Name	**Shubham Golwal**
UID no.	**2020300015**
Experiment No.	**07**
AIM:	To obtain Linear filtering using Fast Fourier Transform (FFT) of the given L point sequence x[n] and M-point sequence h[n] using C-Language
Objective:	(A) To obtain Linear Convolution via Circular Convolution using Fourier Transform (FFT) by means of Overlap-Add Method of the given L point sequence x[n] and M-point sequence h[n] using C-Language (B) To obtain Linear Convolution via Circular Convolution using Fourier Transform (FFT) by means of Overlap-Save Method of the given L point sequence x[n] and M-point sequence h[n] using C-Language
Software used:	C- compiler
Algorithm	Overlap-Add Method 1. Given two input sequences x(n) and h(n), with lengths N and M respectively, and a block length L. 2. Pad both sequences with zeros to length L + M - 1, if necessary, so that they have the same length. 3. Divide the input sequence x(n) into blocks of length L, with an overlap of M - 1 samples between adjacent blocks. 4. For each block of x(n), perform circular convolution with the filter h(n) of length L, to obtain a block of length L+M-1 samples using the FFT convolution algorithm. 5. Save the output block from step 4. 6. Shift the input block by L - M + 1 samples to the right, and repeat steps 4-5 until the end of the input sequence is reached. 7. Concatenate the output blocks obtained from steps 4-6 to obtain the final output sequence.
Code:	Overlap-Add Method #include<stdio.h> #include<math.h> #define max 64 void Fast_Circular_Convolve(float *x, int N, float *h, float *y);

```
void DITFFT_8_Point( int N, float x[8][2], float t[8][2]);
void DITFFT_4_Point( int N, float x[4][2], float t[4][2]);
main() {
int i,j,k,SigLen,M,N,L;
float x[max],p[max],t[max],y[max]; float
h[max];
for(i=0;i<max;i++) // Initialization
{
x[i]=0; h[i]=0; y[i]=0; p[i]=0;
}
//-------------------- INPUT ---------------------------------printf("\n Enter the length of
x[n] = : "); scanf("%d",&SigLen);
printf( "\n Enter the values of x[n] : ");
for(i=0;i<SigLen;i++)
{
scanf("%f",&x[i]);
}
printf("\n\n Enter the length of h[n] M = : "); scanf("%d",&M);
printf( "\n Enter the values of h[n] : ");
for(i=0;i<M;i++)
{
scanf("%f",&h[i]);
}

printf("\n\n x[n] = " );
for(i=0;i<SigLen;i++) printf("
%4.2f ",x[i]); printf("\n\n h[n]
= " ); for(i=0;i<M;i++)
printf(" %4.2f ",h[i]);
//------------------------ OAM
----------------------------------------
N = 8; // Assumption
L = N - M + 1; // Length of decomposed x[n] printf( "\n\n length
of decomposed input Signal : L = %d \n ", L); printf( "\n length of
decomposed output Signal : N = %d \n ", N); j = 0;
for (k=0; k<SigLen; k+=L)
{ for(i=0; i<L; i++) p[i] = x[k+i]; // Splitting of x[n]
Fast_Circular_Convolve(p, N, h, t); // Lc By CC by FFT
for(i=0; i<N; i++)
{
```

```
y[k+i] = y[k+i]+t[i]; // Overap Add
} j++;
printf("\n\n\n x%d[n] = ", j ); // Decomposed x[n]
for(i=0;i< N; i++) printf(" %4.2f ",p[i]);
printf("\n\n y%d[n] = ",j ); // Decomosed y[n]
for(i=0;i< N;i++) printf(" %4.2f ",t[i]); }
//---------------------------------------------------------------
---
printf("\n\n Linear Convolution Output using Over Add Method " );
printf("\n y[n] = " ); // Result of LC for(i=0;i<(SigLen+M-1);i++)
printf(" %4.2f ",y[i]); printf("\n\n"); }
//=================================================================
================
void Fast_Circular_Convolve(float *x, int N, float *h, float *y)
{
int i,j,k,q,s;
float X[max][2],H[max][2],Y[max][2],t[max][2],p[max][2]; for(k=0;
k<N; k++) // Initialization
{
X[k][0] = 0; X[k][1] = 0;
H[k][0] = 0; H[k][1] = 0;
Y[k][0] = 0; Y[k][1] = 0;
}
for(i=0;i<N;i++) // Copy x[n] to t[n][0]
{
t[i][0] = x[i]; t[i][1] = 0;
}
// Find X[k]
if ( N == 4 )
DITFFT_4_Point(N,t,X); else if
( N == 8 )
DITFFT_8_Point(N,t,X);
for(i=0;i<N;i++) // Copy h[n] to t[n][0]
{
t[i][0] = h[i]; t[i][1] = 0;
}
// Find H[k]
if ( N == 4 )
DITFFT_4_Point(N,t,H);
```

```
else if ( N == 8 )
DITFFT_8_Point(N,t,H);
// Find Y[k] for(k=0;
k<N; k++)
{
float a,b,c,d; a = X[k][0];
b = X[k][1]; c = H[k][0]; d
= H[k][1]; Y[k][0] = (a * c)
- (b * d);
Y[k][1] = (b * c) + (a * d);
}
// Find Y*[k]
for (k=0; k< N; k++)
{
Y[k][1] = Y[k][1] * (-1);
}
// Find FFT{Y*[k]} if
( N == 4 )
DITFFT_4_Point(N,Y,p); else if
( N == 8 )
DITFFT_8_Point(N,Y,p);
// find p[n] = { FFT{Y*[k]} / N }* for(i=0;i<N;i++)
{ p[i][0] = p[i][0]/N;
p[i][1] = (-1) * p[i][1]/N;
}
for(i=0;i<N;i++) // Copy p[][] to y[n]
{
y[i] = p[i][0];
}
}
/*-circular convolve();-*/
//0000000000000000000000000000000000000000000000000000000000000
000000000000000
void DITFFT_8_Point( int N, float x[8][2], float t[8][2])
{
int i,j,k,a,b,c,d;
float e;
float X1[4][2], X2[4][2], G[4][2], H[4][2];
for(i = 0; i < 4; i++)
{
```

