CERTIFICATE

Name : PRAVAR KATARIA **Class :**

Roll No : 2021CS10075 (ENTRY NO.) **Exam No.**

School IIT Delhi

This is certified to be the bonafide work of the student in the

_____ Laboratory during the academic

Year 20 /20 .

No of Practical Certified _____ out of _____ in the

Subject of _____

.....
Teacher In Charge

.....
Examiner's Signature

.....
Principal

Date:.....

School Rubber Stamp

(N.B: The candidate is expected to retain his/her journal till he/she passes in the subject.)

Part 1

Question 1 :

* Algorithm for m-bit * n-bit multiplication [binary]

If we observe for binary system, the multiplication by one bit is just the and of the bit with each bit of other number

(b is n bit & a is m bit)

for eg. $b_{n-1}\ b_{n-2} \cdots b_0 = b$

then $b * a_0$ is just (b_n and a_0), (b_{n-1} and a_0) -- (b_0 and a_0)

∴ This motivates us to define a function 'mega-and'

mega_and (a_0, b) : This gives the value $a_0 * b$

in effect it can be computed by

$(a_0 \wedge b_{n-1}), (a_0 \wedge b_{n-2}) \cdots (a_0 \wedge b_0)$

Now the multiplication of $m \times n$-bits is

c_0 = mega_and (a_0, b) [c_0 is a n bit numeral]

c_1 = mega_and (a_1, b) [c_1 is a n bit numeral]

⋮

c_{m-1} = mega_and (a_{m-1}, b) [c_{m-1} is a n bit numeral]

Now we assume that there exists a function which we call addU (add universal) which adds any 2 m, n bit numerals

Now define a function Lshift : (essentially a * 2)
which shifts the numeral one shift to the
left and replaces a 0/false in the empty bit

for eg- $Lshift(a_{m-1}, a_{m-2}, a_{m-3} \dots a_0)$

$= (a_{m-1}, a_{m-2}, a_{m-3} \dots a_1, a_0, 0)$

Now we define Lshift_n_times (n) : (essentially a * 2^n)
which for any arbitrary input n
applies the Lshift function n times

For eg,
applying Lshift_n_times (3,2) to $(a_1, a_0) = a$
output is $(a_1, a_0, 0, 0, 0)$

Now Let,

$d_0 = addU(C_0, Lshift\_n\_times(1, C_1))$

$d_1 = addU(d_0, Lshift\_n\_times(2, C_2))$

$d_2 = addU(d_1, Lshift\_n\_times(3, C_3))$

$d_{m-2} = addU(d_{m-3}, Lshift\ n\ times(m-1, C_{m-1}))$

Then return d_{m-2} as result of our multiplication

Prequel to Q2

Translation from binary to 2^{α} form. [value form]

$a_0 \rightarrow a_0$

$(a_1, a_0) \rightarrow 2a_1 + a_0$

$(a_2, a_1, a_0) \rightarrow 4a_2 + 2a_1 + a_0$

(The place at α is multiplied by $2^{\alpha-1}$)

$$a = \sum_{\alpha=1}^{\alpha=m} 2^{\alpha-1} a_{\alpha-1}$$

also the translation from value to binary

$$y = \sum_{\alpha=1}^{\alpha=m} 2^{x+(\alpha-1)} a_{\alpha-1}$$

$\therefore a_{\alpha-1}$ is at $(x+\alpha)$ place

$\Rightarrow (a_{\alpha-1}, a_{\alpha-2}, \ldots, a_0, \underbrace{0, 0, 0, 0, 0}_{x \text{ times}})$

Q2

We assume that addU is correct

Now check that megaand really computes $a_0 * b$.

for a_0 we want our result to be 0

lets check.

megaand $(0, b) = (0$ and $b_{n-1}, 0$ and b_{n-2} ... $0 \& b_0)$

$= (0, 0, 0, \ldots)$

(0 and Bool) = 0 for any bool

hence megaand $(0, b) = 0 = 0 * b$ = Checked

Now for $a_0 = 1$ we have to check if the result of is b itself.

