1. Write a program for ODD-EVEN sorting and calculate the execution time.

```
#include<stdio.h>
#include<cuda.h>
#define N 5

#define intswap(A,B) {int temp=A;A=B;B=temp;}

__global__ void sort(int *c,int *count)
{
    int l;
    if(*count%2==0)
          l=*count/2;
    else
         l=(*count/2)+1;
    for(int i=0;i<l;i++)
    {
            if((!(threadIdx.x&1)) &&
(threadIdx.x<(*count-1)))  //even phase(&1 will compare
the least significant bit )
            {
                if(c[threadIdx.x]>c[threadIdx.x+1])
                  intswap(c[threadIdx.x],
c[threadIdx.x+1]);
            }

            __syncthreads();
            if((threadIdx.x&1) && (threadIdx.x<(*count-
1)))     //odd phase
            {
                if(c[threadIdx.x]>c[threadIdx.x+1])
                  intswap(c[threadIdx.x],
c[threadIdx.x+1]);
            }
            __syncthreads();
    }

}



int main()
```

```
{int a[N],b[N],n;
    printf("enter size of array");
    scanf("%d",&n);
    if (n > N) {printf("too large!\n"); return 1;}
    printf("enter the elements of array");
  for(int i=0;i<n;i++)
  {
    scanf("%d",&a[i]);
  }
  printf("ORIGINAL ARRAY : \n");
  for(int i=0;i<n;i++)
          {

          printf("%d ",a[i]);
          }
  int *c,*count;
  cudaMalloc((void**)&c,sizeof(int)*N);
  cudaMalloc((void**)&count,sizeof(int));
  cudaMemcpy(c,&a,sizeof(int)*N,cudaMemcpyHostToDevice);

cudaMemcpy(count,&n,sizeof(int),cudaMemcpyHostToDevice);
  sort<<< 1,n >>>(c,count);
  cudaMemcpy(&b,c,sizeof(int)*N,cudaMemcpyDeviceToHost);
  printf("\nSORTED ARRAY : \n");
  for(int i=0;i<n;i++)
      {
         printf("%d ",b[i]);
      }

  printf("\n");
}
```

2. Write a program for sort the elements using Quick sort and calculate the execution time.

```
#include<bits/stdc++.h>

 using namespace std;
 void compAndSwap(int a[], int i, int j, int dir)
 {
 if (dir==(a[i]>a[j]))
 swap(a[i],a[j]);
 }
 void bitonicMerge(int a[], int low, int cnt, int dir)
 {
 if (cnt>1)
 {
```

```
int k = cnt/2;
for (int i=low; i<low+k; i++)
compAndSwap(a, i, i+k, dir);
bitonicMerge(a, low, k, dir);
bitonicMerge(a, low+k, k, dir);
}
}
void bitonicSort(int a[],int low, int cnt, int dir)
{
if (cnt>1)
{
int k = cnt/2;

bitonicSort(a, low, k, 1);
bitonicSort(a, low+k, k, 0);
bitonicMerge(a,low, cnt, dir);
}
}
void sort(int a[], int N, int up)
{
bitonicSort(a,0, N, up);
}

int main()
{
int a[]= {3, 7, 4, 8, 6, 2, 1, 5};
int N = sizeof(a)/sizeof(a[0]);
int up = 1;
sort(a, N, up);
printf("Sorted array: \n");
for (int i=0; i<N; i++)
printf("%d ", a[i]);
return 0;
}
```