	X1[i][0] = x[2*i][0]; X1[i][1] = x[2*i][1]; X2[i][0] = x[2*i+1][0]; X2[i][1] = x[(2*i)+1][1]; } DITFFT_4_Point(4,X1,G);

	DITFFT_4_Point(4,X2,H); //
	X[k] = G[k] + W H[k] e =
	6.283185307179586/N;
	for(k=0; k<4; k++)
	{
	t[k][0] = G[k][0] + (H[k][0] * cos(e*k) + H[k][1] * sin(e*k)); t[k][1] =
	G[k][1] + (H[k][1] * cos(e*k) - H[k][0] * sin(e*k));
	}
	for(k=0; k<4; k++)
	{
	d = k + 4;
	t[d][0] = G[k][0] + (H[k][0] * cos(e*d) + H[k][1] * sin(e*d)); t[d][1] =
	G[k][1] + (H[k][1] * cos(e*d) - H[k][0] * sin(e*d));
	}
	}
	/***
	**************/
	void DITFFT_4_Point(int N, float x[4][2], float t[4][2])
	{
	int i,j,k,n,a,b,c,d;
	float e; float G[4][2],
	H[4][2]; for(n=0; n<N;
	n++)
	{ t[n][0] = 0; t[n][1] = 0;
	G[n][0] = 0; G[n][1] = 0;
	H[n][0] = 0; H[n][1] = 0;
	}
	// Stage-1
	G[0][0] = x[0][0] + x[2][0]; G[0][1] = x[0][1] + x[2][1];
	G[1][0] = x[0][0] - x[2][0]; G[1][1] = x[0][1] - x[2][1];
	H[0][0] = x[1][0] + x[3][0]; H[0][1] = x[1][1] + x[3][1];
	H[1][0] = x[1][0] - x[3][0]; H[1][1] = x[1][1] - x[3][1];
	// Stage-2 e =
	6.283185307179586/N; //
	X[k] = G[k] + WNnk H[k]
	k=0; t[0][0] = G[0][0] + (H[0][0] * cos(e*k) + H[0][1] * sin(e*k));
	t[0][1] = G[0][1] + (H[0][1] * cos(e*k) - H[0][0] * sin(e*k)); k=1;
	t[1][0] = G[1][0] + (H[1][0] * cos(e*k) + H[1][1] * sin(e*k)); t[1][1]
	= G[1][1] + (H[1][1] * cos(e*k) - H[1][0] * sin(e*k)); k=2; t[2][0] =

