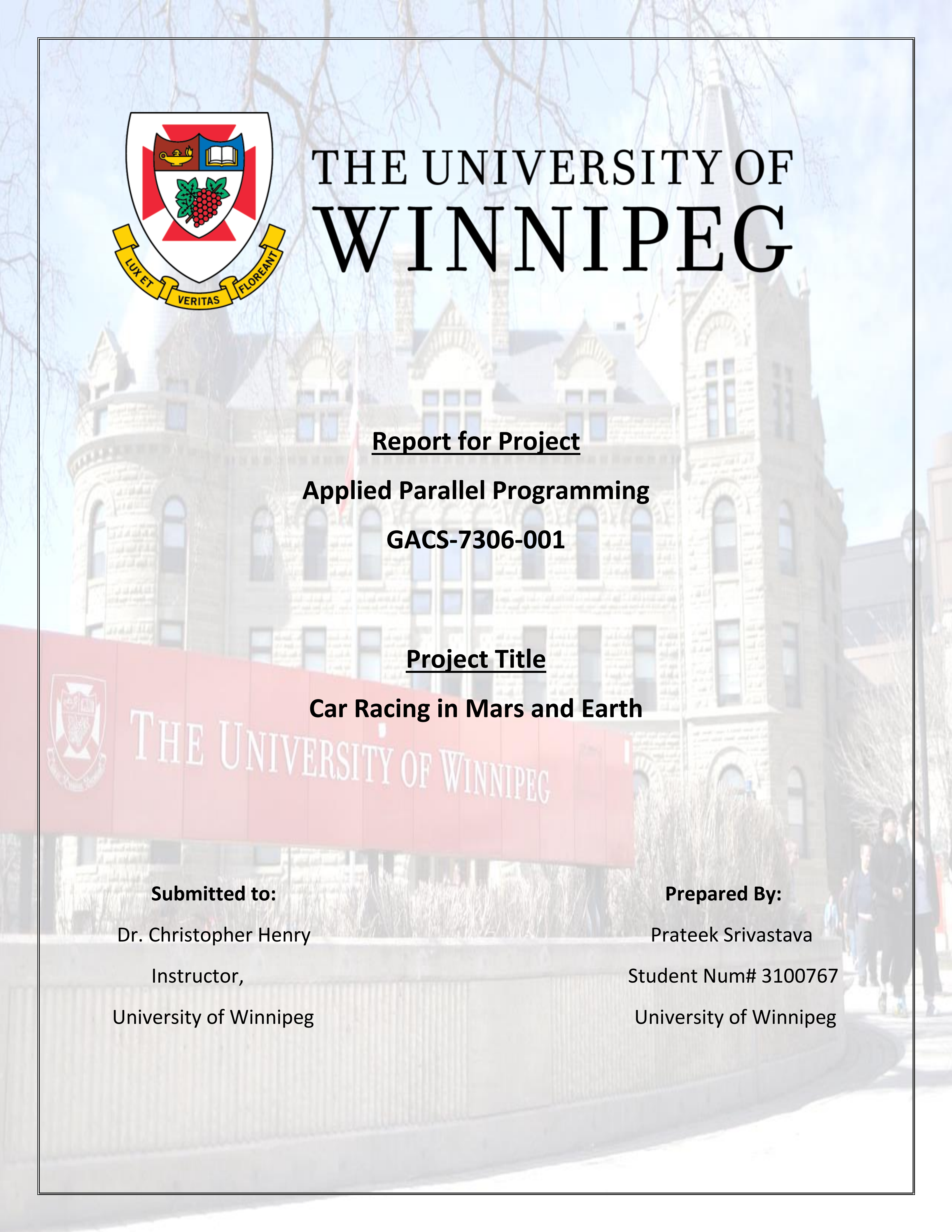


THE UNIVERSITY OF WINNIPEG

Report for Project

Applied Parallel Programming

GACS-7306-001

Project Title

Car Racing in Mars and Earth

Submitted to:

Dr. Christopher Henry

Instructor,

University of Winnipeg

Prepared By:

Prateek Srivastava

Student Num# 3100767

University of Winnipeg

Introduction:

Motivation:

Parallel programming is the field which is growing fast. Since a long parallel programming is a part of developments in fields of science and space. However, high end video games have made it very popular to the common person. Now, developer wants to have their career in parallel programming.

While studying parallel programming first time, I observed there are lot of challenges existing in field of technology which can have a simple solution if we try to solve it in a parallel manner.

While I was going through the course, I came to know about the quick sort, divide and conquer algorithms have been developed parallel.

I came with an idea that if I could also implement bubble sort in a parallel manner, then it could be useful, if we have huge numbers of cars and all are running with different speeds. Here I got that solution and the difference between CPU (Central Processing Unit) and GPU (Graphics Processing Unit) is remarkable.

Problem Definition:

- There are 100,000 cars. There is a competition in the planet Earth and in the planet Mars where all car are participating. The planet mars is chosen as there is a racing track available to accommodate all cars together. In earth there is a limited number of tracks are available.
- The first challenge is that all cars have their own speed and specification. So as competition begins all cars achieve their maximum speed and running on track. Now cars do not change their speed. After this moment all cars are running with same speed. The winner would be who reaches the finish line first, or simply the car having maximum speed which keeps running with the maximum speed.
- Each time one car passes another car positions are swapped in the recorder.
- The car can pass the car which is near to it. That means we can not simply choose a car and put it in front of other car just because it has highest speed. It has to pass all other cars, relative speed comes in picture.
- Eventually car with highest speed will be in front of all cars and will be declared as winner.

Assumptions:

- When the recorder begins recording the cars already attained their maximum speed, after that there is no change in the car's speed.
- There is a same competition going on Earth, where at one time when one car passes the others all other cars do not pass each one. (Sequential race and Dependency).
- The competition on different planet begins and end at same time, we actually interested in the overall time to complete the whole competition.

Outcome :

- The result should be that all cars got their ranking according to their speed. Cars may have same ranking if they would have same maximum speed.
- The result will be giving each car its ranking. Even if the car with maximum speed reaches first, still competition will going on, and give all cars their respective rankings.

Theoretical Framework

Theory Used in Project:

The task while solving this problem is to parallelize any sorting algorithm. I chose to go with Bubble-Sort.

What is Bubble Sort:

- Bubble sort is sometime referred to as a 'Sinking Sort'. It compares the adjacent elements and if the higher number is prior to lower number, it swaps and do the same with next pair of numbers.
- It keeps repeating this trend until all numbers are sorted.

How Bubble sort works:

Suppose we want to sort the list 6 2 5 3 9

First Pass
(**6 2** 5 3 9) → (**2 6** 5 3 9), Here, algorithm compares the first two elements, and swaps since 6> 2.
(2 **6 5** 3 9) → (2 **5 6** 3 9), Swap since 6 > 5
(2 5 **6 1** 9) → (2 5 **3 6** 9), Swap since 6 > 3
(2 5 3 **6 9**) → (2 5 3 **6 9**), Now, since these elements are already in order (9 > 6), algorithm
does not swap them.

