides a comparison of time complexities of linear and quadratic functions. Thus, we can clearly see that a linear computational complexity is preferred over quadratic as less time is spent on computation involving "n" than computations involving "square of n". This is very significant especially when the values of n are very large, in which case, linear calculations save a lot of computational time over quadratic calculations.

The traditional algorithm, discussed in section 2.1, involves 2 multiplications. Multiplication operation is always expensive compared to addition operation if computational complexity is a concern as that in case of serial processing. So it is better to use addition operation in such cases.

The proposed algorithm, discussed in section 2.2, involves 1 multiplication by a constant which is the number "6" and 3 additions per iteration. The complexity of the multiplication "(6*a)" is of the order of $O(3n)$ because 3 bits are used to represent the number "6". If the number of iterations involved is less, then the complexities don't reveal a significant difference.

The computational complexity which is of the order of $O(3n)$ can approximated to the order of $O(n)$. The complexities of additions also end up being in the order of $\Theta(n)$ after approximations. So the overall computational complexity per iteration is of the order of $O(n)$ in the current approach.

Table 1. Elementary operations with their time complexities

Operation	**Input**	**Output**	**Algorithm**	**Complexity**
Addition	Two n-digit numbers	One n+1 – digit number	Schoolbook addition with carry	$\Theta(n)$
Subtraction	Two n-digit numbers	One n+1 – digit number	Schoolbook subtraction with borrow	$\Theta(n)$
Multiplication	Two n-digit numbers	One 2n-digit number	Schoolbook long multiplication	$O(n^2)$
Division	Two n-digit numbers	One n-digit number	Schoolbook long division	$O(n^2)$

Table 2. Values for N=10

i	**cube ← cube + (6*a) + 1**	**a ← a + i**
1	1 ← 0 + (6*0) + 1	1 ← 0 + 1
2	8 ← 1 + (6*1) + 1	3 ← 1 + 2
3	27 ← 8 + (6*3) + 1	6 ← 3 + 3
4	64 ← 27 + (6*6) + 1	10 ← 6 + 4
5	125 ← 64 + (6*10) + 1	15 ← 10 + 5
6	216 ← 125 + (6*15) + 1	21 ← 15 + 6
7	343 ← 216 + (6*21) + 1	28 ← 21 + 7
8	512 ← 343 + (6*28) + 1	36 ← 28 + 8
9	729 ← 512 + (6*36) + 1	45 ← 36 + 9
10	1000 ← 729 + (6*45) + 1	55 ← 45 + 10