	G[0][0] + (H[0][0] * cos(e*k) + H[0][1] * sin(e*k)); t[2][1] = G[0][1] + (H[0][1] * cos(e*k) - H[0][0] * sin(e*k)); k=3; t[3][0] = G[1][0] + (H[1][0] * cos(e*k) + H[1][1] * sin(e*k)); t[3][1] = G[1][1] + (H[1][1] * cos(e*k) - H[1][0] * sin(e*k)); } /**

	**************/
SOLVING	Overlap-Add Method

overlap - Add method

que.

$$x(n) = \{3, 4, 1, 2\}$$

$$h(n) = \{1, 2, 1\} \qquad L_h = 3$$

Solution

Block length $\Rightarrow N = 2L_h = 2 \times 3$
$= 6$

length of subset from $x(n)$

$$L = N - L_h + 1$$
$$= 6 - 3 + 1$$
$$= 4$$

No. of zeros at end of subset $= L_h - 1$
$= 3 - 1$
$= 2$

$$x_1(n) = \{3, 4, 1, 2, 0, 0\}$$
$$h(n) = \{1, 2, 1, 0, 0, 0\}$$

$$y_1(n) = \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 1 & 0 & 0 & 0 \end{array} \\ \begin{bmatrix} 3 & 0 & 0 & 2 & 1 & 4 \\ 4 & 3 & 0 & 0 & 2 & 1 \\ 1 & 4 & 3 & 0 & 0 & 2 \\ 2 & 1 & 4 & 3 & 0 & 0 \\ 0 & 2 & 1 & 4 & 3 & 0 \\ 0 & 0 & 2 & 1 & 4 & 3 \end{bmatrix} \end{array} \begin{bmatrix} 1 \\ 2 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$y_1(n) = \begin{bmatrix} 3 \\ 4+6 \\ 1+8+3 \\ 2+2+4 \\ 4+1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 10 \\ 12 \\ 8 \\ 5 \\ 2 \end{bmatrix}$$

$$y(n) = \{3, 10, 12, 8, 5, 2\}$$

Output

Overlap-Add Method

```
Enter the length of x[n] = : 4

Enter the values of x[n] : 3 4 1 2


Enter the length of h[n] M = : 3

Enter the values of h[n] : 1 2 1


x[n] =  3.00  4.00  1.00  2.00

h[n] =  1.00  2.00  1.00

length of decomposed input Signal : L = 6

length of decomposed output Signal : N = 8



x1[n] =  3.00  4.00  1.00  2.00  0.00  0.00  0.00  0.00

y1[n] =  3.00  10.00  12.00  8.00  5.00  2.00  -0.00  0.00

Linear Convolution Output using Over Add Method
y[n] =  3.00  10.00  12.00  8.00  5.00  2.00
```

Algorithm

Overlap save method

1. Given two input sequences x(n) and h(n), with lengths N and M respectively, and a block length L.
2. Pad the input sequence x(n) with zeros to length L + M - 1, if necessary, so that it can be divided into L-length blocks with no overlap.
3. Divide the filter h(n) into blocks of length L, and append zeros to the end of each block to make it of length L + M - 1.
4. Perform circular convolution of the first L + M - 1 samples of the input sequence x(n) with the first block of the filter h(n), to obtain the first L + M - 1 samples of the output sequence y(n).
5. Shift the input sequence x(n) by L samples to the right, and repeat step 4 with the next L + M - 1 samples of x(n) and the next block of h(n).
6. Discard the first M - 1 samples of each block of y(n) to avoid overlap between adjacent blocks.
7. Concatenate the remaining samples of each block of y(n) to obtain the final output sequence.