Second Pass
(**2 5** 3 6 9) → (**2 5** 3 6 9)
(2 **5 3** 6 9) → (2 **3 5** 6 9), Swap since 5 > 3
(2 3 **5 6** 9) → (2 3 **5 6** 9)
(2 3 5 **6 9**) → (2 3 5 **6 9**)

Shortcoming of Bubble Sort:

- There are some short coming with the bubble sort which make it in efficient in general and particular to solve the problem in my project.
- Bubble sort performs in a serial manner. This would take more time if we have inputs in million digits.
- Bubble sort is very slow.
- Bubble sort checks all sorted numbers every time during iterations.
- Time complexity of Bubble sort in worst and average case is $O(n^2)$ where n shows the input numbers.

Paralleled version of Bubble sort or Odd-Even Transposition Sort

What is Odd-Even Transposition Sort:

Odd-Even Transposition Sort is a parallel **sorting** algorithm. It is based on the **Bubble Sort** technique of comparing two **numbers** and switching them if the first is greater than the second, to achieve a left to right ascending ordering.

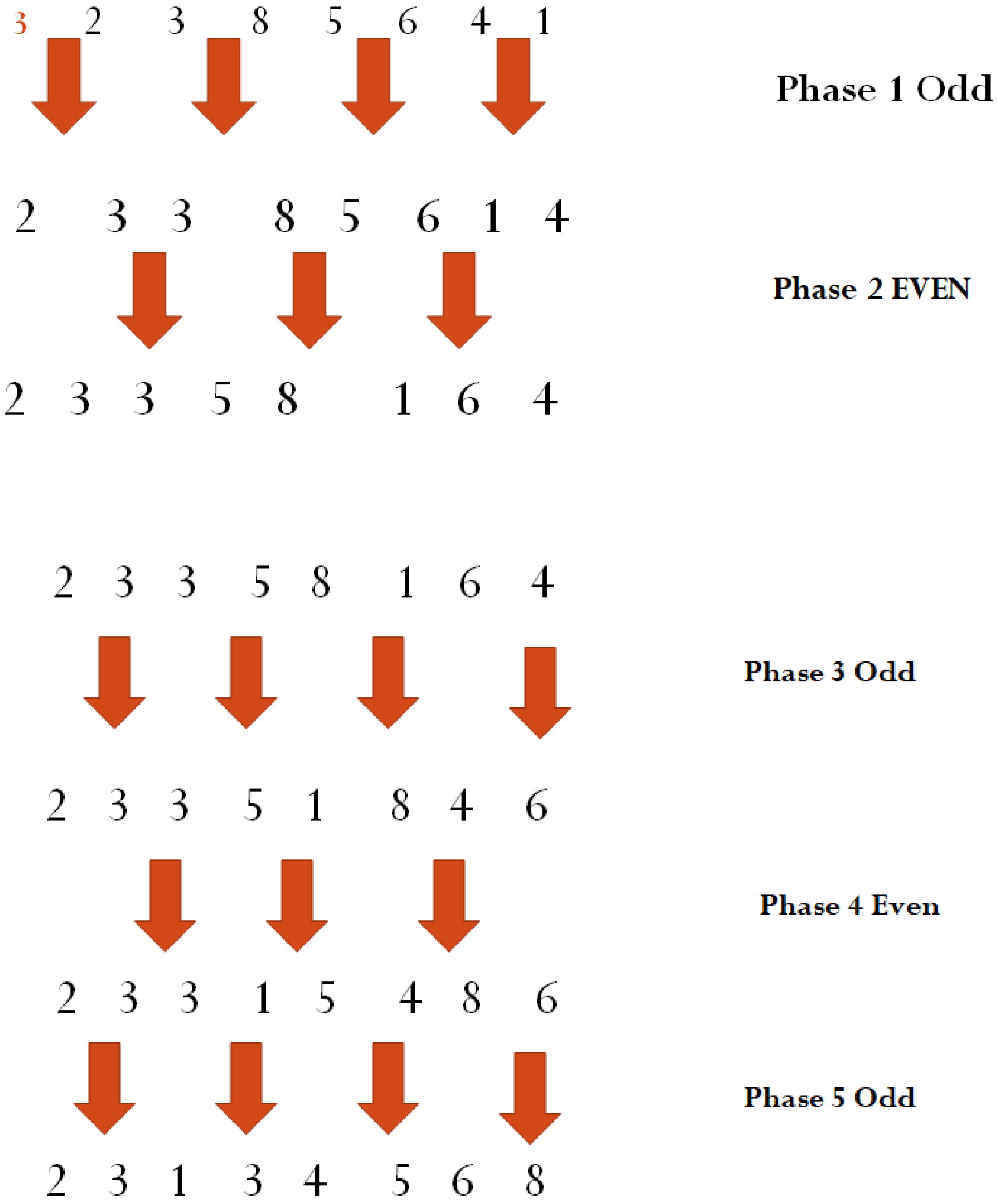
How Odd-Even Transposition Sort works (Algorithm)

The algorithm then operates by alternating between an odd and an even phase :

1. In the even phase, even numbered processors(processor i) communicate with the next odd numbered processors (processor i+1). In this communication process, the two sub-lists for each 2communicating processes are merged together. The upper half of the list is then kept in the higher number processor and the lower half is put in the lower number processor.

2. In the odd phase, odd number processors (processor i) communicate with the previous even number processors (i-1) in exactly the same way as in the even phase.