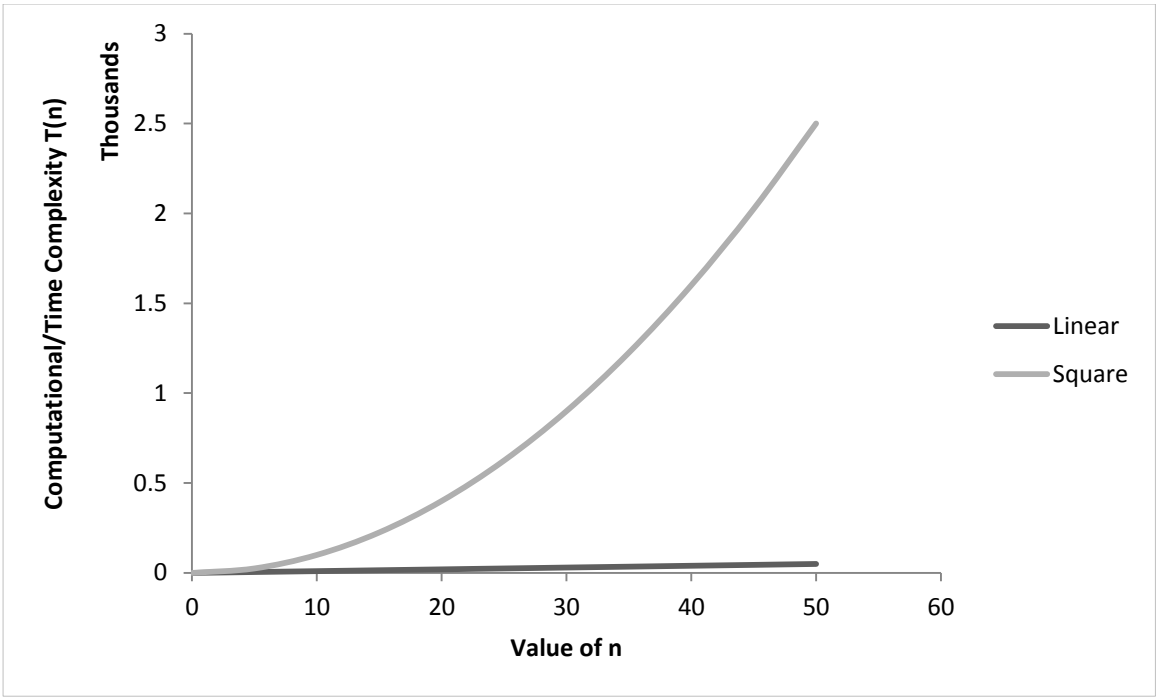

Fig 1: Computational complexities of linear (n) and quadratic/square (n^2) functions

5. OUTPUT TRACING

Initially the values of the variables are shown in Table 3.

Table 3

a	cube
0	0

Before the iteration begins, both the variables: "cube" and "a", are set to 0.

Let us consider the first iteration. The value of the variable "cube" is added with the value of (6*0 + 1), to get the cube of the first number. The value of the variable "cube" becomes 1. Then the value of the variable "a" is incremented by the value of i in order to compute the cube of the next number in the next iteration. The value of the variable "a" becomes 1. The value of variable "i" is incremented by 1. The values of the variables after the first iteration are shown in Table 4.

Table 4

a	cube
1	1

Let us consider the second iteration. After the first iteration, the value of the variable "cube" is 1 and "a" is also 1. The value of the variable "cube" is added to the value of (6*1 + 1), to get the cube of the second number. The new value of the variable "cube" becomes 8. Then the value of the variable "a" is incremented by the iteration number in order to compute the cube of the next number in the next iteration. The new value of the variable "a" becomes 3. The values of the variables after the second iteration are shown in Table 5.

Table 5

a	cube
3	8

Let us consider the third iteration. The value of the variable "cube" is added to the value of (6*3 + 1), to get the cube of the third number. Then the value of the variable "a" is incremented by the iteration number in order to compute the cube of the next number in the next iteration. The values of the variables after the third iteration are shown in Table 6.

Table 6

a	cube
6	27

Similarly, the loop runs till the value of the variable "i" reaches "n".

6. MATHEMATICAL PROOF

Given, for i = 0, $cube_{(i)} = 0$, and $\alpha_{(i)} = 0$

To prove,

$$\mathbf{cube_{(i)} = cube_{(i-1)} + 6*\alpha_{(i-1)} + 1} \text{ ----- (i)}$$

Equation (i) is true for $i \geq 1$

where, $\boldsymbol{\alpha_i = \alpha_{(i-1)} + i}$ ------- (ii)

Proof by induction:

STEP 1**:**

For i = 1,

LHS,

$cube_{(1)} = 1*1*1 = 1$

RHS,

$cube_{(1-1)} + 6*\alpha_{(1-1)} + 1$
$= cube_{(0)} + 6*\alpha_{(0)} + 1$
$= 0 + 0 + 1 = 1$

STEP 2:

Suppose Equation (i) is true for some $i = k \geq 1$, that is

$cube_{(k)} = cube_{(k-1)} + 6*\alpha_{(k-1)} + 1$ ---------- (iii)

STEP 3:

Prove that Equation (i) is true for $i = k + 1$, that is

LHS,

$cube_{(k+1)} = (k+1) * (k+1) * (k+1)$
$= (k^2 + k + k + 1) * (k+1)$
$= (k^2 + 2k + 1) * (k + 1)$
$= k^3 + 2k^2 + k + k^2 + 2k + 1$
$= k^3 + 3k^2 + 3k + 1$