Code	Overlap-Save Method
	
	#include<stdio.h>
	#include<math.h> #define
	max 64
	void Fast_Circular_Convolve(float *x, int N, float *h, float *y);
	void DITFFT_8_Point(int N, float x[8][2], float t[8][2]); void
	DITFFT_4_Point(int N, float x[4][2], float t[4][2]); void main()
	{
	int i,j,k,SigLen,M,N,L;
	float x[max],p[max],t[max],y[max]; float
	h[max];
	for(i=0;i<max;i++) // Initialization
	{
	x[i]=0; h[i]=0; y[i]=0; p[i]=0; }
	//------------------------- INPUT ----------------------------------printf("\n Enter the length of
	x[n] = : "); scanf("%d",&SigLen);
	printf("\n Enter the values of x[n] : ");
	for(i=0;i<SigLen;i++)
	{
	scanf("%f",&x[i]);
	}
	printf("\n\n Enter the length of h[n] M = : "); scanf("%d",&M);
	printf("\n Enter the values of h[n] : ");
	for(i=0;i<M;i++)
	{
	scanf("%f",&h[i]);
	} printf("\n\n x[n] = ");
	for(i=0;i<SigLen;i++)
	printf(" %4.2f ",x[i]);
	printf("\n\n h[n] = ");
	for(i=0;i<M;i++)
	printf(" %4.2f ",h[i]);
	//------------------------------ OSM
	-- 	
	N = 8; // Assumption
	L = N - M + 1; // Length of decomposed x[n] printf("\n\n length
	of decomposed input Signal : L = %d ", L); printf("\n\n length of
	decomposed output Signal : N = %d ", N); for(i=0; i<L; i++)

	p[i+M-1] = x[i]; // Splitting of x[n] : first signal with initial zeros

	j = 0; int
	d = 0;
	for (k=0; k<=SigLen; k+= L)
	{
	Fast_Circular_Convolve(p, N, h, t); // Lc By CC by FFT for(i=0;
	i<L; i++)
	{
	y[k+i] = t[i+M-1]; // Overap Save
	} j++;
	printf("\n\n\n x%d[n] = ", j); // Decomposed x[n]
	for(i=0;i< N; i++) printf(" %4.2f ",p[i]);
	printf("\n\n y%d[n] = ",j); // Decomosed y[n]
	for(i=0;i< N;i++) printf(" %4.2f ",t[i]); d = k+ L-
	(M-1);
	for(i=0; i<N; i++) p[i] = x[d+i];
	// Splitting of x[n] with previous M-1 value
	}
	//---
	------ 	
	printf("\n\n Linear Convolution Output using Over Save Method ");
	printf("\n y[n] = "); // Result of LC for(i=0;i<(SigLen+M-1);i++)
	printf(" %4.2f ",y[i]); printf("\n\n"); }
	//==
	===================
	void Fast_Circular_Convolve(float *x, int N, float *h, float *y)
	{
	int i,j,k,q,s;
	float X[max][2],H[max][2],Y[max][2],t[max][2],p[max][2]; for(k=0;
	k<N; k++) // Initialization
	{
	X[k][0] = 0; X[k][1] = 0;
	H[k][0] = 0; H[k][1] = 0;
	Y[k][0] = 0; Y[k][1] = 0;
	}
	for(i=0;i<N;i++) // Copy x[n] to t[n][0]
	{
	t[i][0] = x[i]; t[i][1] = 0;
	}
	// Find X[k] if
	(N == 4)

```
DITFFT_4_Point(N,t,X); else if
( N == 8 )
DITFFT_8_Point(N,t,X);
for(i=0;i<N;i++) // Copy h[n] to t[n][0]
{
t[i][0] = h[i]; t[i][1] = 0;
}
// Find H[k]
if ( N == 4 )
DITFFT_4_Point(N,t,H); else if
( N == 8 )
DITFFT_8_Point(N,t,H);
// Find Y[k] for(k=0;
k<N; k++)
{
float a,b,c,d; a = X[k][0];
b = X[k][1]; c = H[k][0]; d
= H[k][1]; Y[k][0] = (a * c)
- (b * d);
Y[k][1] = (b * c) + (a * d);
}
// Find Y*[k]
for (k=0; k< N; k++)
{
Y[k][1] = Y[k][1] * (-1);
}
// Find FFT{Y*[k]} if
( N == 4 )
DITFFT_4_Point(N,Y,p); else if
( N == 8 )
DITFFT_8_Point(N,Y,p);
// find p[n] = { FFT{Y*[k]} / N }* for(i=0;i<N;i++)
{ p[i][0] = p[i][0]/N;
p[i][1] = (-1) * p[i][1]/N;
}
for(i=0;i<N;i++) // Copy p[][] to y[n]
{
y[i] = p[i][0];
```

```
}
}
/*-circular convolve();-*/
//0000000000000000000000000000000000000000000000000000000000000000
000000000000000000
void DITFFT_8_Point( int N, float x[8][2], float t[8][2])
```

```
{
int i,j,k,a,b,c,d;
float e;
float X1[4][2], X2[4][2], G[4][2], H[4][2];
for(i = 0; i < 4; i++)
{
X1[i][0] = x[2*i][0]; X1[i][1] = x[2*i][1];
X2[i][0] = x[2*i+1][0]; X2[i][1] = x[(2*i)+1][1];
}
DITFFT_4_Point(4,X1,G);
DITFFT_4_Point(4,X2,H); //
X[k] = G[k] + W H[k] e =
6.283185307179586/N;
for(k=0; k<4; k++)
{
t[k][0] = G[k][0] + ( H[k][0] * cos(e*k) + H[k][1] * sin(e*k) ); t[k][1] =
G[k][1] + ( H[k][1] * cos(e*k) - H[k][0] * sin(e*k) );
}
for(k=0; k<4; k++)
{
d = k + 4;
t[d][0] = G[k][0] + ( H[k][0] * cos(e*d) + H[k][1] * sin(e*d) ); t[d][1] =
G[k][1] + ( H[k][1] * cos(e*d) - H[k][0] * sin(e*d) );
}
}
/*************************************************************
*****************/
void DITFFT_4_Point( int N, float x[4][2], float t[4][2])
{
int i,j,k,n,a,b,c,d;
float e; float G[4][2],
H[4][2]; for(n=0; n<N;
n++)
{ t[n][0] = 0; t[n][1] = 0;
G[n][0] = 0; G[n][1] = 0;
H[n][0] = 0; H[n][1] = 0;
}
// Stage-1
G[0][0] = x[0][0] + x[2][0]; G[0][1] = x[0][1] + x[2][1];
G[1][0] = x[0][0] - x[2][0]; G[1][1] = x[0][1] - x[2][1];
```