- Input Array- array = {3, 2, 3, 8, 5, 6, 4, 1}



3 2 3 8 5 6 4 1

Phase 1 Odd

2 3 3 8 5 6 1 4

Phase 2 EVEN

2 3 3 5 8 1 6 4

2 3 3 5 8 1 6 4

Phase 3 Odd

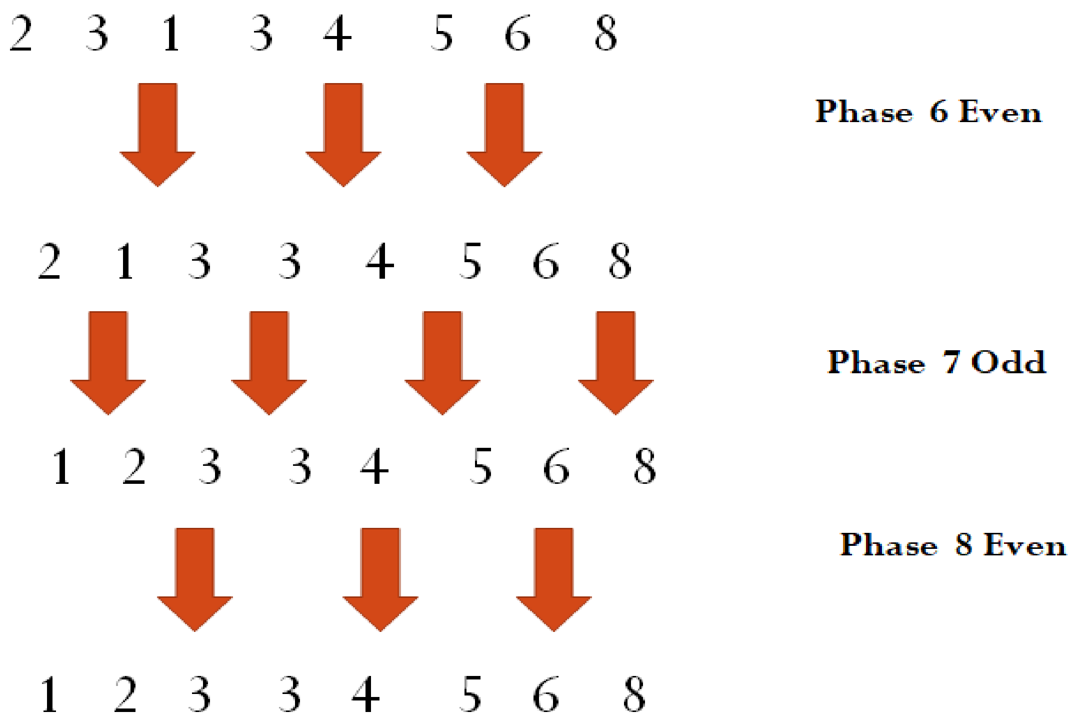
2 3 3 5 1 8 4 6

Phase 4 Even

2 3 3 1 5 4 8 6

Phase 5 Odd

2 3 1 3 4 5 6 8



<u>Part of the Program which will be parallelized</u>

- // A function to implement bubble sort

```
 void bubbleSort(int arr[], int n)
 {
    int i, j;
    for (i = 0; i < n-1; i++)
    // Last i elements are already in place
     for (j = 0; j < n-i-1; j++)
      if (arr[j] > arr[j+1])
      swap(&arr[j], &arr[j+1]);
}
```

- This loop will be replaced by parallelized kernel code.

System Description:

Hardware:

- GPU- Tesla K40c
- CPU-12 Core Intel Xeon

Software:

- Linux GNU/Linux

Development Environment

- C++ with CUDA API
- nsight

Code

```
#include<stdio.h>
#include<cuda.h>
#include <stdlib.h>
#include <iostream>
#include <time.h>
#include <math.h>

#define N 100000
using namespace std;
static const long BLK_SIZE =1000 ;
#define CUDA_CHECK_RETURN(value) {
                      \
      cudaError_t _m_cudaStat = value;
                 \
      if (_m_cudaStat != cudaSuccess) {
                      \
           fprintf(stderr, "Error %s at line %d in file %s\n",
                 \
                      cudaGetErrorString(_m_cudaStat), __LINE__,
__FILE__);       \
           exit(1);
                            \
      } }
```

```
__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%2==0)  //even phase
            {
                if(c[threadIdx.x]>c[threadIdx.x+1])
                {
                    int temp=c[threadIdx.x];
                    c[threadIdx.x]=c[threadIdx.x+1];
                    c[threadIdx.x+1]=temp;
                }

            __syncthreads();
            }
            else     //odd phase
            {
                if(c[threadIdx.x]>c[threadIdx.x+1])
                {
                    int temp=c[threadIdx.x];
                    c[threadIdx.x]=c[threadIdx.x+1];
                    c[threadIdx.x+1]=temp;
                }

            __syncthreads();
            }
    }

}

void swap(int *xp, int *yp)
{
     int temp = *xp;
     *xp = *yp;
     *yp = temp;
}

// An optimized version of Bubble Sort
void bubbleSort(int arr[], int n)
{

}

int main()
{
```

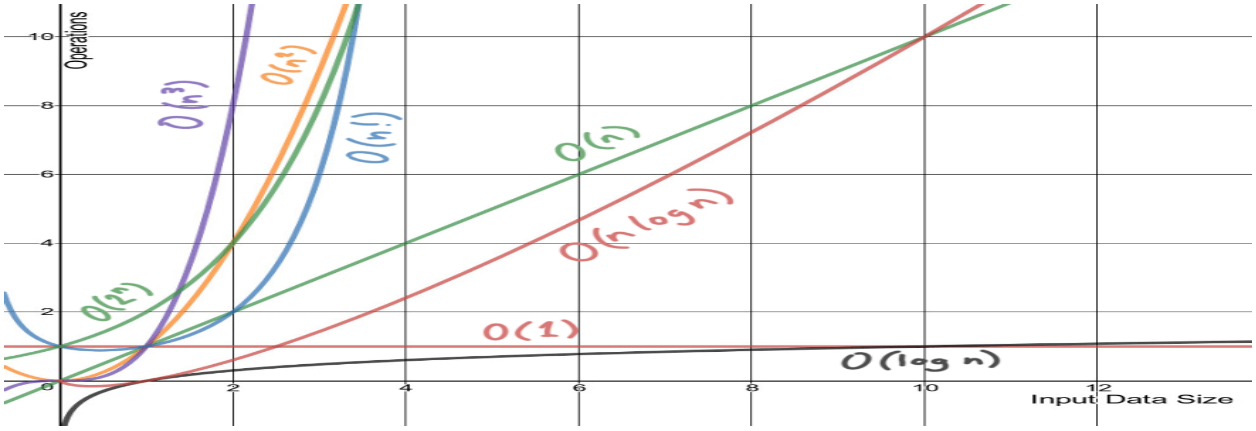
```
int a[N],b[N];
    for (int i = 0; i < N; i++) {
           a[i] = (float) rand() / (float) RAND_MAX * 100;

     }


  printf("ORIGINAL ARRAY : \n");
  for(int i=0;i<N;i++)
          {

          printf("%d ",a[i]);
          }


  int *c,*count;
  int k=N;


  cudaMalloc((void**)&c,sizeof(int)*N);
  cudaMalloc((void**)&count,sizeof(int));
  cudaMemcpy(c,&a,sizeof(int)*N,cudaMemcpyHostToDevice);
  cudaMemcpy(count,&k,sizeof(int),cudaMemcpyHostToDevice);


  //Time kernel launch
      //Time kernel launch
      cudaEvent_t start, stop;
      CUDA_CHECK_RETURN(cudaEventCreate(&start));
      CUDA_CHECK_RETURN(cudaEventCreate(&stop));
      float elapsedTime;

      CUDA_CHECK_RETURN(cudaEventRecord(start, 0));



  sort<<< ceil(N/(float)BLK_SIZE),BLK_SIZE >>>(c,count);

  CUDA_CHECK_RETURN(cudaEventRecord(stop, 0));

      CUDA_CHECK_RETURN(cudaEventSynchronize(stop));
      CUDA_CHECK_RETURN(cudaEventElapsedTime(&elapsedTime, start,
stop));
      CUDA_CHECK_RETURN(cudaThreadSynchronize()); // Wait for the GPU
launched work to complete
      CUDA_CHECK_RETURN(cudaGetLastError()); //Check if an error
occurred in device code
      CUDA_CHECK_RETURN(cudaEventDestroy(start));
      CUDA_CHECK_RETURN(cudaEventDestroy(stop));
      cout << "done.\nElapsed kernel time: " << elapsedTime << " ms\n";

      cout << "Copying results back to host .... "<<endl;
  cudaMemcpy(&b,c,sizeof(int)*N,cudaMemcpyDeviceToHost);
  printf("\nSORTED ARRAY : \n");

  for(int i=0;i<N;i++)
      {
         printf("%d ",b[i]);
      }

  //Add code to time host calculations
     clock_t st, ed;

     st = clock();
     //bool valid = true;

  //bubbleSort(a,N);

     int i, j;
     bool swapped;
     for (i = 0; i < N-1; i++)
     {
           swapped = false;
           for (j = 0; j < N-i-1; j++)
           {
                 if (a[j] > a[j+1])
                 {
                 swap(&a[j], &a[j+1]);
                 swapped = true;
                 }
           }

           // IF no two elements were swapped by inner loop, then
break
           if (swapped == false)
                 break;
     }

  printf("\n");
        printf("BYCPU");
        printf("\n");
  for(int i=0;i<N;i++)
        {

        printf("%d ",a[i]);
        }
  ed = clock() - st;
     cout << "Elapsed time on host: " << ((float) ed) / CLOCKS_PER_SEC
* 1000
                 << " ms" << endl;

}
```

