EXPERIMENT 3

20CP209P – Design and Analysis of Algorithm Lab

Aim:

Use singly linked lists to implement integers of unlimited size. Each node of the list should store one digit of the integer. You should implement addition, subtraction, multiplication, and exponentiation operations. Limit exponents to be positive integers.

What is the asymptotic running time for each of your operations, expressed in terms of the number of digits for the two operands of each function?

Code:

Addition:

```
Node* lladditer(Node* head1, Node* head2, int carry)
{
   Node* revhead1 = reverselist(head1);
   // printlist(revhead1);
   Node* revhead2 = reverselist(head2);
   // printlist(revhead2);

   Node* result = NULL;

   int sum;

   while (revhead1 != NULL || revhead2 != NULL || carry)
   {
      sum = carry;

      if (revhead1)
      {
         sum += revhead1->data;
         revhead1 = revhead1->next;
      }

      if (revhead2)
      {
         sum += revhead2->data;
         revhead2 = revhead2->next;
      }

      carry = sum / 10;

      Node* newnode = create_node(sum % 10);

      newnode->next = result;
```

```
        result = newnode;
    }

    // printlist(result);

    if (result != NULL)
    {
        int size = 0;
        Node* temp = result;
        while (temp != NULL) {
            size++;
            temp = temp->next;
        }
        result->size = size;
    }

    return result;
}
```

Output:

```
PS B:\sem4\23bcp153_daa\lab3> gcc sllarithops.c -o sllarithops
PS B:\sem4\23bcp153_daa\lab3> ./sllarithops
Enter the first number: 999
Enter the second number: 999
The first number is: 9 9 9
num size: 3
The second number is: 9 9 9
num size: 3
9 9 9
9 9 9
1 9 9 8
```

Algorithm:

Arithmetic operations

Addition()

```
rev1 = rev(head1);
rev2 = rev(head2)
res = NULL
sum = 0
while(rev1 != NULL || rev2 != NULL || carry)
        sum = carry
        if(rev1)
                sum += rev1.data
                rev1 = rev1.next
        if (rev2)
                sum += rev2.data
                rev2 = rev2.next
        carry = sum / 10;
        newnode = create_node(sum % 10);
        newnode.next = result
return res;
```

Complexity Analysis:

Cost	Times
c_1	n
c_2	n
c_3	1
c_4	1
c_5	$n+1$ $\because \sum_{i=1}^{n} 1 + 1$
c_6	$\cancel{\sum_{i=1}^{n} i}$ n
c_7	$\cancel{\sum_{i=1}^{n}}$ n
c_8	n
c_9	n
c_{10}	n
c_{11}	n
c_{12}	n
c_{13}	n
c_{14}	n
c_{15}	n
c_{16}	1

$$T(n) = c_1(n) + c_2(n) + c_3 + c_4 + c_5(n+1) + c_6(n) + c_7(n) + c_8(n) + c_9(n) + c_{10}(n) + c_{11}(n) + c_{12}(n) + c_{13}(n) + c_{14}(n) + c_{15}(n) + c_{16}$$

$$= (c_1 + c_2 + c_5 + c_6 + c_7 + c_8 + c_9 + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} + c_{15}) + (c_3 + c_4 + c_5 + c_{16})$$

$$= an + b = O(n)$$

Code:

Subtraction:

```
Node* llsubiter(Node* head1, Node* head2, int borrow)
{
   Node* revhead1 = reverselist(head1);
   // printlist(revhead1);
   Node* revhead2 = reverselist(head2);
```

```
    // printlist(revhead2);

    Node* result = NULL;

    int diff;

    while (revhead1 || revhead2)
    {
        diff = borrow;
        if (revhead1)
        {
            diff += revhead1->data;
            revhead1 = revhead1->next;
        }

        if (revhead2)
        {
            if (diff >= revhead2->data)
            {
                diff -= revhead2->data;
            }
            else
            {
                borrow = -1;
                int fordiff = 10 - revhead2->data;
                diff += fordiff;
            }
            revhead2 = revhead2->next;
        }

        Node* newnode = create_node(diff);

        newnode->next = result;
        result = newnode;
    }

    if (result != NULL)
    {
        int size = 0;
        Node* temp = result;
        while (temp != NULL) {
            size++;
            temp = temp->next;
        }
        result->size = size;
    }

    return result;
}
```

Output:

```
PS B:\sem4\23bcp153_daa\lab3> gcc sllarithops.c -o sllarithops
PS B:\sem4\23bcp153_daa\lab3> ./sllarithops
Enter the first number: 1000
Enter the second number: 999
The first number is: 1 0 0 0
num size: 4
The second number is: 9 9 9
num size: 3
1 0 0 0
9 9 9
0 0 0 1
PS B:\sem4\23bcp153_daa\lab3>
```

Algorithm:

Subtraction Analysis

Page No.
Date: / /

Algorithm.

```
rev1 = rev(head1)
rev2 = rev(head2)
res = NULL
diff = 0
while (rev1 || rev2)
    diff = borrow
    if (rev1)
        diff += rev1.data
        rev1 = rev1.next
    if (rev2)
        if (diff >= rev2.data)
            diff -= rev2.data
        else
            borrow = -1
            int fordiff = 10 - rev2.data
            diff += fordiff
        rev2 = rev2.next
    newnode = create_node(diff)
    newnode.next = res
    res = newnode
return res
```

Complexity Analysis:

Cost	Times
C_1	n
C_2	n
C_3	1
C_4	1
C_5	$n+1$
C_6	$\sum_{i=1}^{n} 1 = n$
C_7	$\sum_{i=1}^{n} 1 = n$
C_8	n
C_9	n
C_{10}	n
C_{11}	n } if
C_{12}	n
C_{13}	n
C_{14}	n } else
C_{15}	n
C_{16}	n
C_{17}	n
C_{18}	n
C_{19}	n
C_{20}	n
C_{21}	1

$\boxed{\begin{array}{r} 1000 \\ -999 \end{array}} \rightarrow n=4$

$$T(n) = C_1(n) + C_2(n) + C_3(1) + C_4(1) + C_5(n+1) + C_6(n) + C_7(n) + C_8(n) + C_9(n) + C_{10}(n) + C_{11}(n) + C_{12}(n) + C_{13}(n) + C_{14}(n) + C_{15}(n) + C_{16}(n) + C_{17}(n) + C_{18}(n) + C_{19}(n) + C_{20}(n) + C_{21}$$

$$= an + b$$

$$= O(n)$$

Code:

Multiplication:

```
Node* llmuliter(Node* head1, Node* head2, int carry)
{
   Node* revhead1 = reverselist(head1);
   // printlist(revhead1);
   Node* revhead2 = reverselist(head2);
   // printlist(revhead2);

   Node* final_result = NULL;

   int product;

   Node* trav1 = revhead1;
   Node* trav2 = revhead2;

   int pad_count = -1;

   while (trav2)
   {
      Node* result = NULL;
      // since we are adding null - we need to change the if condition inside padding function
      // result = add_padding_back(result, ++pad_count);
      // so i just found out that the add padding back function was useless
      result = add_padding(result, ++pad_count);
      trav1 = revhead1;
      carry = 0;
      while (trav1)
      {
         // printlist(result);
         // result = add_padding(result, ++pad_count);
         product = carry;
         product += trav1->data * trav2->data;
         carry = product / 10;

         Node* newnode = create_node(product % 10);

         newnode->next = result;
         result = newnode;

         trav1 = trav1->next;
      }

      // bhai carry to dekho
      if (carry > 0)
      {
         Node* newnode = create_node(carry);
         newnode->next = result;
         result = newnode;
      }
```

```
        // printlist(result);
        final_result = lladditer(final_result, result, 0);
        trav2 = trav2->next;
    }

    // somehow reverselist function is changing head1 and just keeping it to be its first node
    // i.e. for 123 - it is making it 1
    // so i am retrieving head1 again with the reversed list
    // although i don't think this is good practice - it is working
    // i suspect that this is due to the fact that in reverselist function we are taking current = head
    // which may be changing the head - unintentionally - deepseek - deepthink r1 can help identify
that
    // another approach could be to first create a copy of the head and then reverse both the lists
    // also freelist function needs to be implemented: 22:34 04-02-2025
    head1 = reverselist(revhead1);
    head2 = reverselist(revhead2);

    return final_result;
}
```

Output:

```
PS B:\sem4\23bcp153_daa\lab3> gcc sllarithops.c -o sllarithops
PS B:\sem4\23bcp153_daa\lab3> ./sllarithops
Enter the first number: 999
Enter the second number: 999
The first number is: 9 9 9
num size: 3
The second number is: 9 9 9
num size: 3
9 9 9
9 9 9
9 9 8 0 0 1
PS B:\sem4\23bcp153_daa\lab3>
```

Code:

Exponential:

```
Node* llexpiter(Node* head, int power)
{
    if (power < 0)
    {
        printf("Power less than 1 not supported");
        return NULL;
    }
    else if (power == 0)
    {
        return create_node(1);
    }
```

```
    Node* result = create_node(1);
    for (int i = 0; i < power; i++)
    {
        result = llmuliter(result, head, 0);
    }

    return result;
}
```

Output:

```
PS B:\sem4\23bcp153_daa\lab3> gcc sllarithops.c -o sllarithops
PS B:\sem4\23bcp153_daa\lab3> ./sllarithops
Enter the first number: 2
Enter the second number: 9
The first number is: 2
num size: 1
The second number is: 9
num size: 1
2
9
5 1 2
PS B:\sem4\23bcp153_daa\lab3>
```