(1 and Bool) = Bool for any bool

$\therefore$ megaand $(1, b) \equiv (1$ and $b_{n-1}, 1$ and b_{n-2} ... $1 \& b_0)$

$= (b_{n-1}, b_{n-2}, \ldots b_0) = b$

hence the correctness of megaand is checked

for

Lshift we can compare the definition & output hence Lshift does what we expect hence it is correct.

(essentially we want to multiply by 2)

$$2\left(\sum_{m=1}^{m-1} 2^{m-1} a_{m-1}\right) \equiv \left(\sum_{m=1}^{m=m} 2^m a_{m-1}\right)$$

$$= \downarrow$$

$(a_{m-1}, \ldots, 0)$

a_{m-1} at $(m+1)$ place; 0 at 1st place

Left shift n times also by the definition does what we expect it to i.e multiplying by 2^n. (multiplying by 2 n times)

$$\left(2^n \sum_{i=1}^{m} 2^{i-1} a_{i-1}\right) = \left(\sum_{i=1}^{m} 2^{n+i-1} a_{i-1}\right)$$

$$\Big(\underbrace{a_{m-1}, \ldots\ldots a_0}_{\text{at } m+n \text{ place}}, 0, 0, 0, 0\Big)$$

mult(a,b) :

c_0 = mega and $(a_0, b) \rightarrow a_0 * b$

c_1 = mega and $(a_1, b) \rightarrow a_1 * b$

Now d_0 = add(c_0, Lshy + 1 c_1)

= add U(c_0, $2 * c_1$)

$d_0 = c_0 + 2 * c_1$

Similarly $d_1 = c_0 + 2 * c_1 + 2 * c_2 \ldots\ldots$

$\Rightarrow d_{m-2} = c_0 + 2 * c_1 + 4 * c_2 \ldots\ldots 2^{m-1} c_{m-1}$

$d_{m-2} = a_0 + 2 * a_1 * b + 4 * a_2 * b \ldots\ldots 2^{m-1} * a_{m-1} * b$

$= b * (a_0 + 2 * a_1 + \ldots\ldots 2^{m-1} * a_{m-1})$

$= b * (a)$ Hence checked

Q4 We first begin by calculating the time complexity of add function. For that we first provide a rough algorithm for calculation of sum of 2 numerals (binary)

As defined in Q3 scrap, we use that here. (but it does not add false)

For eg. scrap$(b_{n-1}\ b_{n-2}\ b_{n-3} \dots b_0) = (b_{n-1}\ b_{n-2} \dots b_1)$
→ also join adds 2 tuples in the same order

We define a function
addU_aux(a, b, c) where a & b are arbitrary (m & n resp.) size numerals and c is 1 bit numeral.

addU_aux(a,b,c) :
- a, if b has no digits & c = 0
- b, if a has no digits & c = 0
- addU_aux(a, 1, 0) if b has no digits & c = 1
- addU_aux(b, 1, 0) if a has no digits & c = 1
- join(e, d_0) otherwise.

Now checking all the auxillas conditions takes O(1) time as at max we have to check 4 condition so the ~~order~~ time is always ≤ K (for some K)

e above is defined as addU_aux(scrap(a), scrap(b), d_1)

& d_1, d_0 = add(a_0, b_0, c_0) [3 bit addition]

a_0 & b_0 are the last bit in a_0 & b_0

where

a_2 a_{n-1} a_{n-2} —— a_1 a_0

b = b_{n-1} b_{n-2} —— b_1 b_0

The add1 function also require $O(1)$ time as specified

∴ order of Time required for addU_aux(a, b, c0)

is recursively

$O(1)$ + addU_aux(scrap(a), scrap(b), c

∴ Time(addU_aux) is just Time (addU_aux with one digits less)