Complexity Analysis

- Time Complexity : O(N^2) where, N = Number of elements in the input array. Space Complexity : O(1). Just like bubble sort this is also an in-place algorithm. We first do bubble sort on odd indexed elements and then a bubble sort on the even indexed elements.

- Odd-Even transposition sort actually require input string/2 iterations. As one it choose for even phase and one it choose for odd phase.

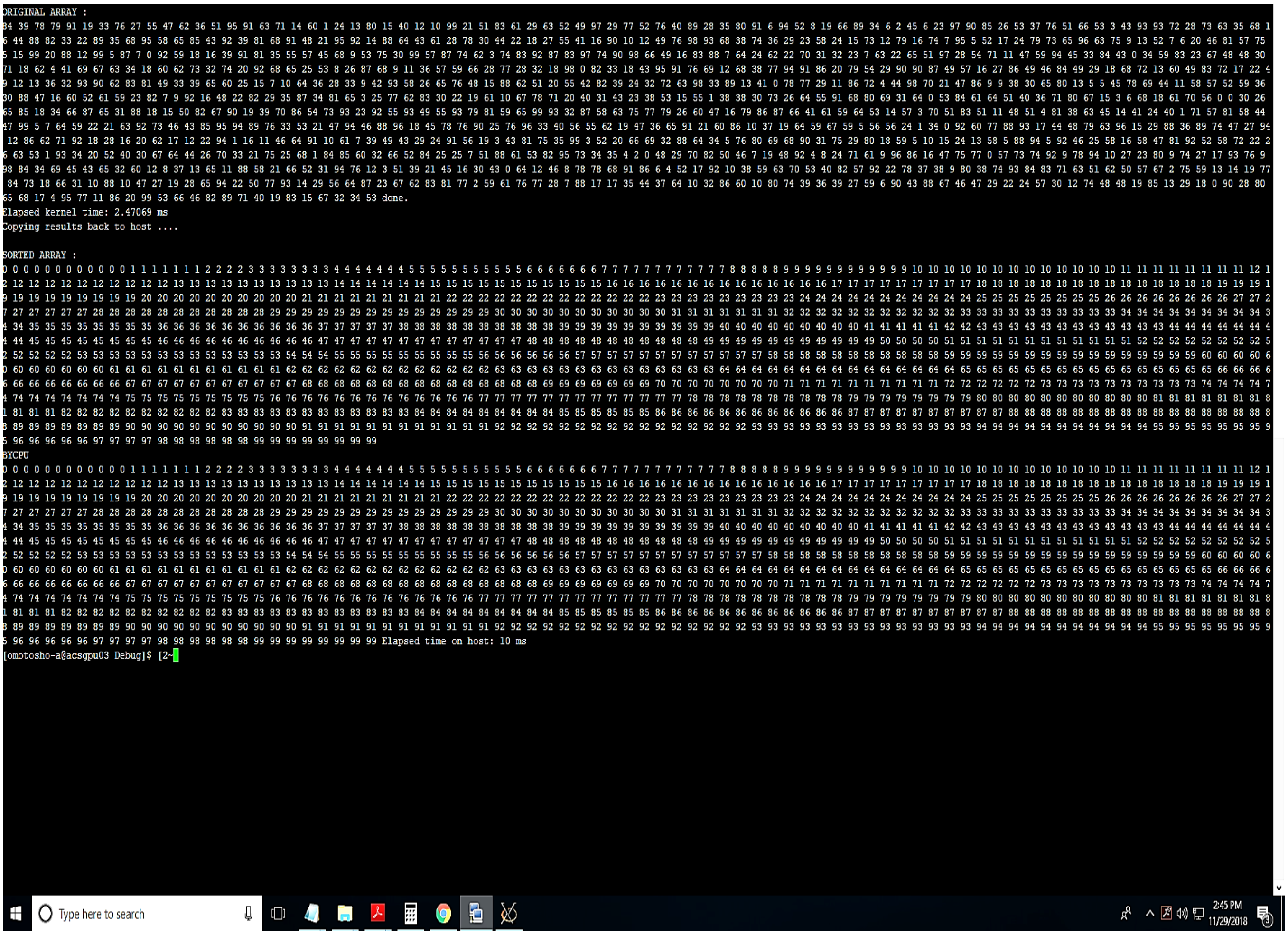
Worst Case Senario : O(n^2)
BestCase Senario : O(n)



Results:

When input string is of 1000 length of numbers.

