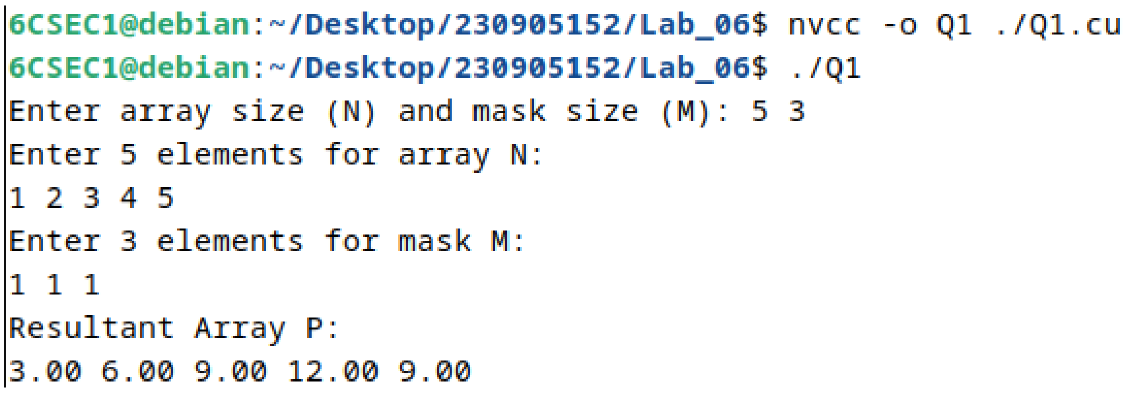
Lab 6 – Programs on arrays in CUDA cont.

Q1.cu

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <stdlib.h>

__global__ void convolution1D(float *N, float *M, float *P, int width, int
mask_width) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < width) {
        float Pvalue = 0;
        int start_point = i - (mask_width / 2);
        for (int j = 0; j < mask_width; j++) {
            int cur_idx = start_point + j;
            if (cur_idx >= 0 && cur_idx < width) {
                Pvalue += N[cur_idx] * M[j];
            }
        }
        P[i] = Pvalue;
    }
}

int main() {
    int n, m_width;
    printf("Enter array size (N) and mask size (M): ");
    scanf("%d %d", &n, &m_width);

    size_t size_n = n * sizeof(float);
    size_t size_m = m_width * sizeof(float);

    float *h_N = (float*)malloc(size_n);
    float *h_M = (float*)malloc(size_m);
    float *h_P = (float*)malloc(size_n);

    printf("Enter %d elements for array N:\n", n);
    for (int i = 0; i < n; i++) scanf("%f", &h_N[i]);
    printf("Enter %d elements for mask M:\n", m_width);
    for (int i = 0; i < m_width; i++) scanf("%f", &h_M[i]);

    float *d_N, *d_M, *d_P;
    cudaMalloc(&d_N, size_n);
    cudaMalloc(&d_M, size_m);
    cudaMalloc(&d_P, size_n);

    cudaMemcpy(d_N, h_N, size_n, cudaMemcpyHostToDevice);
    cudaMemcpy(d_M, h_M, size_m, cudaMemcpyHostToDevice);

    int threads = 256;
    int blocks = (n + threads - 1) / threads;

    convolution1D<<<blocks, threads>>>(d_N, d_M, d_P, n, m_width);

    cudaMemcpy(h_P, d_P, size_n, cudaMemcpyDeviceToHost);

    printf("Resultant Array P:\n");
    for (int i = 0; i < n; i++) printf("%.2f ", h_P[i]);
    printf("\n");

    cudaFree(d_N);
    cudaFree(d_M);
    cudaFree(d_P);
    free(h_N);
    free(h_M);
    free(h_P);

