Ahsania Mission University of Science & Technology

Lab Report

Course Code: CSE 2202

Course Title: Computer Algorithm Sessional

Experiment No: 02

Experiment Date: 12.02.25

Submitted By:

Mehrin Nusrat Chowdhury

Roll: 1012320005101026

1st Batch, 2nd Year, 1st Semester

Department of Computer Science and Engineering

Ahsania Mission University of Science & Technology

Submitted To:

Md. Fahim Faisal

Lecturer,

Department of Computer Science and Engineering

Faculty of Engineering, Ahsania Mission University of Science & Technology

Submission Date: 19.02.25

Task 01: A C++ program that will implement the Grade School Multiplication Algorithm and calculate the running time of the program.

Theory:

Grade school multiplication is a straightforward method, commonly taught in primary education, where numbers are multiplied digit-by-digit from right to left. This algorithm involves the multiplication of individual digits and the summation of the intermediate products, accounting for positional values and carrying over as needed. Implementing this algorithm in programming involves storing numbers in arrays to handle large inputs, performing nested loops for the multiplication, and adjusting for carries. We also measure the running time using the clock() function to understand the computational efficiency.

Code:

```
#include <iostream>
#include <cstring> // For memset
#include<ctime>
using namespace std;

#define MAX 200 // Maximum digits to handle large numbers

class BigIntMultiplication
{
private:
   int numA[MAX], numB[MAX], result[MAX];
   int lenA, lenB;

public:
   // Constructor to initialize arrays
   BigIntMultiplication()
   {
      memset(numA, 0, sizeof(numA));
      memset(numB, 0, sizeof(numB));
      memset(result, 0, sizeof(result));
      lenA = lenB = 0;
   }

   // Function to convert an integer into a digit array (least significant digit first)
   void storeNumber(int num, int arr[], int &length)
   {
      while (num > 0) {
```

```
        arr[length++] = num % 10; // storing digit
        num /= 10;
    }
}

// Function to multiply two integers using grade school multiplication
void multiply(int A, int B)
{
    if (A == 0 || B == 0)
    {
        cout << "0" << endl; // If either number is 0, the product is 0
        return;
    }

    // Store numbers in digit arrays
    storeNumber(A, numA, lenA);
    storeNumber(B, numB, lenB);

    // Perform grade school multiplication
    for (int i = 0; i < lenB; i++)
    {
        for (int j = 0; j < lenA; j++)
        {
            result[i + j] += numB[i] * numA[j];
            result[i + j + 1] += result[i + j] / 10; // carry
            result[i + j] %= 10; // remainder
        }
    }
    printResult();
}

// Function to print the final result
void printResult()
{
    int lenResult = lenA + lenB;
    while (lenResult > 1 && result[lenResult - 1] == 0)
    {
        lenResult--; // Remove leading zeros
    }

    // Print the result in correct order (reverse of storage)
    for (int i = lenResult - 1; i >= 0; i--)
    {
        cout << result[i];
    }
    cout << endl;
```

```
    }
};

// Driver Code
int main() {
    int A, B;
    cout << "Enter two integers: ";
    cin >> A >> B;

    BigIntMultiplication multiplier;

    clock_t startT = clock();

    cout << "Product: ";
    multiplier.multiply(A, B);

    clock_t endT = clock();

    double duration;
    duration = (double)(endT - startT) / CLOCKS_PER_SEC;
    cout << "\nDuration: " << duration << " seconds";

    return 0;
}
```

Output:

```
Enter two integers: 123 45
Product: 5535

Duration: 0.002 seconds
Process returned 0 (0x0)   execution time : 4.492 s
Press any key to continue.
```

Conclusion:

The grade school multiplication algorithm, although simple and easy to implement, may not be the most efficient for very large numbers due to its $O(n^2)$ time complexity. The running time calculation reveals how the algorithm performs with different input sizes, and the results indicate its practicality for small to moderate inputs. However, for significantly large inputs, more efficient algorithms may be required.

Task 02: A C++ program that will implement Karatsuba Multiplication Algorithm and calculate the running time of the program.

