190905104
Parth Shukla

1)

```
#include <cuda.h>
#include <stdlib.h>
#include <stdio.h>
// block size as N
__global__ void vecAddKernel_1a(float *A, float *B, float *C)
{
   int idx = threadIdx.x + blockIdx.x * blockDim.x;
   C[idx] = A[idx] + B[idx];
}
// N threads within a block
__global__ void vecAddKernel_1b(float *A, float *B, float *C)
{
   int idx = threadIdx.x + blockIdx.x * blockDim.x;
   C[idx] = A[idx] + B[idx];
}
// Keep the number of threads per block as 256 (constant) and vary the number of blocks to handle N elements.
__global__ void vecAddKernel_1c(float *A, float *B, float *C,
                                int n)
{
   int idx = threadIdx.x + blockIdx.x * blockDim.x;
   if (idx < n)
   {
      C[idx] = A[idx] + B[idx];
   }
}
void vecAdd(float *A, float *B, float *C, int n)
{
   int size = n * sizeof(float);
   float *d_A;
   float *d_B;
   float *d_C;
   cudaMalloc((void **)&d_A, size);
   cudaMalloc((void **)&d_B, size);
   cudaMalloc((void **)&d_C, size);
   cudaMemcpy(d_A, A, size, cudaMemcpyHostToDevice);
   cudaMemcpy(d_B, B, size, cudaMemcpyHostToDevice);
   printf("A: ");
   for (int i = 0; i < n; i++)
   {
      printf("%f, ", A[i]);
   }
   printf("\n");
   printf("B: ");
   for (int i = 0; i < n; i++)
   {
      printf("%f, ", B[i]);
```

```
    }
    printf("\n\n");
    vecAddKernel_1a<<<n, 1>>>(d_A, d_B, d_C);
    cudaMemcpy(C, d_C, size, cudaMemcpyDeviceToHost);
    printf("A+B (from 1a kernel): ");
    for (int i = 0; i < n; i++)
    {
        printf("%f, ", C[i]);
    }
    printf("\n");
    vecAddKernel_1b<<<1, n>>>(d_A, d_B, d_C);
    cudaMemcpy(C, d_C, size, cudaMemcpyDeviceToHost);
    printf("A+B (from 1b kernel): ");
    for (int i = 0; i < n; i++)
    {
        printf("%f, ", C[i]);
    }
    printf("\n");
    vecAddKernel_1c<<<ceil(n / 256.0), n>>>(d_A, d_B,
                                        d_C, n);
    cudaMemcpy(C, d_C, size, cudaMemcpyDeviceToHost);
    printf("A+B (from 1c kernel): ");
    for (int i = 0; i < n; i++)
    {
        printf("%f, ", C[i]);
    }
    printf("\n");
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
}
int main()
{
    float *h_A, *h_B, *h_C;
    int n = 5;
    int size = n * sizeof(float);
    h_A = (float *)malloc(size);
    h_B = (float *)malloc(size);
    h_C = (float *)malloc(size);
    for (int i = 0; i < n; i++)
    {
        h_A[i] = (i + 1) * 10;
        h_B[i] = i + 1;
    }
    vecAdd(h_A, h_B, h_C, n);
    return 0;
}
```

```
student@dblab-hp-280-10:~/190905104_ParthShukla_PCAP/week5$ scp q1.cu D-19090510
4@172.16.57.152:/home/D-190905104
D-190905104@172.16.57.152's password:
q1.cu                                         100% 2460    72.7KB/s   00:00
student@dblab-hp-280-10:~/190905104_ParthShukla_PCAP/week5$ scp q2.cu D-19090510
4@172.16.57.152:/home/D-190905104
D-190905104@172.16.57.152's password:
q2.cu                                         100% 1494    32.6KB/s   00:00
student@dblab-hp-280-10:~/190905104_ParthShukla_PCAP/week5$ scp q3.cu D-19090510
4@172.16.57.152:/home/D-190905104
D-190905104@172.16.57.152's password:
q3.cu                                         100% 1602    47.8KB/s   00:00
student@dblab-hp-280-10:~/190905104_ParthShukla_PCAP/week5$
```

```
D-190905104@lplab-ProLiant-DL380-G6:~/week5$ nvcc q1.cu
D-190905104@lplab-ProLiant-DL380-G6:~/week5$ ./a.out
A: 10.000000, 20.000000, 30.000000, 40.000000, 50.000000,
B: 1.000000, 2.000000, 3.000000, 4.000000, 5.000000,

A+B (from 1a kernel): 11.000000, 22.000000, 33.000000, 44.000000, 55.000000,
A+B (from 1b kernel): 11.000000, 22.000000, 33.000000, 44.000000, 55.000000,
A+B (from 1c kernel): 11.000000, 22.000000, 33.000000, 44.000000, 55.000000,
D-190905104@lplab-ProLiant-DL380-G6:~/week5$
```

2)

```
#include <cuda.h>
#include <stdlib.h>
#include <stdio.h>
__global__ void
selectionsortkernel(float *A, float *B, int n)
{
   int idx = threadIdx.x + blockIdx.x * blockDim.x;
   float key = A[idx];
   int pos = 0;

   for (int i = 0; i < n; i++)
   {
      if (A[i] < key || (A[i] == key && i < idx))
      {
         pos++;
      }
   }
   B[pos] = key;
}

void selsort(float *A, float *B, int n)
{
   int size = n * sizeof(float);
   float *d_unsorted_arr;
   float *d_sorted_arr;
```