    return 0;
}
```

Output:



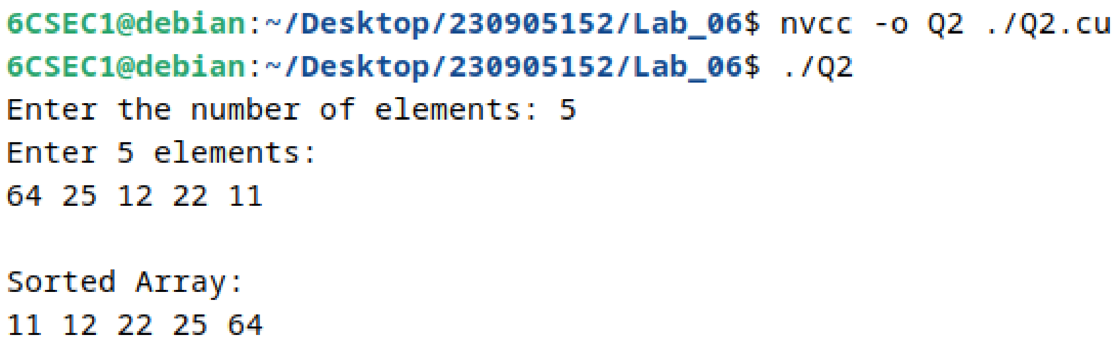
```
6CSEC1@debian:~/Desktop/230905152/Lab_06$ nvcc -o Q1 ./Q1.cu
6CSEC1@debian:~/Desktop/230905152/Lab_06$ ./Q1
Enter array size (N) and mask size (M): 5 3
Enter 5 elements for array N:
1 2 3 4 5
Enter 3 elements for mask M:
1 1 1
Resultant Array P:
3.00 6.00 9.00 12.00 9.00
```

Q2.cu

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
```

```
#include <stdio.h>
#include <stdlib.h>

__global__ void selectionSortParallel(int *input, int *output, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) {
        int count = 0;
        int key = input[i];

        for (int j = 0; j < n; j++) {
            if (input[j] < key) {
                count++;
            }
            else if (input[j] == key && j < i) {
                count++;
            }
        }
        output[count] = key;
    }
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    if (scanf("%d", &n) != 1) return 1;

    size_t size = n * sizeof(int);
    int *h_in = (int*)malloc(size);
    int *h_out = (int*)malloc(size);

    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &h_in[i]);
    }

    int *d_in, *d_out;
    cudaMalloc((void**)&d_in, size);
    cudaMalloc((void**)&d_out, size);

    cudaMemcpy(d_in, h_in, size, cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocksPerGrid = (n + threadsPerBlock - 1) / threadsPerBlock;
    selectionSortParallel<<<blocksPerGrid, threadsPerBlock>>>(d_in, d_out, n);
    cudaDeviceSynchronize();
    cudaMemcpy(h_out, d_out, size, cudaMemcpyDeviceToHost);

    printf("\nSorted Array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", h_out[i]);
    }
    printf("\n");

    cudaFree(d_in);
    cudaFree(d_out);
    free(h_in);
    free(h_out);

    return 0;
}
```

Output:

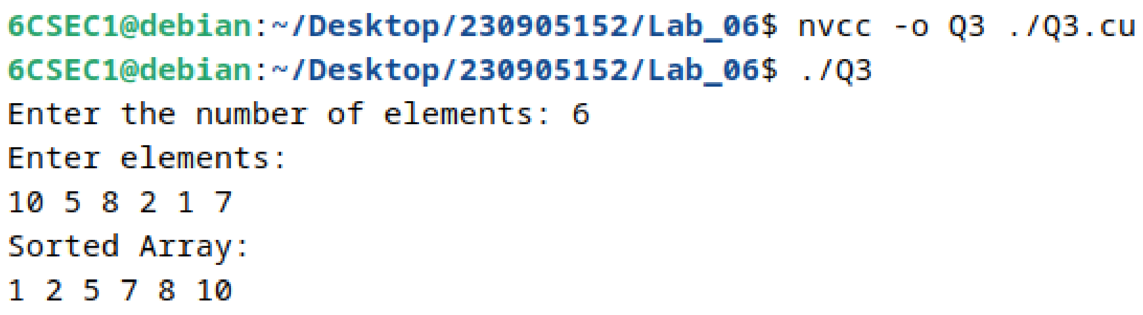


```
6CSEC1@debian:~/Desktop/230905152/Lab_06$ nvcc -o Q2 ./Q2.cu
6CSEC1@debian:~/Desktop/230905152/Lab_06$ ./Q2
Enter the number of elements: 5
Enter 5 elements:
64 25 12 22 11

Sorted Array:
11 12 22 25 64
```