Theory:

The Karatsuba multiplication algorithm is a divide-and-conquer approach that reduces the multiplication of two n-digit numbers to at most three multiplications of $\frac{n}{2}$ digit numbers. This method breaks down each number into two halves and recursively computes the products and sums. The key formula for Karatsuba's algorithm is:

$$ac.10^{2m} + \big((a+b)(c+d) - ac - bd\big).10^{m} + bd,$$

where m is half the number of digits. This approach offers a faster time complexity of O($n^{1.585}$) compared to the traditional O(n^2) method.

The algorithm's performance is also measured using the clock() function.

Code:

```
#include<iostream>
#include<cmath>
#include<ctime>
using namespace std;

int karatsuba(int x , int y)
{
   if(x < 10 || y < 10)
   {
      return x * y;
   }
   int n = max((int)log10(x) + 1 , (int)log10(y) + 1);
   int m = n / 2;

   //Dividing the numbers equally
   int a = x / pow(10 , m);
   int b = x % (int)pow(10 , m);
   int c = y / pow(10 , m);
   int d = y % (int)pow(10 , m);

   //Formula
   int z0 = karatsuba(b , d);
   int z1 = karatsuba((a + b) , (c + d));
   int z2 = karatsuba(a , c);
```

```
    return (z2 * pow(10 , (2 * m))) + ((z1 - z2 - z0) * pow(10 , m)) + z0;
}

int main()
{
    clock_t start;
    double duration;

    start = clock();

    int num1;
    cout << "Enter first number: ";
    cin >> num1;
    int num2;
    cout << "Enter second number: ";
    cin >> num2;

    int result = karatsuba(num1 , num2);
    cout << "The product: " << result;

    duration = (clock() - start) / (double)CLOCKS_PER_SEC;
    cout << "\nDuration: " << duration << " seconds";
    return 0;
}
```

Output:

```
Enter first number: 123
Enter second number: 45
The product: 5535
Duration: 11.977 seconds
Process returned 0 (0x0)   execution time : 12.038 s
Press any key to continue.
```

Conclusion:

Karatsuba's algorithm significantly improves the efficiency of multiplication operations for large numbers compared to the grade school method. The reduced number of recursive multiplications demonstrates its effectiveness. The running time measurements validate the theoretical time complexity, showcasing Karatsuba's suitability for applications requiring large number multiplications with better performance than traditional methods.

Task 03: A C++ program that will implement Insertion Sort Algorithm.

Theory:

Insertion sort is a simple and intuitive sorting algorithm that builds the final sorted array one item at a time. It iterates through the array and, for each element, it compares the current element with the largest value in the sorted array. If the current element is smaller, it finds the correct position within the sorted array and inserts it there. This process is repeated until the entire array is sorted. Insertion sort is known for its efficiency with small datasets and nearly sorted arrays, having an average and worst-case time complexity of $O(n^2)$.

Code:

```
#include<iostream>
using namespace std;

int main()
{
   int n;
   cout << "Enter the size of the numbers: ";
   cin >> n;

   int A[n];

   cout << "Numbers: ";
   for(int i = 0 ; i < n ; i++)
   {
      cin >> A[i];
   }

   for(int i = 1 ; i < n ; i++)
   {
      int key;
      key = A[i];

      int j;
      j = i - 1;

      while(j >= 0 && key < A[j])
      {
         A[j + 1] = A[j];
         j = j - 1;
```

```
        }
        A[j + 1] = key;
    }

    cout << "Sorted: ";
    for(int i = 0 ; i < n ; i++)
    {
        cout << A[i] << " ";
    }

    return 0;

}
```

Output:

```
Enter the size of the numbers: 10
Numbers: 3 6 1 9 2 2 4 7 0 5
Sorted: 0 1 2 2 3 4 5 6 7 9
Process returned 0 (0x0)   execution time : 41.163 s
Press any key to continue.
```

Conclusion:

Insertion sort is an efficient algorithm for sorting small datasets or arrays that are already partially sorted. Its simplicity and ease of implementation make it a practical choice for such scenarios. However, for larger datasets, its $O(n^2)$ time complexity can be a disadvantage, necessitating the use of more advanced sorting algorithms like QuickSort or MergeSort for improved performance.