	H[0][0] = x[1][0] + x[3][0]; H[0][1] = x[1][1] + x[3][1];
	H[1][0] = x[1][0] - x[3][0]; H[1][1] = x[1][1] - x[3][1];
	// Stage-2 e =
	6.283185307179586/N; //
	X[k] = G[k] + WNnk H[k]

```
k=0; t[0][0] = G[0][0] + ( H[0][0] * cos(e*k) + H[0][1] * sin(e*k) );
t[0][1] = G[0][1] + ( H[0][1] * cos(e*k) - H[0][0] * sin(e*k) ); k=1;
t[1][0] = G[1][0] + ( H[1][0] * cos(e*k) + H[1][1] * sin(e*k) ); t[1][1]
= G[1][1] + ( H[1][1] * cos(e*k) - H[1][0] * sin(e*k) ); k=2; t[2][0] =
G[0][0] + ( H[0][0] * cos(e*k) + H[0][1] * sin(e*k) ); t[2][1] = G[0][1]
+ ( H[0][1] * cos(e*k) - H[0][0] * sin(e*k) ); k=3; t[3][0] = G[1][0] + (
H[1][0] * cos(e*k) + H[1][1] * sin(e*k) ); t[3][1] = G[1][1] + ( H[1][1]
* cos(e*k) - H[1][0] * sin(e*k) ); }
/******************************************************************
******************/
```

SOLVING	Overlap-Save Method

que. Overlap-save method.