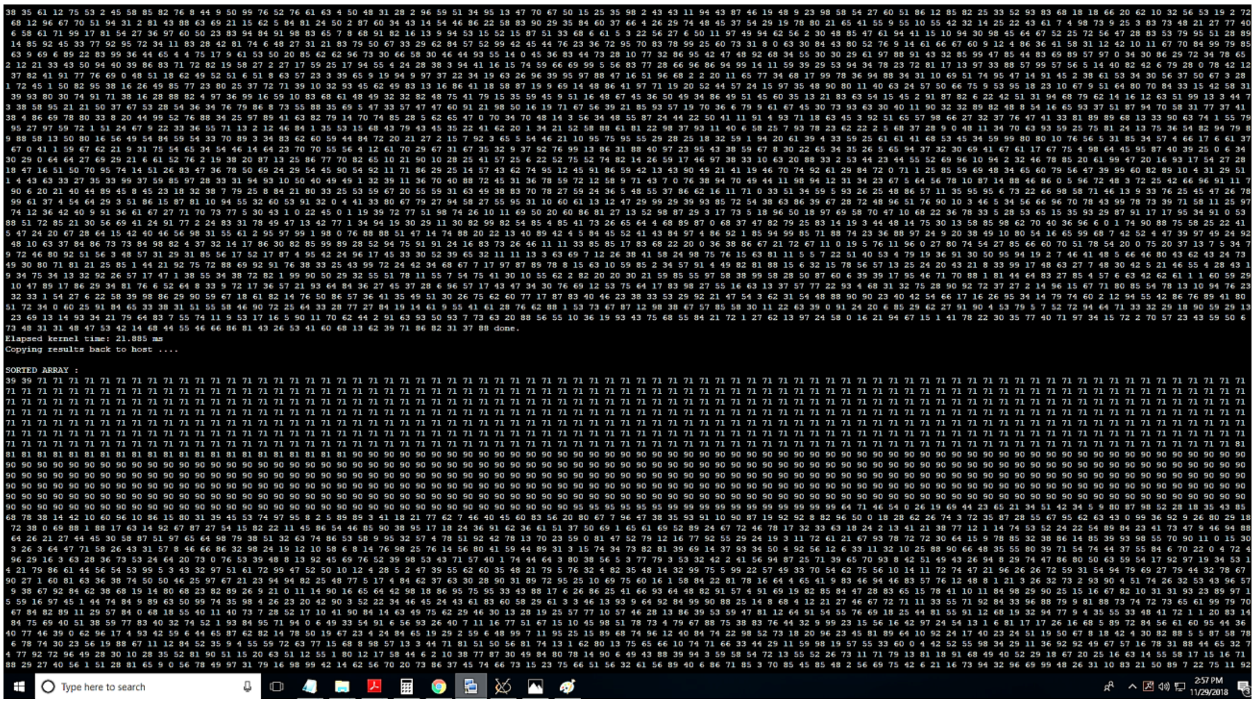
CPU= 10 ms   GPU= 2.47 ms



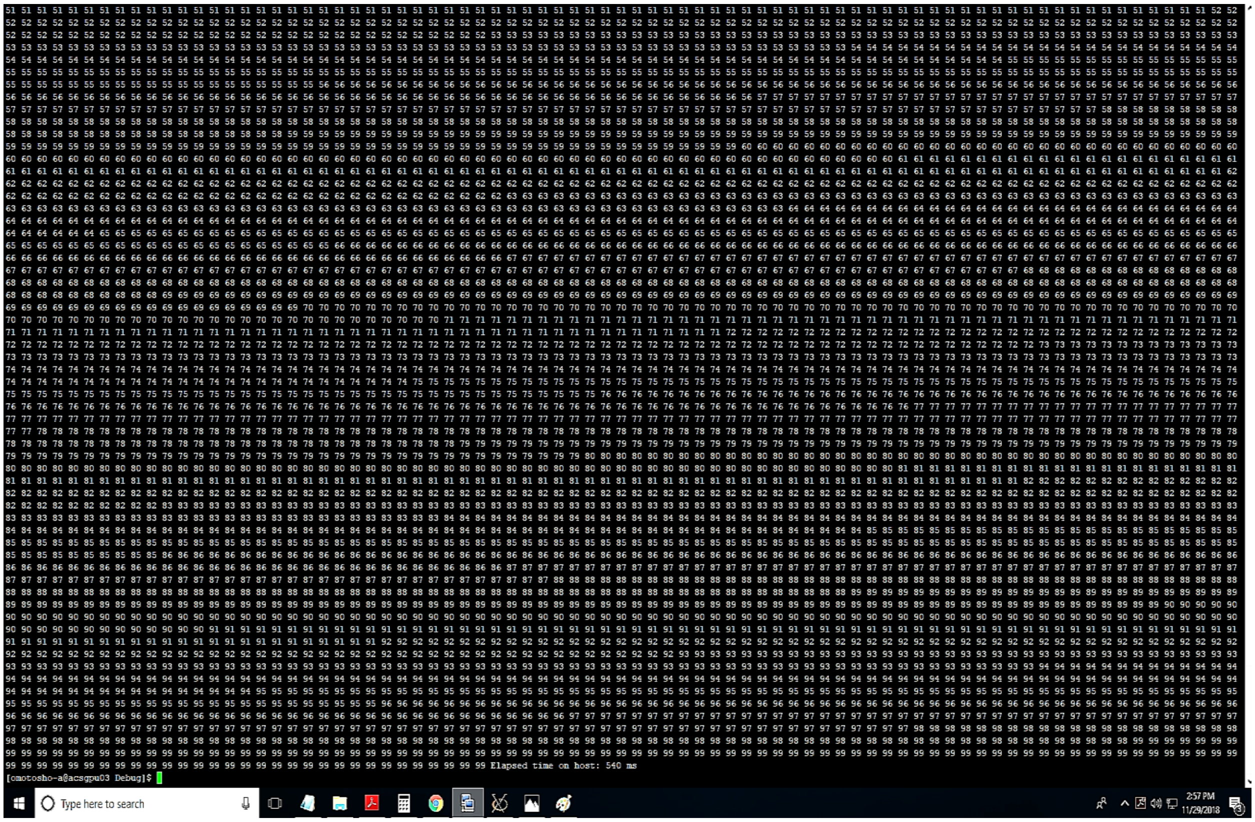
```
ORIGINAL ARRAY :
34 39 78 79 91 19 33 76 27 55 47 62 36 51 95 91 63 71 14 60 1 24 13 80 15 40 12 10 99 21 51 83 61 29 63 52 49 97 29 77 52 76 40 89 28 35 80 91 6 94 52 8 19 66 89 34 6 2 45 6 23 97 90 85 26 53 37 76 51 66 53 3 43 93 93 72 28 73 63 35 68 1
6 44 88 82 33 22 89 35 68 95 58 65 85 43 92 39 81 68 91 48 21 95 92 14 88 64 43 61 28 78 30 44 22 18 27 55 41 16 90 10 12 49 76 98 93 68 38 74 36 29 23 58 24 15 73 12 79 16 74 7 95 5 52 17 24 79 73 65 96 63 75 9 13 52 7 6 20 46 81 57 75
5 15 99 20 88 12 99 5 87 7 0 92 59 18 16 39 91 81 35 55 57 45 68 9 53 75 30 99 57 87 74 62 3 74 83 92 87 83 97 74 90 98 66 49 16 83 88 7 64 24 62 22 70 31 32 23 7 63 22 65 51 97 28 54 71 11 47 59 94 45 33 84 43 0 34 59 83 23 67 48 48 30
71 18 62 4 41 69 67 63 34 18 60 62 73 32 74 20 92 68 65 25 53 8 26 87 68 9 11 36 57 59 66 28 77 28 32 18 98 0 82 33 18 43 95 91 76 69 12 68 38 77 94 91 86 20 79 54 29 90 90 87 49 57 16 27 86 49 46 84 49 29 18 68 72 13 60 49 83 72 17 22 4
9 12 13 36 32 93 90 62 83 81 49 33 39 65 60 25 15 7 10 64 36 28 33 9 42 93 58 26 65 76 48 15 88 62 51 20 55 42 82 39 24 32 72 63 98 33 89 13 41 0 78 77 29 11 86 72 4 44 98 70 21 47 86 9 9 38 30 65 80 13 5 5 45 78 69 44 11 58 57 52 59 36
30 88 47 16 60 52 61 59 23 82 7 9 92 16 48 22 82 29 35 87 34 81 65 3 25 77 62 83 30 22 19 61 10 67 78 71 20 40 31 43 23 38 53 15 55 1 38 38 30 73 26 64 55 91 68 80 69 31 64 0 53 84 61 64 51 40 36 71 80 67 15 3 6 68 18 61 70 56 0 0 30 26
65 85 18 34 66 87 65 31 88 18 15 50 82 67 90 19 39 70 86 54 73 93 23 92 55 93 49 55 93 79 81 59 65 99 93 32 87 58 63 75 77 79 26 60 47 16 79 86 87 66 41 61 59 64 53 14 57 3 70 51 83 51 11 48 51 4 81 38 63 45 14 41 24 40 1 71 57 81 58 44
47 99 5 7 64 59 22 21 63 92 73 46 43 85 95 94 89 76 33 53 21 47 94 46 88 96 18 45 78 76 90 25 76 96 33 40 56 55 62 19 47 36 65 91 21 60 86 10 37 19 64 59 67 59 5 56 56 24 1 34 0 92 60 77 88 93 17 44 48 79 63 96 15 29 88 36 89 74 47 27 94
 12 86 62 71 92 18 28 16 20 62 17 12 22 94 1 16 11 46 64 91 10 61 7 39 49 43 29 24 91 56 19 3 43 81 75 35 99 3 52 20 66 69 32 88 64 34 5 76 80 69 68 90 31 75 29 80 18 59 5 10 15 24 13 58 5 88 94 5 92 46 25 58 16 58 47 81 92 52 58 72 22 2
6 63 53 1 93 34 20 52 40 30 67 64 44 26 70 33 21 75 25 68 1 84 85 60 32 66 52 84 25 25 7 51 88 61 53 82 95 73 34 35 4 2 0 48 29 70 82 50 46 7 19 48 92 4 8 24 71 61 9 96 86 16 47 75 77 0 57 73 74 92 9 78 94 10 27 23 80 9 74 27 17 93 76 9
98 84 34 69 45 43 65 32 60 12 8 37 13 65 11 88 58 21 66 52 31 94 76 12 3 51 39 21 45 16 30 43 0 64 12 46 8 78 78 68 91 86 6 4 52 17 92 10 38 59 63 70 53 40 82 57 92 22 78 37 38 9 80 38 74 93 84 83 71 63 51 62 50 57 67 2 75 59 13 14 19 77
 84 73 18 66 31 10 88 10 47 27 19 28 65 94 22 50 77 93 14 29 56 64 87 23 67 62 83 81 77 2 59 61 76 77 28 7 88 17 17 35 44 37 64 10 32 86 60 10 80 74 39 36 39 27 59 6 90 43 88 67 46 47 29 22 24 57 30 12 74 48 48 19 85 13 29 18 0 90 28 80
65 68 17 4 95 77 11 86 20 99 53 66 46 82 89 71 40 19 83 15 67 32 34 53 done.
Elapsed kernel time: 2.47069 ms
Copying results back to host ....

SORTED ARRAY :
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 2 2 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 10 10 10 10 10 10 10 10 10 10 10 10 10 11 11 11 11 11 11 11 11 12 1
2 12 12 12 12 12 12 12 12 12 12 13 13 13 13 13 13 13 13 13 13 14 14 14 14 14 14 15 15 15 15 15 15 15 15 15 15 15 16 16 16 16 16 16 16 16 16 16 16 16 16 16 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 19 19 19 1
9 19 19 19 19 19 19 19 19 20 20 20 20 20 20 20 20 20 20 21 21 21 21 21 21 21 21 21 22 22 22 22 22 22 22 22 22 22 22 22 22 23 23 23 23 23 23 23 23 23 24 24 24 24 24 24 24 24 24 24 24 25 25 25 25 25 25 25 25 26 26 26 26 26 26 26 26 27 27 2
7 27 27 27 27 27 28 28 28 28 28 28 28 28 28 28 28 29 29 29 29 29 29 29 29 29 29 29 29 29 29 30 30 30 30 30 30 30 30 30 30 30 31 31 31 31 31 31 31 32 32 32 32 32 32 32 32 32 32 32 33 33 33 33 33 33 33 33 33 33 34 34 34 34 34 34 34 34 34 3
4 34 35 35 35 35 35 35 35 35 36 36 36 36 36 36 36 36 36 36 37 37 37 37 37 38 38 38 38 38 38 38 38 38 38 39 39 39 39 39 39 39 39 39 39 40 40 40 40 40 40 40 40 40 41 41 41 41 41 42 42 43 43 43 43 43 43 43 43 43 43 43 43 44 44 44 44 44 44 4