$= O(1) + O(1) + O(1)$ —— ∈

full when digits of a or b extinguish

addU(a,b) = addU_aux (a, b, 0)

PAGE NO.
DATE

Now let us consider the worst case
Without loss of generality - let $m > n$

∴ digits of b will extinguish first

let the final result be stuck at evaluating

addU_aux (a_{new}, 0, 1)
(with m-n digits)

Now by the defn of the function

= addU_aux (a_{new}, 1, 0)

Now the defined e = addc (scrap(a), scrap(b), d_1)
where $d_1 d_0$ = add(a_0, 1, 0)
scrap b = (0) → no digits.
the value of d_1 can possibly be 1 if $a_0 = 1$
∴ the algorithm moves to evaluating

addU_aux (scrap(a), 1, 0) which is essentially the same

hence although the digits of b vanish at n steps
but the worst case scenario takes the algorithm
to m steps

Hence Time complexity (addU) = O(1) + O(1) ——

= O(max(n, m))

hence order of time complexity of addli(a,b)
is $O(\max(m,n))$

Now we calculate the time complexity for other functions

Time complexity for megaand(a_0, b)

is Time for $(a_0$ and $b_0) + (a_0$ and $b_1) - -$

$= O(1) + O(1) \underbrace{\text{---}}_{n \text{ times}} O(1)$

$= O(n)$

PAGE NO.
DATE

Now we return to the algorithm of mul4( ):

time required for calculation of c_0 = $O(n)$
" " c_1 " = $O(n)$
" " c_{m-1} " = $O(n)$

∴ total time required = $m \cdot O(n)$
for calculations = $O(mn)$
of all c_i's

L shift, scrap, L shift n times require ① time essentially
(bcz they are combination of removing rearranging and adding but) (as mentioned in assignment)

Time required for calculation of $d_0 = O(\max(n, n+1)) = O(n+1)$
also d_0 is at max $(n+2)$ but numeral.

∴ Time for $d_1 = O(\max(n+2, n+2)) = O(n+2)$
d_1 is at max $n+3$ but num

* Time for $d_2 = O(\max(n+3, n+3)) = O(n+3)$

∴ we observe that $T(d_i) = O(n+i+1)$
(* proove by induction) except for $i=0$

Therefore the total time required
$= O(mn)$ for calculation of c_{ij}s + Time for calculat^n of d_{ij}s

for d_{ij}s the time is $O(n+i+1)$, $i = n-2$

hence Σ Time$(d_{ij}) = \sum_{i=1} O(n+i+1)$

$$= O\left(nm + \frac{m^2}{2}\right)$$

$\therefore$ the final order of time complexity $= O\left(nm + \frac{m^2}{2}\right) + O(nm)$

$$= O\left(nm + \frac{m^2}{2}\right) \text{ as } \left(nm + \frac{m^2}{2} \geq nm\right)$$

$$= O\left(\frac{nm}{2} + \frac{nm}{2} + \frac{m^2}{2}\right)$$

$$= O\left(\frac{nm}{2}\right) + O\left(\frac{nm}{2} + \frac{m^2}{2}\right)$$

$$= O\left(\frac{nm + m^2}{2}\right)$$

$$= O(m \cdot \max(m, n))$$

For space complexity :

First we define a function $S(f(n))$ which is basically the space complexity for $f(n)$

$O(S(f(n)))$ denote the order (asymptotic) for space complexity of $f(n)$

Now we just calculate $S(addU\_aux(a,b,c))$ which is equivalent to $S(addU(a,b))$ in order

Now

$O(S(addU\_aux))$ is a function of m & n (the input size)

let this $O(S(addU\_aux)) = f(m,n)$

Now considering the worst case
let the algorithm proceed to the scrap step and hence

$$O(S(addU\_aux)\ scrap(a), scrap(b), d\ldots)) + S(\text{for storing } a \text{ \& } b \text{ as inputs}) = f(m,n)$$

in other words

$$f(m,n) = O(m) + O(n) + f(m-1, n-1)$$

DATE

Base case: The claim is true for m or n = 0

Induction hypothesis

Let f(m, n) be O(m*n)

$$\therefore O(mn - (m-1)(n-1)) = O(m+n+1)$$
$$= O(m) + O(n)$$

hence $S(\text{add } U(a,b)) = O(mn)$

Now lets calculate the space complexity for other steps.