Q3.cu

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <stdlib.h>

__global__ void oddEvenSort(int *a, int n, int phase) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (phase == 0) {
        int idx = 2 * i;
        if (idx + 1 < n) {
            if (a[idx] > a[idx + 1]) {
                int temp = a[idx];
                a[idx] = a[idx + 1];
                a[idx + 1] = temp;
            }
        }
    } else {
        int idx = 2 * i + 1;
        if (idx + 1 < n) {
            if (a[idx] > a[idx + 1]) {
                int temp = a[idx];
                a[idx] = a[idx + 1];
                a[idx + 1] = temp;
            }
        }
    }
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int size = n * sizeof(int);
    int *h_a = (int*)malloc(size);

    printf("Enter elements:\n");
    for (int i = 0; i < n; i++) scanf("%d", &h_a[i]);

    int *d_a;
    cudaMalloc((void**)&d_a, size);
    cudaMemcpy(d_a, h_a, size, cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocksPerGrid = (n / 2 + threadsPerBlock - 1) / threadsPerBlock;

    for (int i = 0; i < n; i++) {
        oddEvenSort<<<blocksPerGrid, threadsPerBlock>>>(d_a, n, i % 2);
        cudaDeviceSynchronize();
    }

    cudaMemcpy(h_a, d_a, size, cudaMemcpyDeviceToHost);

    printf("Sorted Array:\n");
    for (int i = 0; i < n; i++) printf("%d ", h_a[i]);
    printf("\n");

    cudaFree(d_a);
    free(h_a);

    return 0;
}
```

Output.

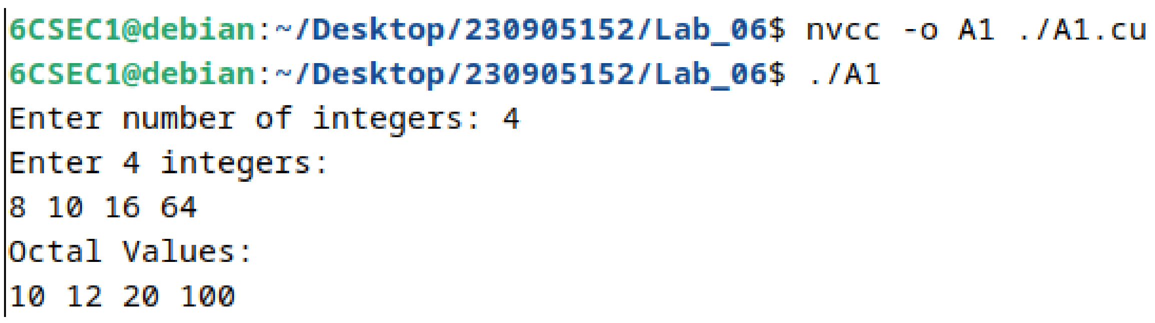


```
6CSEC1@debian:~/Desktop/230905152/Lab_06$ nvcc -o Q3 ./Q3.cu
6CSEC1@debian:~/Desktop/230905152/Lab_06$ ./Q3
Enter the number of elements: 6
Enter elements:
10 5 8 2 1 7
Sorted Array:
1 2 5 7 8 10
```

A1.cu

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <stdlib.h>

__global__ void convertToOctal(int *input, int *output, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) {
        int temp = input[i];
        int octal = 0, place = 1;
        while (temp != 0) {
            octal += (temp % 8) * place;
            temp /= 8;
            place *= 10;
        }
        output[i] = octal;
    }
}

int main() {
    int n;
    printf("Enter number of integers: ");
    scanf("%d", &n);
    size_t size = n * sizeof(int);
    int *h_in = (int*)malloc(size);
    int *h_out = (int*)malloc(size);

    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++) scanf("%d", &h_in[i]);

    int *d_in, *d_out;
    cudaMalloc(&d_in, size);
    cudaMalloc(&d_out, size);
    cudaMemcpy(d_in, h_in, size, cudaMemcpyHostToDevice);

    int threads = 256;
    int blocks = (n + threads - 1) / threads;

    convertToOctal<<<blocks, threads>>>(d_in, d_out, n);

    cudaMemcpy(h_out, d_out, size, cudaMemcpyDeviceToHost);

    printf("Octal Values:\n");
    for (int i = 0; i < n; i++) printf("%d ", h_out[i]);
    printf("\n");

    cudaFree(d_in);
    cudaFree(d_out);
    free(h_in);
    free(h_out);

    return 0;
}
```

