Omkar Mesur
1BM19CS330

Q1] 1) Because convolution is defined in terms of neighbouring elements, boundary conditions naturally exist for output elements close to ends of the array.

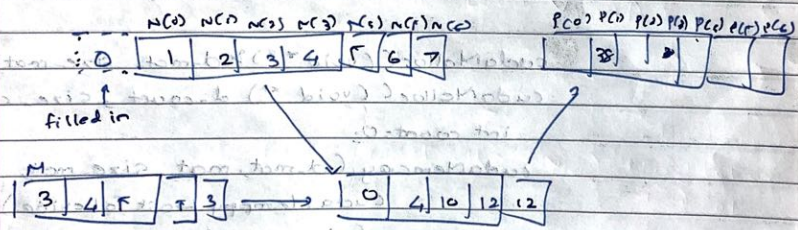
2) Sometimes, there are not enough elements to calculate an output element.

~~3) According to the definition~~

3) A typical approach to handling such a boundary condition is to define a default value to the missing N elements

4) For most applications the default value is 0. These missing elements are typically referred to as ghost elements.

5) These ghost elements can have significant impact on the complexity or efficiency of filing.



Q2]

```
#include <stdio.h>
#include <stdlib.h>
const int N;
__global__ void borderprimes(int * mat, int *count);
__device__ int isPrime(int n);
int main()
{
    printf("Enter value of N\n");
    scanf("%d", &N);
    int mat[N][N];
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<N; j++)
            scanf("%d", &mat[i][j]);
    }

    int *d_mat, *d_count;
    int size_mat = N*N*sizeof(int);
    int size_count = sizeof(count);

    cudaMalloc((void**) &d_mat, size_mat);
    cudaMalloc((void **) &d_count, size_count);
    int count=0;
    cudaMemcpy(d_mat, mat, size_mat,
                cudaMemcpyHostToDevice);
    cudaMemcpy(d_count, &count, size_count,
                cudaMemcpyHostToDevice);

    dim3 numberofBlocks(N/2, N/2, 1);
    dim3 numberofThreads(1, 1, 1);

    borderprimes<<<numberofBlocks, numberofThreads
                >>>(d_mat, d_count);
```

```
cudaMemcpy(&count, d_count, size, count,
        cudaMemcpyDeviceToHost);

printf("Number of primes: %d\n", count);

cudaFree(d_mat);
cudaFree(d_count);

return 0;
}

__global__ void border_primes(int *mat, int *count)
{
    int col = threadIdx.x + blockIdx.x * blockDim.x;
    int row = threadIdx.y + blockDim.y * blockIdx.y;

    if (row == 0 || row == N-1) // First row or last row
    {
        if(isPrime(mat[row*N+col]))
        {
            atomicAdd(count, 1);
        }
    }
    else if (col == 0 || col == N-1) // First col or last col
    {
        if (isPrime(mat[row*N+col]))
        {
            atomicAdd(count, 1);
        }
    }
}
```

```
int isPrime(int a)
{
    if(a==1) return 0;
    int flag=0;
    for(int i=2; i<=a/2; i++)
    {
        if(a%i==0)
        {
            flag=1;
            break;
        }
    }
    return (!flag);
}
```