To store the value of C_0 it requires n bits.

hence $(S(C_i)) = n$

hence total space to store all C_is $= O(n \times m)$

Now to calculate d_is

to store the values the calculated for d_i we need the space of O(numerals)

$$= O(n+1) \text{ -- -- } O(n+m-1) = O(nm + \frac{m^2}{2})$$

In addition to this additional space is required to calculate the addition

* Note: These values are temporary hence the memory is cleared when the f^n is called hence the maximum space required again

= the max [addU(c_0, c_i shifted once)

addU(d_{m-3}, c_{m-1} shifted m-1 times)

$$= \max \begin{cases} O(n*(n+1)) \\ \vdots \\ O((n+m-1)*(n+m)) \end{cases}$$

PAGE NO.
DATE

∴ Space required for all addition algorithms
$= O((n+m)^2)$

∴ total time required

$= O((n+m)^2) + O(mn + m^2/2) + O(nm)$

$= O(2 \cdot (n^2/2 + nm + m^2/2)) + O(mn + m^2/2)$

$= O((n+m)^2)$ (as $(n+m)^2 \geq mn + m^2/2$)

$= O(n^2 + m^2 + nm)$

$= O(2^* (n^2 + m^2))$

$= O(n^2) + O(m^2)$

$= O(\max(m^2, n^2))$

$= (O(\max(m, n)))^2$

Prequel to OS part 1

First we define some functions which will be used throughout

meg a and (a, b)
which returns (a and b_0, a and b_1 ——— a and b_n

lowest(b) gives b_0
where b_0 is the least valued bit
where $b = b_{n-1} b_{n-2}$ ——— b_0

~~highest (b) gives~~

shift_n_times(a, i)
which shifts the tuple / numeral a by i bits

for eg. L_shift_n_times (a, 3)
$= a_{m-1} a_{m-2}$ ——— $a_0, 0, 0, 0$.

scratch(b) which removes the lowest bit from the numeral b

i.e scratch(b) = $b_{n-1} b_{n-2}$ ——— b_1

OS part 1

PAGE NO.
DATE

we define a function itera which takes the input a(a is a m bit numerical), b (b is a n bit numerical) a temporary variable temp and a running variable i
(b0 is the lower of b)
This function itera works basically as a loop
n is the no. of digits in b & m in (a)
The def^n of itera is

itera (a, b, temp, i) {
  if b has no digits or i=0
    temp ,
  temp = addU (temp , c)
  where c = 2_shift_n_times (a, n-i)
  & d is megaand (b0, a)
  return itera (a, scratch (b), temp(p, if))
}

lower (b) ←

In effect the b is always changed hence the b0 is continuously changing from the original b0 to b1, to b2 to b3 ---- to bn-1

the (n-i) is the number of bits by how much d is shifted

We initialize i as n.
and temp as 0/false

Now the first bit multiplied and is shifted & both and added to temp

∴ temp after 1st step = temp + b0* a
(false)

= b0* a

PAGE NO.
DATE

temp after second step =
sum of temp + $b_1 * a$ shifted once
↓
lower (scratch (b))

$= 2 * b_1 * a + b_0 * a$

Hence finally the product y

$(2^{n-1} \cdot b_{n-1} + 2^{n-2} \cdot b_{n-2} + \cdots b_0) * a$

$= (a * b)$

So we finally define mul4i (a, b)
y b has n digits

mul4i(a,b) as

```
return (a, Mul(a, b, False, n))
```