RHS,

$cube_{(k)} + 6*\alpha_{(k)} + 1$
$= (k*k*k) + 6 [\alpha_{(k-1)} + k] + 1$
$= (k*k*k) + 6 [\alpha_{(k-2)} + (k-1) + k] + 1$
$= (k*k*k) + 6 [\alpha_{(k-3)} + (k-2) + (k-1) + k] + 1$
…
…
$= (k*k*k) + 6 [\alpha_{(0)} + 1 + 2 + 3 + \ldots (k-2) + (k-1) + k] + 1$
$= (k*k*k) + 6 [0 + 1 + 2 + 3 + \ldots (k-2) + (k-1) + k] + 1$

Using result on summation of natural numbers [6]:
where, $1 + 2 + 3 + \ldots + n = n(n+1)/2$,
Therefore, the above RHS can be reduced to the following:
$= (k*k*k) + 6[k(k+1)/2] + 1$
$= (k*k*k) + 3(k*k) + 3(k) + 1$
$= k^3 + 3k^2 + 3k + 1$
Therefore, LHS = RHS.
Hence proved that the equation (i) holds true.

7. ADDITIONAL RESULT

The algorithm discussed in the paper can be extended to compute the cubes of a given range.

If the given range is Range(a,b),where a < b, and the cube of the number (a-1) is provided or computed using the traditional approach, then the series will follow the trend listed below:

$cube_{(a)} = cube_{(a-1)} + 6*\alpha_{(a-1)} + 1$

where, $\alpha_{(a-1)} = [\alpha_{(a-2)} + (a-1)]$
$= [\alpha_{(a-3)} + (a-2) + (a-1)]$
…
$= [\alpha_{(0)} + 1 + 2 + 3 + \ldots + (a-2) + (a-1)]$
$= [0 + 1 + 2 + 3 + \ldots + (a-2) + (a-1)]$

Using result on summation of natural numbers [6]:

where, $1 + 2 + 3 + \ldots + n = n(n+1)/2$,

we get:

$\alpha_{(a-1)} = a(a-1)/2$

Similarly,

$cube_{(b)} = cube_{(b-1)} + 6*\alpha_{(b-1)} + 1;$

where, $\alpha_{(b)} = \alpha_{(b-1)} + b$ (from (ii))

8. CONCLUSIONS

In this paper, we presented a more optimized solution of computing the cubes of first "N" natural numbers in terms of computational complexity, than the traditional algorithm.

Usually, addition and subtraction are preferred to multiplication and division, because the latter operations cause higher overhead in terms of computational complexity, as in case of serial processing.

We have mathematically and graphically presented the computational complexities of traditional addition and multiplication methods, and reduced the problem of computing the cubes of first "N" natural numbers from 2 multiplications per iteration, to 1 multiplication by a constant, and 3 addition operations.

We have also shown that the same approach is applicable in computing a range of cubes (i.e., finding cubes of all natural numbers between say numbers "a" and "b").

9. FUTURE SCOPE

This method can be applied to compute the sum of cubes of first N natural numbers, or to compute the sum of cubes of a range of natural numbers.

Also, the approach discussed in this paper and reference [3] is for cubes and squares respectively. Similar ideas may be applicable for power 4, 5 and so on, and a generalized result can be derived. This will also allow easy computation of a range of numbers with any power. Additionally, the sum of any power of first N natural numbers or a range of natural numbers can be computed.

10. DISCLOURE

*The work on this study was performed by the authors independent of their primary affiliations. The contents of this paper are solely the responsibility of the authors and do not represent the official view of Samsung R&D Institute and AT&T.

11. REFERENCES

[1] Levitin, A. 2002. Introduction to the Design and Analysis of Algorithms, 2nd ed., Addison Wesley.

[2] Harel, D. and Feldman Y. 2004. Algorithmics: The Spirit of Computing, 3rd ed., Addison-Wesley Publishers Limited.

[3] Tandon, R. 2012. "Algorithm to Compute Squares of 1st "N" Natural Numbers Without Using Multiplication", arXiv:1212.5645v1 [cs.DS].

[4] Knuth, D. E. 1997. The Art of Computer Programming, vol. 1, 3rd ed., Addison-Wesley.

[5] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. 2009. Introduction to Algorithms, 3rd ed., The MIT Press and McGraw-Hill.

[6] Malhotra, O. P., Gupta S. K., and Gangal A., 2007. I.S.C. Mathematics Book I for Class XI (page 6-39), S. Chand & Company Ltd.