Report for Assignment 1

Deep Kiran Shroti
Roll Number- 12EC35013

March 9, 2016

Multiplication of large integers using FFT method

Assignment Statement Using the FFT/IFFT technique, you are required to multiply long integers

1 Implementation Details

1. **First Step: Generation of a random integer of a length specified by the user**
   To generate large integers such as of 1024 digits a **inputNumTag** struct is used

   - **num*, which is a pointer to an dynamic array, the array can be assigned memory according to the need of the user.

   During the execution of the code, a user is asked to enter the number of digits of the large integer to be multiplied. Then a random integer of the user specified length is generated. The generated number is also padded with required number of zeros to make FFT possible

2. **Second Step: FFT Calculation**
   FFT is realize using DFT. A recursive algorithm is applied to get FFT of the two randomly generated numbers.

   pseudo-code:

```
void fft(int n, complexType A[], complexType F[]){
        { int,i, k,shift = n/2;}
    if (n == 1){
        F[0] = A[0];
        return;
    }
    complexType omega,omegaK, E[n/2], O[n/2], EF[n/2], OF[n/2] ;
```

```
    for (i = 0 , k = 0 ; i < n ; k = k+1, i = i+2){
        E[k] = A[i];
        O[k] = A[i+1];
    }
    fft(n/2, E , EF);
    fft(n/2, O , OF);
    omega.re = cos(2*PI/n); omega.im = -sin(2*PI/n);
    omegaK = omega;
    for(i = 0; i < n/2; i++, omegaK = omegaK*omega){
        F[i] = EF[i]+ omegaK*OF[i];
        F[i+shift] = EF[i] - omegaK*OF[i];
}
```

3. **Third Step: Point-wise Multiplication** A point wise multiplication is done for the FFT calculated from the two random numbers generated.

   Fourth Step: IFFT Calculation

```
void ifft(int n, complexType A[], complexType F[]){
    int i,k,shift = n/2;
    if (n == 1){
        F[0] = A[0];
        return;
    }
    complexType omega,omegaK, E[n/2], O[n/2], EF[n/2], OF[n/2] ;
    for (i = 0 , k = 0 ; i < n ; k = k+1, i = i+2){
        E[k] = A[i];
        O[k] = A[i+1];
    }
    ifft(n/2, E , EF);
    ifft(n/2, O , OF);
    omega.re = cos(2*PI/n); omega.im = sin(2*PI/n);
    omegaK = omega;
    for(i = 0; i < n/2; i++, omegaK = omegaK*omega){
        F[i] = EF[i]+ omegaK*OF[i];
        F[i+shift] = EF[i] - omegaK*OF[i];
    }
}
```

   Fifth Step: Recombining The inverse FFT coefficients are combined to obtain the final product of the two randomly generated numbers.

2 Conclusion

1. The algorithm is an improvement over normal polynomial multiplication, which has a time complexity of $\mathcal{O}(n^2)$.

2. In this algorithm, we have FFT method which uses Divide and Conquer strategy, which itself has a time complexity of $\mathcal{O}(n \log n)$ and an added time for point-to-point multiplication giving an additional time complexity of $\mathcal{O}(n)$,thus improving the overall time complexity.