4 44 45 45 45 45 45 45 45 45 46 46 46 46 46 46 46 46 46 46 47 47 47 47 47 47 47 47 47 47 47 47 47 48 48 48 48 48 48 48 48 48 48 48 49 49 49 49 49 49 49 49 49 49 49 50 50 50 50 51 51 51 51 51 51 51 51 51 51 51 51 52 52 52 52 52 52 52 52 5
2 52 52 52 52 53 53 53 53 53 53 53 53 53 53 53 53 53 54 54 54 55 55 55 55 55 55 55 55 55 56 56 56 56 56 56 57 57 57 57 57 57 57 57 57 57 57 57 58 58 58 58 58 58 58 58 58 58 58 59 59 59 59 59 59 59 59 59 59 59 59 59 59 59 59 60 60 60 60 6
0 60 60 60 60 60 60 61 61 61 61 61 61 61 61 61 61 61 62 62 62 62 62 62 62 62 62 62 62 62 62 63 63 63 63 63 63 63 63 63 63 63 63 63 63 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 66 66 66 6
6 66 66 66 66 66 66 66 67 67 67 67 67 67 67 67 67 67 67 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 69 69 69 69 69 69 69 70 70 70 70 70 70 70 70 71 71 71 71 71 71 71 71 71 71 72 72 72 72 72 72 73 73 73 73 73 73 73 73 73 73 74 74 74 74 7
4 74 74 74 74 74 74 74 75 75 75 75 75 75 75 75 75 76 76 76 76 76 76 76 76 76 76 76 76 76 77 77 77 77 77 77 77 77 77 77 77 77 77 78 78 78 78 78 78 78 78 78 78 79 79 79 79 79 79 79 79 80 80 80 80 80 80 80 80 80 80 80 81 81 81 81 81 81 81 8
1 81 81 81 82 82 82 82 82 82 82 82 82 82 83 83 83 83 83 83 83 83 83 83 83 83 84 84 84 84 84 84 84 84 84 85 85 85 85 85 85 86 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 8
8 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 91 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 94 94 94 94 94 95 95 95 95 95 95 95 9
5 96 96 96 96 96 97 97 97 97 98 98 98 98 98 98 99 99 99 99 99 99 99 99
BYCPU
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 2 2 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 10 10 10 10 10 10 10 10 10 10 10 10 10 11 11 11 11 11 11 11 11 12 1
2 12 12 12 12 12 12 12 12 12 12 13 13 13 13 13 13 13 13 13 13 14 14 14 14 14 14 15 15 15 15 15 15 15 15 15 15 15 16 16 16 16 16 16 16 16 16 16 16 16 16 16 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 19 19 19 1
9 19 19 19 19 19 19 19 19 20 20 20 20 20 20 20 20 20 20 21 21 21 21 21 21 21 21 21 22 22 22 22 22 22 22 22 22 22 22 22 22 23 23 23 23 23 23 23 23 23 24 24 24 24 24 24 24 24 24 24 24 25 25 25 25 25 25 25 25 26 26 26 26 26 26 26 26 27 27 2
7 27 27 27 27 27 28 28 28 28 28 28 28 28 28 28 28 29 29 29 29 29 29 29 29 29 29 29 29 29 29 30 30 30 30 30 30 30 30 30 30 30 31 31 31 31 31 31 31 32 32 32 32 32 32 32 32 32 32 32 33 33 33 33 33 33 33 33 33 33 34 34 34 34 34 34 34 34 34 3
4 34 35 35 35 35 35 35 35 35 36 36 36 36 36 36 36 36 36 36 37 37 37 37 37 38 38 38 38 38 38 38 38 38 38 39 39 39 39 39 39 39 39 39 39 40 40 40 40 40 40 40 40 40 41 41 41 41 41 42 42 43 43 43 43 43 43 43 43 43 43 43 43 44 44 44 44 44 44 4
4 44 45 45 45 45 45 45 45 45 46 46 46 46 46 46 46 46 46 46 47 47 47 47 47 47 47 47 47 47 47 47 47 48 48 48 48 48 48 48 48 48 48 48 49 49 49 49 49 49 49 49 49 49 49 50 50 50 50 51 51 51 51 51 51 51 51 51 51 51 51 52 52 52 52 52 52 52 52 5
2 52 52 52 52 53 53 53 53 53 53 53 53 53 53 53 53 53 54 54 54 55 55 55 55 55 55 55 55 55 56 56 56 56 56 56 57 57 57 57 57 57 57 57 57 57 57 57 58 58 58 58 58 58 58 58 58 58 58 59 59 59 59 59 59 59 59 59 59 59 59 59 59 59 59 60 60 60 60 6
0 60 60 60 60 60 60 61 61 61 61 61 61 61 61 61 61 61 62 62 62 62 62 62 62 62 62 62 62 62 62 63 63 63 63 63 63 63 63 63 63 63 63 63 63 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 66 66 66 6
6 66 66 66 66 66 66 66 67 67 67 67 67 67 67 67 67 67 67 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 69 69 69 69 69 69 69 70 70 70 70 70 70 70 70 71 71 71 71 71 71 71 71 71 71 72 72 72 72 72 72 73 73 73 73 73 73 73 73 73 73 74 74 74 74 7
4 74 74 74 74 74 74 74 75 75 75 75 75 75 75 75 75 76 76 76 76 76 76 76 76 76 76 76 76 76 77 77 77 77 77 77 77 77 77 77 77 77 77 78 78 78 78 78 78 78 78 78 78 79 79 79 79 79 79 79 79 80 80 80 80 80 80 80 80 80 80 80 81 81 81 81 81 81 81 8
1 81 81 81 82 82 82 82 82 82 82 82 82 82 83 83 83 83 83 83 83 83 83 83 83 83 84 84 84 84 84 84 84 84 84 85 85 85 85 85 85 86 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 8
8 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 91 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 94 94 94 94 94 95 95 95 95 95 95 95 9
5 96 96 96 96 96 97 97 97 97 98 98 98 98 98 98 99 99 99 99 99 99 99 99 Elapsed time on host: 10 ms
[omotosho-a@acsgpu03 Debug]$ [2~
```