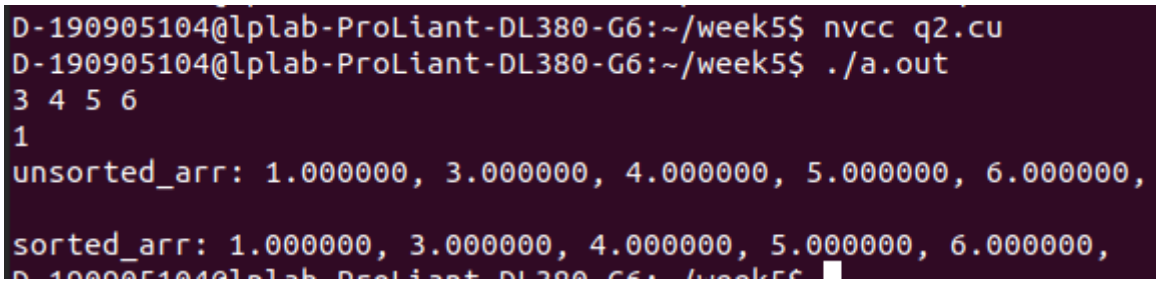
```
    cudaMalloc((void **)&d_unsorted_arr, size);
    cudaMalloc((void **)&d_sorted_arr, size);

    cudaMemcpy(d_unsorted_arr, A, size, cudaMemcpyHostToDevice);

    selectionsortkernel<<<1, n>>>(d_unsorted_arr, d_sorted_arr, n);
    cudaMemcpy(B, d_sorted_arr, size, cudaMemcpyDeviceToHost);

    cudaFree(d_unsorted_arr);
    cudaFree(d_sorted_arr);
}

int main()
{
    float *h_unsorted_arr, *h_sorted_arr;
    int n = 5;
    int size = n * sizeof(float);
    h_unsorted_arr = (float *)malloc(size);
    h_sorted_arr = (float *)malloc(size);
    for (int i = 0; i < 5; i++)
    {
        scanf("%f", &h_unsorted_arr[i]);
    }
    selsort(h_unsorted_arr, h_sorted_arr, n);
    printf("unsorted_arr: ");
    for (int i = 0; i < n; i++)
    {
        printf("%f, ", h_sorted_arr[i]);
    }
    printf("\n\n");
    printf("sorted_arr: ");
    for (int i = 0; i < n; i++)
    {
        printf("%f, ", h_sorted_arr[i]);
    }
    printf("\n");
    return 0;
}
```



3)

```
#include <cuda.h>
#include <stdlib.h>
#include <stdio.h>
        __global__ void
        oddEven(float *arr, int n)
{
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if (idx % 2 == 1 && idx + 1 < n)
    {
        if (arr[idx] > arr[idx + 1])
        {
            float temp = arr[idx];
            arr[idx] = arr[idx + 1];
            arr[idx + 1] = temp;
        }
    }
}
__global__ void evenOdd(float *arr, int n)
{
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if (idx % 2 == 0 && idx + 1 < n)
    {
        if (arr[idx] > arr[idx + 1])
        {
            float temp = arr[idx];
            arr[idx] = arr[idx + 1];
            arr[idx + 1] = temp;
        }
    }
}
void oddEvenTranspositionSort(float *arr, int n)
{
    int size = n * sizeof(float);
    float *d_arr;
    cudaMalloc((void **)&d_arr, size);
    cudaMemcpy(d_arr, arr, size, cudaMemcpyHostToDevice);
    for (int i = 0; i <= n / 2; i++)
    {
        oddEven<<<1, n>>>(d_arr, n);
        evenOdd<<<1, n>>>(d_arr, n);
    }
    cudaMemcpy(arr, d_arr, size, cudaMemcpyDeviceToHost);
    cudaFree(d_arr);
}
int main()
{
    float *h_arr;
    int n = 5;
    int size = n * sizeof(float);
    h_arr = (float *)malloc(size);
    for (int i = 0; i < 5; i++)
```

```
    {
        // h_arr[i] = rand() % 50;
        scanf("%f", &h_arr[i]);
    }
    printf("unsorted_arr: ");
    for (int i = 0; i < n; i++)
    {
        printf("%f, ", h_arr[i]);
    }
    printf("\n\n");
    oddEvenTranspositionSort(h_arr, n);
    printf("sorted_arr: ");
    for (int i = 0; i < n; i++)
    {
        printf("%f, ", h_arr[i]);
    }
    printf("\n");
    return 0;
}
```

```
D-190905104@lplab-ProLiant-DL380-G6:~/week5$ nvcc q3.cu
D-190905104@lplab-ProLiant-DL380-G6:~/week5$ ./a.out
45 62 32 43 78
unsorted_arr: 45.000000, 62.000000, 32.000000, 43.000000, 78.000000,

sorted_arr: 32.000000, 43.000000, 45.000000, 62.000000, 78.000000,
```