Task 04: A C++ program that will compare the running times of the Grade School Multiplication and Karatsuba Multiplication Algorithms.

Theory:

This task involves comparing the running times of two multiplication algorithms: grade school multiplication and Karatsuba multiplication. Grade school multiplication is a direct approach, while Karatsuba employs a divide-and-conquer technique for efficiency. By implementing both algorithms and measuring their execution times on the same inputs, we can evaluate their performance differences. The running times are recorded using the clock() function, allowing for a direct comparison of computational efficiency.

Code:

```
#include <iostream>
#include <cstring> // For memset
#include<ctime>
#include<cmath>
using namespace std;

#define MAX 200 // Maximum digits to handle large numbers

//Grade school
class BigIntMultiplication
{
private:
   int numA[MAX], numB[MAX], result[MAX];
   int lenA, lenB;

public:
   // Constructor to initialize arrays
   BigIntMultiplication()
   {
      memset(numA, 0, sizeof(numA));
      memset(numB, 0, sizeof(numB));
      memset(result, 0, sizeof(result));
      lenA = lenB = 0;
   }

   // Function to convert an integer into a digit array (least significant digit first)
   void storeNumber(int num, int arr[], int &length)
   {
```

```
    while (num > 0) {
        arr[length++] = num % 10; // storing digit
        num /= 10;
    }
}

// Function to multiply two integers using grade school multiplication
void multiply(int A, int B)
{
    if (A == 0 || B == 0)
    {
        cout << "0" << endl; // If either number is 0, the product is 0
        return;
    }

    // Store numbers in digit arrays
    storeNumber(A, numA, lenA);
    storeNumber(B, numB, lenB);

    // Perform grade school multiplication
    for (int i = 0; i < lenB; i++)
    {
        for (int j = 0; j < lenA; j++)
        {
            result[i + j] += numB[i] * numA[j];
            result[i + j + 1] += result[i + j] / 10; // carry
            result[i + j] %= 10; // remainder
        }
    }
    printResult();
}

// Function to print the final result
void printResult()
{
    int lenResult = lenA + lenB;
    while (lenResult > 1 && result[lenResult - 1] == 0)
    {
        lenResult--; // Remove leading zeros
    }

    // Print the result in correct order (reverse of storage)
    for (int i = lenResult - 1; i >= 0; i--)
    {
        cout << result[i];
    }
```

```
        cout << endl;
    }
};

//Karatsuba
int karatsuba(int x , int y)
{
    if(x < 10 || y < 10)
    {
        return x * y;
    }
    int n = max((int)log10(x) + 1 , (int)log10(y) + 1);
    int m = n / 2;

    //Dividing the numbers equally
    int a = x / pow(10 , m);
    int b = x % (int)pow(10 , m);
    int c = y / pow(10 , m);
    int d = y % (int)pow(10 , m);

    //Formula
    int z0 = karatsuba(b , d);
    int z1 = karatsuba((a + b) , (c + d));
    int z2 = karatsuba(a , c);

    return (z2 * pow(10 , (2 * m))) + ((z1 - z2 - z0) * pow(10 , m)) + z0;
}

int main()
{
    int A, B;
    cout << "Enter two integers: ";
    cin >> A >> B;

    BigIntMultiplication multiplier;

    clock_t startT = clock();

    cout << "Product Using Grade School Multiplication: ";
    multiplier.multiply(A, B);

    clock_t endT = clock();

    double duration_grade_scool = (double)(endT - startT) / CLOCKS_PER_SEC;
    cout << "Run Time For Grade School Multiplication: " << duration_grade_scool << "
seconds\n";
```

```
    clock_t startK = clock();

    int result = karatsuba(A , B);
    cout << "Product Using Karatsuba Multiplication: " << result << "\n";

    clock_t endK = clock();

    double duration_karatsuba = (double)(endK- startK) / CLOCKS_PER_SEC;
    cout << "Run Time For Karatsuba Multiplication: " << duration_karatsuba << "
seconds\n";

    double r = (abs(duration_grade_scool - duration_karatsuba));
    cout << "Compared Time: " << r << "seconds\n";

    return 0;
}
```

