NATIONAL INSTITUTE OF TECHNOLOGY, SILCHAR

Department of Computer Science and Engineering

Project Report on,

ALGORITHMS

Subject : Algorithms (CS-206)

Submitted by,
Name : Arvinder Singh
Scholar ID : 18-1-5-126
B.Tech 4th SEM (CSE)

August 13, 2020

ABSTRACT

"In this project report, I have tried to give a concise overview on my approach to solve the given problem statement in the most optimal and feasible way.

Starting with the project description I have tried to devise my approach in tackling the problem. Then gradually I have build up the algorithm. Later I have proved the correctness of my designed algorithm using mathematical induction. Later I have justified the paradigm used in my solution.

Rigorous time complexity analysis is done at the end. Example input and output for the algorithm is also shown alongwith step-wise running of the algorithm and final output is obtained. Lastly, some drawbacks of the algorithm is also discussed.

This project is developed using one of the open source software LaTeX
."

Contents

1 Project Description

In this project we develop an algorithm that converts the given input which is in *integer (base* 10*)* format to *binary (base* 2*)* representation. We use one of the algorithm paradigms *Divide and Conquer* to achieve this. We exploit given procedure FAST-MULTIPLY(x, y) to achieve this for performance reasons *(discussed later).*

We devise two algorithms BIN-POW-TEN(n) which takes a single parameter n and raises 10 to the power n and returns the value in binary. Also, another function DEC-TO-BIN(x) is devised to convert *decimal* or *integer* value to *binary.*

In the following section we will give the concise algorithm for all the procedures.

2 Algorithm

In this section I have given algorithms for the functions described in above, *i.e.,* FAST-MULTIPLY(x, y), BIN-POW-TEN(n) and DEC-TO-BIN(x). For simplifying our analysis we consider cost of all operations to be 1. The addition and multiplications of 2 takes linear time since it is merely *left shifts.*

- FAST-MULTIPLY(x, y) // $x, y \rightarrow$ *binary numbers*
  Input: $x = 1011_2, y = 1110_2$
  Output: 1001010_2

		Cost
1:	$n = max$(no. of bits of x, no. of bit of y)	$\mathcal{O}(1)$
2:	**if** $n = 1$ **then**	
3:	*return* xy	$\mathcal{O}(1)$
4:	$x_L, x_R = leftmost\lceil\frac{n}{2}\rceil, rightmost\lfloor\frac{n}{2}\rfloor$ *of* x	$\mathcal{O}(1)$
5:	$y_L, y_R = leftmost\lceil\frac{n}{2}\rceil, rightmost\lfloor\frac{n}{2}\rfloor$ *of* y	$\mathcal{O}(1)$
6:	$M_1 =$ FAST-MULTIPLY(x_L, x_R)	$T(\frac{n}{2})$
7:	$M_2 =$ FAST-MULTIPLY(y_L, y_R)	$T(\frac{n}{2})$
8:	$M_3 =$ FAST-MULTIPLY$(x_L + x_R, y_L + y_R)$	$T(\frac{n}{2})$
9:	*return* $2^n M_1 + 2^{\frac{n}{2}}(M_3 - M_1 - M_2) + M_2$	$\mathcal{O}(n)$

- BIN-POW-TEN(n) // $n \rightarrow$ *power to which 10 is to be raised*
  Input: $n = 2$
  Output: 1100100_2

		Cost
1:	**if** $n = 1$ **then**	
2:	*return* 1010_2	$\mathcal{O}(1)$
3:	**else**	
4:	$a =$BIN-POW-TEN$(\frac{n}{2})$	$G(\frac{n}{2})$
5:	*return* FAST-MULTIPLY(a, a)	$\mathcal{O}(n^{log_2 3})$