Output.



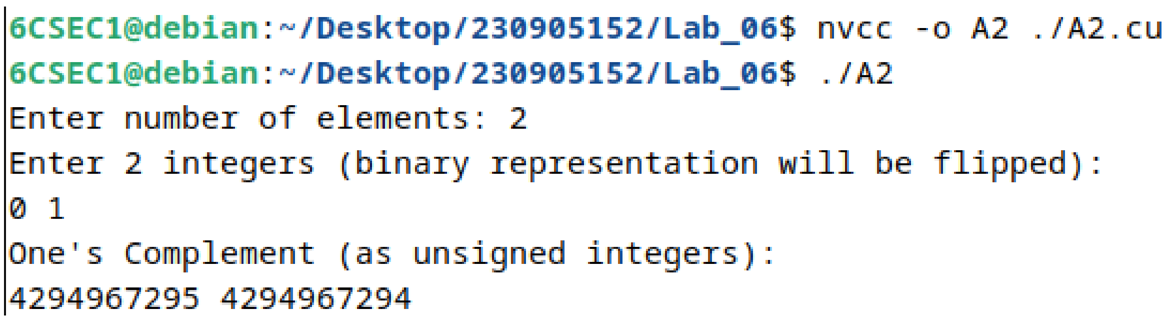
```
6CSEC1@debian:~/Desktop/230905152/Lab_06$ nvcc -o A1 ./A1.cu
6CSEC1@debian:~/Desktop/230905152/Lab_06$ ./A1
Enter number of integers: 4
Enter 4 integers:
8 10 16 64
Octal Values:
10 12 20 100
```

A2.cu

```
#include "cuda_runtime.h"
```

```
#include "device_launch_parameters.h"
#include <stdio.h>
#include <stdlib.h>

__global__ void onesComplement(unsigned int *input, unsigned int *output, int n)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) {
        output[i] = ~input[i];
    }
}

int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    size_t size = n * sizeof(unsigned int);
    unsigned int *h_in = (unsigned int*)malloc(size);
    unsigned int *h_out = (unsigned int*)malloc(size);

    printf("Enter %d integers (binary representation will be flipped):\n", n);
    for (int i = 0; i < n; i++) scanf("%u", &h_in[i]);

    unsigned int *d_in, *d_out;
    cudaMalloc(&d_in, size);
    cudaMalloc(&d_out, size);
    cudaMemcpy(d_in, h_in, size, cudaMemcpyHostToDevice);

    int threads = 256;
    int blocks = (n + threads - 1) / threads;
    onesComplement<<<blocks, threads>>>(d_in, d_out, n);

    cudaMemcpy(h_out, d_out, size, cudaMemcpyDeviceToHost);

    printf("One's Complement (as unsigned integers):\n");
    for (int i = 0; i < n; i++) printf("%u ", h_out[i]);
    printf("\n");

    cudaFree(d_in);
    cudaFree(d_out);
    free(h_in);
    free(h_out);
    return 0;
}
```

Output.



```
6CSEC1@debian:~/Desktop/230905152/Lab_06$ nvcc -o A2 ./A2.cu
6CSEC1@debian:~/Desktop/230905152/Lab_06$ ./A2
Enter number of elements: 2
Enter 2 integers (binary representation will be flipped):
0 1
One's Complement (as unsigned integers):
4294967295 4294967294
```