$$x[n] = \{2, 1, -1, 2, 1, 1\}$$

$$h[n] = \{1, 2, 3\}$$

solution

$$L_h - 1 = 3 - 1 = 2$$

$$x(n) = \{0, 0, 2, 1, -1, 2, 1, 1\}$$

$$\text{Block length} = 2L_h = 2 \times 3 = 6$$

$$\text{overlap} = L_{h-1} = 3 - 1 = 2$$

$$x_1(n) = \{0, 0, 2, 1, \underline{-1, 2}\}$$

$$x_2(n) = \{-1, 2, 1, 1, 0, 0\}$$

$$h(n) = \{1, 2, 3, 0, 0, 0\}$$

$$\begin{array}{cc} & \begin{array}{cccccc} 1 & 2 & 3 & 0 & 0 & 0 \end{array} \\ y_1(n) = & \begin{bmatrix} 0 & 2 & -1 & 1 & 2 & 0 \\ 0 & 0 & 2 & -1 & 1 & 2 \\ 2 & 0 & 0 & 2 & -1 & 1 \\ 1 & 2 & 0 & 0 & 2 & -1 \\ -1 & 1 & 2 & 0 & 0 & 2 \\ 2 & -1 & 1 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 0 \\ 0 \\ 0 \end{bmatrix} \end{array}$$

$$y_1(n) = \begin{bmatrix} \cancel{1} \\ \cancel{6} \\ 2 \\ 5 \\ 7 \\ 3 \end{bmatrix}$$

$$
\begin{array}{c}
\begin{array}{cccccc} 1 & 2 & 3 & 0 & 0 & 0 \end{array} \\
y_2(n) = \begin{bmatrix} -1 & 0 & 0 & 1 & 1 & 2 \\ 2 & -1 & 0 & 0 & 1 & 1 \\ 1 & 2 & -1 & 0 & 0 & 1 \\ 1 & 1 & 2 & -1 & 0 & 0 \\ 0 & 1 & 1 & 2 & -1 & 0 \\ 0 & 0 & 1 & 1 & 2 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 0 \\ 0 \\ 0 \end{bmatrix}
\end{array}
$$

$$
= \begin{bmatrix} -1 \\ 2-2 \\ 1+4-3 \\ 1+2+6 \\ 2+3 \\ 3 \end{bmatrix} = \begin{bmatrix} \cancel{-1} \\ \cancel{0} \\ 2 \\ 9 \\ 5 \\ 3 \end{bmatrix}
$$

$$
y(n) = \{2, 5, 7, 3, 2, 9, 5, 3\}
$$

Output

Overlap-Save Method

```
Enter the length of x[n] = : 6

Enter the values of x[n] : 2 1 -1 2 1 1


Enter the length of h[n] M = : 3

Enter the values of h[n] : 1 2 3


x[n] =  2.00  1.00  -1.00  2.00  1.00  1.00

h[n] =  1.00  2.00  3.00

length of decomposed input Signal : L = 6

length of decomposed output Signal : N = 8


x1[n] =  0.00  0.00  2.00  1.00  -1.00  2.00  1.00  1.00

y1[n] =  5.00  3.00  2.00  5.00  7.00  3.00  2.00  9.00


x2[n] =  1.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00

y2[n] =  1.00  3.00  5.00  3.00  -0.00  -0.00  -0.00  0.00

Linear Convolution Output using Over Save Method
y[n] =  2.00  5.00  7.00  3.00  2.00  9.00  5.00  3.00
```

Conclusion	1. Two methods were implemented to obtain linear convolution via circular convolution using Fourier Transform (FFT) in C-language: the Overlap-Add method and the Overlap-Save method. 2. Both methods involve breaking down large sequences into smaller segments and using circular convolution to compute the final linear convolution result. 3. The choice between the two methods depends on the specific requirements of the application. 4. The use of FFT and circular convolution provides a fast and e cient way to perform linear convolution in signal processing applications.
Reference	1. S. K. Mitra, "Digital Signal Processing: A Computer-Based Approach," Tata McGraw Hill Education, 4th ed., 2011. 2. B. Venkataramani and M. Bhaskar, "Digital Signal Processing: Principles and Applications," Tata McGraw Hill Education, 2nd ed., 2015. 3. A. V. Oppenheim and R. W. Schafer, "Discrete-Time Signal Processing," Prentice Hall, 3rd ed., 2009.