When input string is of 10,000 length.

CPU=540 ms   GPU=21.88 ms



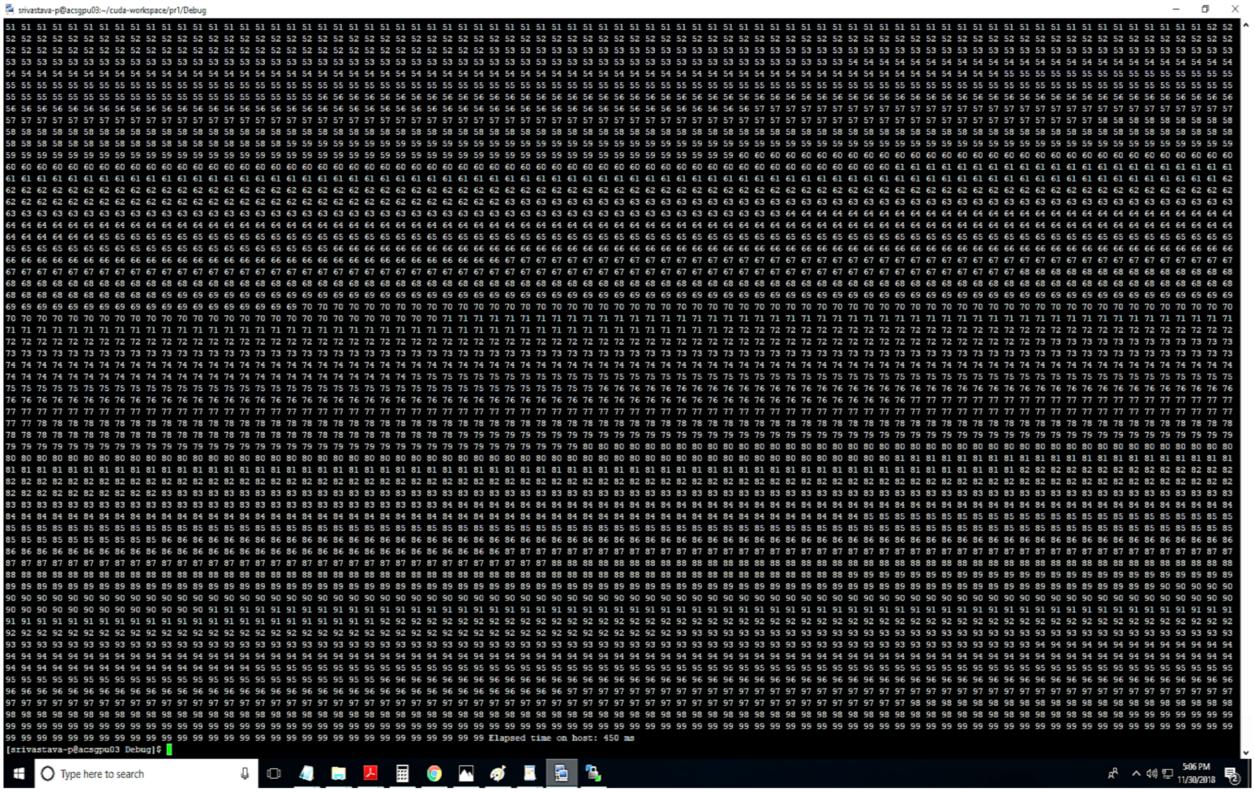
```
... done.
Elapsed kernel time: 21.885 ms
Copying results back to host ....

SORTED ARRAY :
39 39 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 ...
```