Output:

```
Enter two integers: 46782 1237
Product Using Grade School Multiplication: 57869334
Run Time For Grade School Multiplication: 0.002 seconds
Product Using Karatsuba Multiplication: 57869334
Run Time For Karatsuba Multiplication: 0 seconds
Compared Time: 0.002seconds

Process returned 0 (0x0)   execution time : 9.702 s
Press any key to continue.
```

Conclusion:

The comparison between grade school and Karatsuba multiplication algorithms reveals the significant performance advantage of Karatsuba's method for large inputs. The results confirm the theoretical time complexities, with Karatsuba's algorithm consistently outperforming the grade school method in terms of running time. This comparison underscores the importance of choosing appropriate algorithms based on input size and computational efficiency requirements.

Task 05: A C++ program that will multiply two Large Integers ($0 \leq X, Y < 10^{250}$).

Theory:

When dealing with extremely large integers, traditional data types are insufficient due to their limited storage capacity. This task involves multiplying two large numbers represented as strings, which allows for handling numbers up to 10^250. The multiplication is carried out using the grade school method, where digits are multiplied individually, and carries are managed explicitly. This approach leverages arrays to store intermediate results, ensuring accurate computation of large products. The algorithm also includes a mechanism to eliminate leading zeros from the final output. The running time is calculated to understand the efficiency of the implementation.

Code:

```
#include <iostream>
#include <cstring> // For memset
#include<ctime>
using namespace std;

#define MAX 500 // Maximum digits to handle large numbers

class BigIntMultiplication
{
private:
   int numA[MAX];
   int numB[MAX];
   int result[MAX * 2]; //250*2=500
   //int lenA, lenB;

public:
   // Constructor to initialize arrays
   BigIntMultiplication()
   {
      memset(numA, 0, sizeof(numA));
      memset(numB, 0, sizeof(numB));
      memset(result, 0, sizeof(result));
      //lenA = lenB = 0;
   }

   // Function to multiply two large integers represented as strings
```

```
void multiply(const string &numA , const string &numB)
{
   int lenA = numA.size();
   int lenB = numB.size();

   if (lenA == 0 || lenB == 0 || numA == "0" || numB == "0")
   {
      cout << "0" << endl; // If either number is 0, the product is 0
      return;
   }

   // Perform grade school multiplication
   for (int i = lenA - 1 ; i >= 0; i--)
   {
      for (int j = lenB - 1; j >= 0; j--)
      {
         int product = (numA[i] - '0') * (numB[j] - '0');
         int sum = product + result[i + j + 1];
         result[i + j + 1] = sum % 10;
         result[i + j] += sum / 10;
      }
   }
   printResult(lenA + lenB);
}

// Function to print the final result
void printResult(int lenResult)
{
   int i = 0;
   while(i < lenResult && result[i] == 0)
   {
      i++;
   }
   if(i == lenResult)
   {
      cout << "0";
   }
   else
   {
      for(; i <lenResult ; i++)
      {
         cout << result[i];
      }
   }
   cout << "\n";
}
```

```
};


// Driver Code
int main()
{
    string X;
    cout << " Enter number X: ";
    cin >> X;

    string Y;
    cout << " Enter number Y: ";
    cin >> Y;


    BigIntMultiplication multiplier;

    clock_t startT = clock();

    cout << "Product: ";
    multiplier.multiply(X, Y);

    clock_t endT = clock();

    double duration;
    duration = (double)(endT - startT) / CLOCKS_PER_SEC;
    cout << "\nDuration: " << duration << " seconds";

    return 0;
}
```

Output:

```
 Enter number X: 12345678901234567
 Enter number Y: 12398653134001322
Product: 153069790400165961591190410097574

Duration: 0.002 seconds
Process returned 0 (0x0)   execution time : 13.566 s
Press any key to continue.
```

Conclusion:

This implementation successfully handles the multiplication of extremely large integers by treating them as strings. The use of arrays to manage digits and results ensures that the multiplication process is both accurate and manageable. Although the grade school method is not the most efficient for very large numbers, it demonstrates the feasibility of performing such operations within reasonable time frames. The measured running time provides insights into the algorithm's performance, highlighting the potential need for more advanced techniques for even larger inputs or faster computations.