- DEC-TO-BIN(x) // $x \rightarrow$ *number which need to be converted to binary*

  Input: $x = 13$

  Output: 1101_2 *Cost*

  1: initialize bin[0...9] with all the binary values
  n $\leftarrow$ is the number of digits in x $\mathcal{O}(1)$
  2: **if** $n = 1$ **then**
  3: *return* $bin[x]$ $\mathcal{O}(1)$
  4: **else**
  5: $x_L, x_R = leftmost\frac{n}{2}, rightmost\frac{n}{2}$ *of* x $\mathcal{O}(1)$
  6: *return* FAST-MULTIPLY(BIN-POW-TEN($\frac{n}{2}$), DEC-TO-BIN(x_L))
  +DEC-TO-BIN(x_R) $2H(\frac{n}{2}) + G(\frac{n}{2}) + T(\frac{n}{2})$

3 Proof of Correctness

First of all we will prove the correctness of the FAST-MULTIPLY(x, y) algorithm, using structural induction of this recursive algorithm itself.

In this algorithm we know that the inputs is power of 2 as given in the problem statement. Also, in induction we first prove the *Basis* step and then move on to proving the *inductive* step. We also have to prove that the the algorithm terminates and that if it terminates then, the preconditions imply the postconditions.

We know that the algorithm terminates because the number of bits in each successive recursive call is strictly decreasing sequence belonging to the natural numbers,*i.e.,* $\in \mathbf{N}$.

Proof:

Let the statement be $T(n)$: Number of bits in the input.

Basis: If the length of the input numbers is 1. The algorithm simply multiply the two numbers and return it. Hence, the basis step is proved.

Inductive: Let us assume that the algorithm is correct for $T(k)$ we have to prove that the algorithm is correct for $T(k')$, where $k' > k$, in fact for our input k' is nothing but $k' = 2k$. We do this by showing that the combined step is indeed correct.

We know that multiplication of any two binary numbers is
$xy = (2^{k/2}x_L + x_R)(2^{k/2}y_L + y_R)$, where k is the length of the maximum of the length of the two numbers x and y. Also, the above can be simplified to shown to be,
$2^k x_L y_L + 2^{\frac{k}{2}}((x_L + x_R)(y_L + y_R) - x_L x_R - y_L y_R) + x_R y_R$.

The above combined step is indeed correct which is actually what is in the algorithm. This proves our inductive step. Therefore, this algorithm holds for k'.

Therefore, by the principle of mathematical induction we proved the correctness of our first algorithm.

We now prove the working the second algorithm,*i.e.,* BIN-POW-TEN(n) it is almost exactly same as the above except in the *basis* step it returns the actual value of 10 in *binary*. Also, the combined step in the inductive process we have the FAST-MULTIPLY(x, y) algorithm which is already proved.

So, all the requirements of the proof is satisfied and hence this second algorithm is also proved to be correct.

We now move to our final algorithm which combines the functionalities of the above two algorithms to solve our problem in the problem statement. This is the DEC-TO-BIN(x) algorithm. It has also the same structural proof as in the first algorithm. In the *basis* step *i.e.,* at $n = 1$ it simply return the actual binary value of the decimal number stored in the memory, so the basis step is proved to be correct. In the inductive step again we just use our helper algorithms which are already proved to be correct tp return the result.

So, this completes our proof and this section as well.

4 Justification of Solution

To convert decimal integer to binary we have used the divide and conquer paradigm. The reason behind using this technique is that the problem get divided into smaller problems and we find solutions to those smaller problems, combine them in next step and build our solution ground up.

In our initial approach of this divide and conquer algorithm for binary number multiplication we have the multiplication of the two numbers as follows:

$$xy = (2^{\frac{n}{2}}x_L + x_R)(2^{\frac{n}{2}}y_L + y_R)$$

where x_L, x_R are the two equal parts of x of length $\frac{n}{2}$ namely left half and right half, similarly for y. n is the number of bits in maximum of x or y.

$$\implies xy = 2^n x_L y_L + 2^{\frac{n}{2}}(x_L y_R + x_R y_L) + x_R y_R \tag{1}$$

The recurrence relation for the above equation if we write an algorithm for it is $T(n) = 4T(\frac{n}{2}) + \mathcal{O}(n)$.

The time-complexity upon solving the above equation (*discussed in the complexity section*) is $\approx \mathcal{O}(n^2)$. This is the time-complexity of the un-optimized algorithm, we can do even better than that.

