**COL100**

**Assignment - 2**

**Part 1**

**Algorithms**

Note: The numbers represented below will be used for the entire discussion. All of them are in binary form, unless specified otherwise.

The numbers and are represented as follows:

Where and belongs to , provided if and if .

is a single bit number, either 0 or 1.

is any arbitrary number in base 10, such that .

The functions required are:

1. The function returns the number having the bit concatenated to it times at its end.
2. The function returns the product of the binary number and a single bit .
3. The function returns the product of the two binary numbers and .

The algorithm is as follows:

1. If or , the product is 0.
2. Else:

Therefore, the product is:

**Proving the correctness**

1. The function is correct by definition.
2. :

Claim: It returns .

Proof:

Case I:

Case II:

Hence

1. :

Claim: It returns .

Proof:

It will be a proof by strong induction on b.

Base case:

For

Inductive hypothesis:

Let

Inductive Step:

Let .

Note:

All the numbers defined ahead would be in base 2 unless specified explicitly.

If

Then returns:

Where and

Note that

Moreover, for the rest of the . Therefore .

Now,

Note that as is same, therefore:

Therefore,

If :

Then returns:

Where and

Note that

Note that and

Therefore:

Therefore, by Principle of Mathematical Induction,

**Part 2**

**Addition Algorithm Helpers**

1. The function last(a) returns the last digit of the given binary number.
2. The function higher(a) returns the binary number left after removing the last digit.
3. The function add1(ai, bi, c) returns the sum of the three bits ai + bi + c.
4. The function returns the sum .

Where:

So, the sum of a and b is .

**Time Complexity**

1. The function is executed immediately without any delay.
2. The time complexity of and is as in all three, only single bit comparison is done a constant number of times whenever the function is called.
3. The time complexity of is as always 3 comparisons are used, each one for the three input bits.
4. Time complexity for add:

Case I:

For addition of every bit, where , it always takes time proportional to .

Note that when it checks for the bit, then the function stops in the best case, as b becomes 0. In the worst case, it goes till the step.

Therefore, for the former, the order is . But for the latter, it is . As , so the time complexity would be .

Case II:

For addition of every bit, where , it always takes time proportional to .

But here, the function always runs for times. Therefore, the time complexity is .

Therefore, by combining the two cases, we get that the time complexity = .

1. Time complexity for :

In the function, runs times. Therefore, the time complexity for this step is

Now, for the addition steps, it would call binary addition times. And for each addition call, the time complexity would be . Therefore, the time complexity for this step would be .

Therefore, the overall time complexity is:

Therefore, the time complexity of the algorithm is quadratic.

**Space Complexity**:

1. The space complexity for and is , where is the number of bits of input.
2. The space complexity for ) as the input has bits.
3. The space complexity for , where is number of bits of the input number.
4. The space complexity of .
5. Space complexity for :

Without loss of generality, let us suppose that .

The recursion for space complexity is:

As , therefore solving the above recursion relation, we get:

Therefore, the space complexity is .

1. Space complexity of :

In the function, the function runs times, but every run happens only after the previous run has finished. There are new variables formed, each storing at maximum digits. Therefore, the order of space complexity is .

For the next step, the join function uses at maximum space of .

For the new variables , the space required is:

For the addition, the binary addition would require at max space:

For the storage of product, the space required is .

Therefore, the space complexity is:

Therefore, the space complexity of the algorithm is quadratic.

**New Iterative Algorithm**

For this, a new helper function is needed:

The function removes the first digit of the number and returns the rest.

Now, the iterative algorithm is:

Where:

Therefore .