In the eq.(1) we see that there is multiplication of 4 terms. There, exists a very clever observation which allows us to do the same multiplication of x and y in 3 operations. This is the new optimized approach. Upon observation we see that the term $(x_L y_R + x_R y_L)$ in equation (1) can be written as, $((x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R)$. This greatly reduced the number of multiplications made, only a few addition operations are introduced which takes lesser time as compared to another multiplication. Upon writing the recurrence relation for this algorithm we have the recurrence relation as $T(n) = 3T(\frac{n}{2}) + \mathcal{O}(n)$. Just the factor 4 in previous recurrence changes to 3. If we solve this relation then we have the time-complexity as $\mathcal{O}(n^{log_2 3}) \approx \mathcal{O}(n^{1.59})$.

From the above discussion we can see that this is a huge improvement from the previous approach. Just a small tweak had so large a impact on the complexity of the algorithm.

5 Time Complexity Analysis

Before we begin the analysis of the algorithm we will be using master's theorem for finding the time complexities of the recurrence relation, which is stated as follows:
If $T(n) = aT(\lceil \frac{n}{b} \rceil) + \mathcal{O}(n^d)$ for some constants $a > 0, b > 1$, and $d \geq 0$,then

$$T(n) = \begin{cases} \mathcal{O}(n^d), & if d > log_b a \\ \mathcal{O}(n^d log n), & if d = log_b a \\ \mathcal{O}(n^{log_b a}), & if d < log_b a \end{cases} \tag{2}$$

- We first analyze the time complexity of our first algorithm FAST-MULTIPLY(x, y). Let the cost of this algorithm is $T(n)$. In the *section* 2 we have already shown the cost of each step of this algorithm now we write the recurrence relation for it,which is as follows:

  $$T(n) = 3T(\frac{n}{2}) + \mathcal{O}(n) \tag{3}$$

  if we compare the above equation with that in equation (2), we have $d = 1$ and $log_2 3$ and $d < log_2 3$ so, from the third case of the master's theorem we have,

  $$T(n) = \mathcal{O}(n^{log_2 3}) \approx \mathcal{O}(n^{1.59})$$

  Next we move on to our second algorithm.

- We first analyze the time complexity of our second algorithm BIN-POW-TEN(n). Let the cost of this algorithm is $G(n)$. In the *section* 2 we have already shown the cost of each step of this algorithm now we write the recurrence relation for it,which is as follows:

  $$G(n) = G(\frac{n}{2}) + \mathcal{O}(n^{log_2 3}) \tag{4}$$

  if we compare the above equation with that in equation (2), we have $d = log_2 3$ and $log_2 1 = 0$ and $d > 0$ so, from the first case of the master's theorem we have,

  $$G(n) = \mathcal{O}(n^{log_2 3}) \approx \mathcal{O}(n^{1.59})$$

  Upon observation we can see that the complexity is exactly same as the first algorithm. Next we move on to our final and the algorithm that solves the problem given in our problem statement.

- We first analyze the time complexity of our last algorithm DEC-TO-BIN(x). Let the cost of this algorithm is $H(n)$. In the *section* 2 we have already shown the cost of each step of this algorithm now we write the recurrence relation for it,which is as follows:

  $$H(n) = 2H(\frac{n}{2}) + G(\frac{n}{2}) + T(\frac{n}{2}) \tag{5}$$

  $$\implies H(n) = 2H(\tfrac{n}{2}) + \mathcal{O}((\tfrac{n}{2})^{log_2 3}) + \mathcal{O}((\tfrac{n}{2})^{log_2 3})$$

The above equation simplifies to the following,

$$\implies H(n) = 2H(\tfrac{n}{2}) + \mathcal{O}(n^{log_2 3})$$

if we compare the above equation with that in equation (2), we have $d = log_2 3$ and $log_2 2 = 1$ and $d > 1$ so, from the first case of the master's theorem we have,

$$G(n) = \mathcal{O}(n^{log_2 3}) \approx \mathcal{O}(n^{1.59})$$

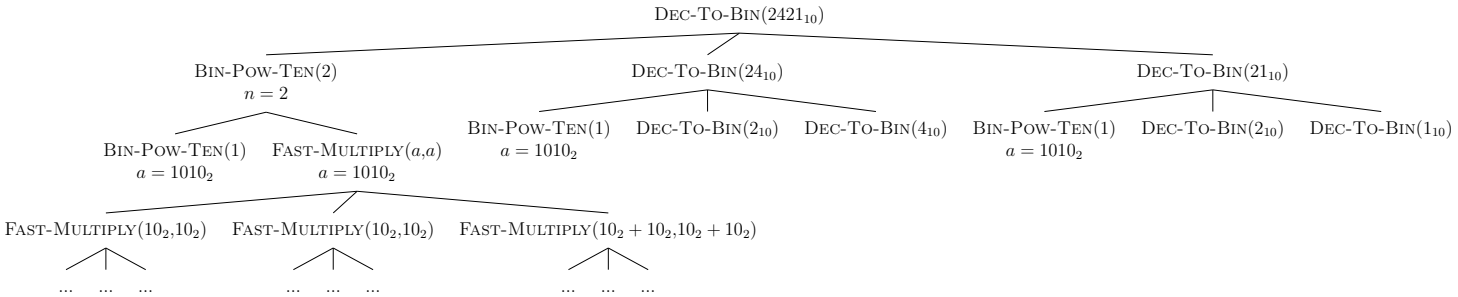
Upon observation we can see that the complexity is exactly same as the first two algorithm. Therefore the required complexity of our final algorithm is:

$$\boxed{\approx \mathcal{O}(n^{1.59})}$$

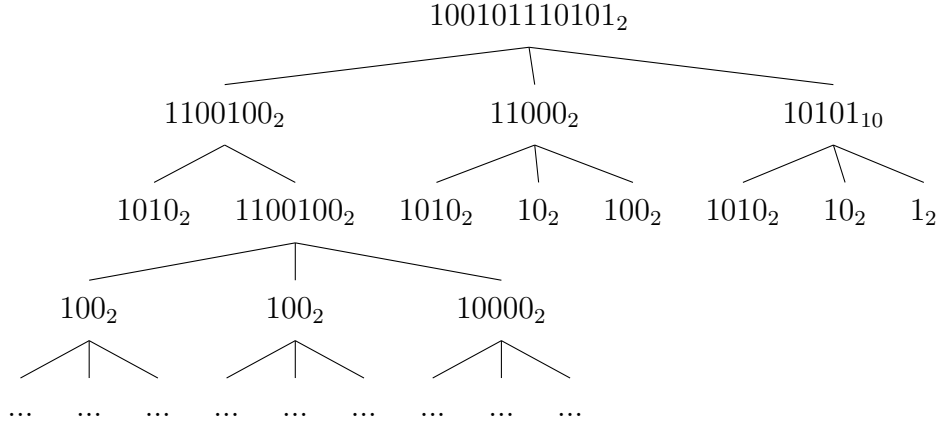
This concludes our analysis for this section.

6 Results and Discussion

In this section we demonstrate the running of DEC-TO-BIN(x) algorithm. Each call to the respective function is shown as the following.
For demonstrating the running of our algorithm we take the following input: 2421_{10} and we will convert it to binary. The recursion tree for the example is shown as follows



From the above tree we can clearly see each successive step of the algorithm. The answer is generated from the bottom most calls first, then in their just next combination state the answer to another sub-problem is created, in this fashion the entire tree in the top-most node generates answer to our original problem.



The answer tree for the the first recursion tree is shown in the above diagram. The answers are combined by the formulas given in the algorithm itself.

Had this algorithm not been optimized as given by eq. (1) we would have had a recursion tree with at-most 4 children, this would have affected the efficiency of the algorithm as we have already seen mathematically in the the complexity analysis section.

7 Drawbacks

We have developed this algorithm only for decimal number conversion (*i.e., base 10*), but this algorithm will not work for octal or hexadecimal numbers, there is room for improvement. As any input like DEC-TO-BIN(7_8) will give incorrect output. Otherwise for any decimal inputs the algorithm is flawless.