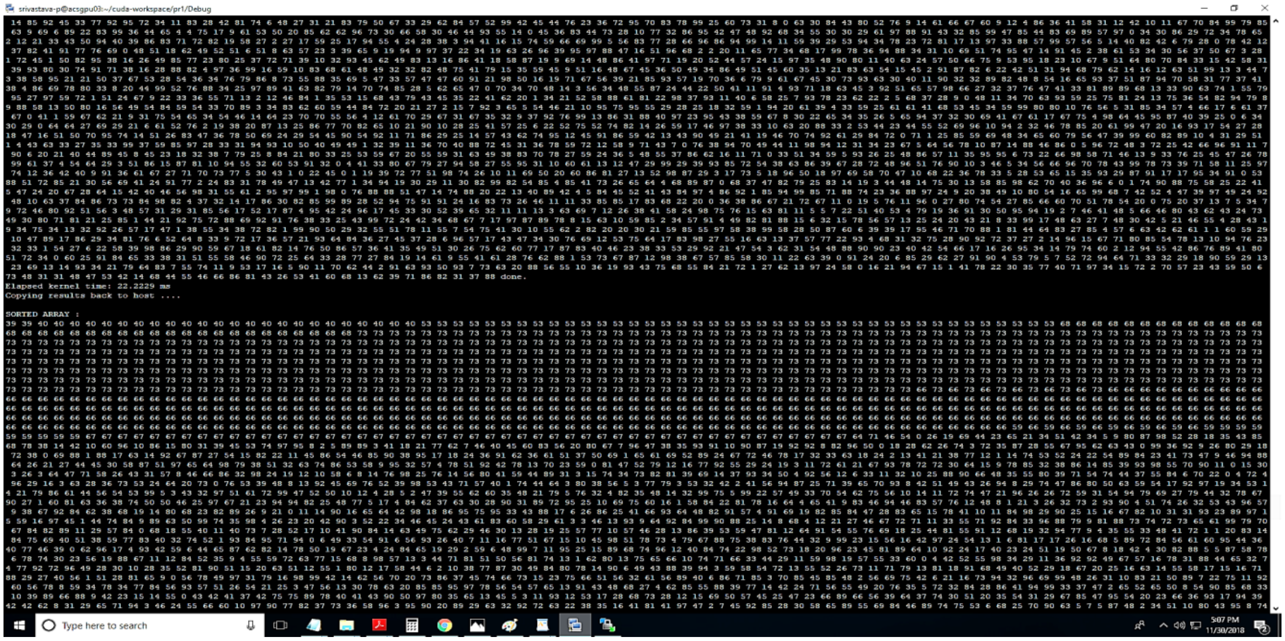
```
... 99 99 99 99 Elapsed time on host: 540 ms
[omotosho-a@acsgpu03 Debug]$ 
```

When input string is of 100,000 length.

CPU=450 ms   GPU=22.22 ms



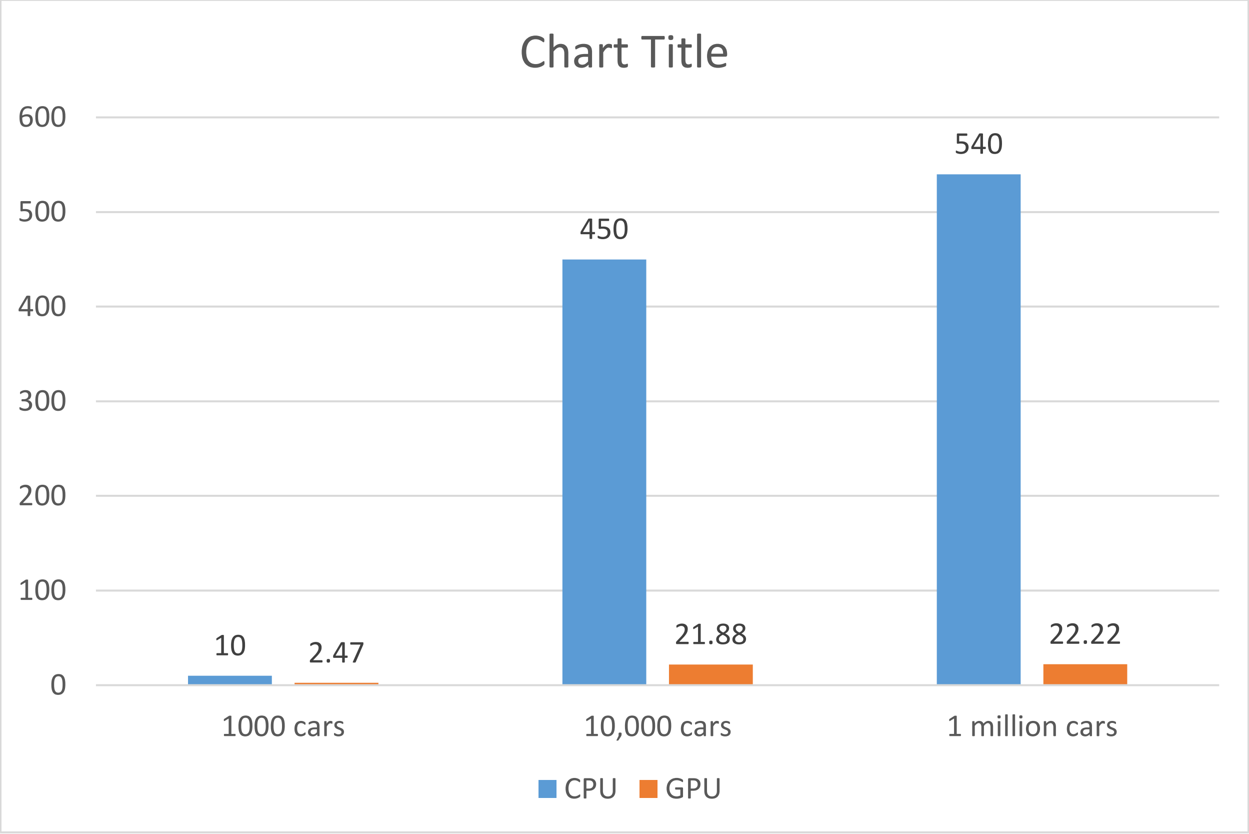
```
[illegible]
99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 Elapsed time on host: 450 ms
[srivastava-p@acsgpu03 Debug]$
```



```
[illegible]
73 48 31 31 48 47 53 42 14 68 44 55 46 66 86 81 43 26 53 41 60 68 13 62 39 71 86 82 31 37 88 done.
Elapsed kernel time: 22.2229 ms
Copying results back to host ....

SORTED ARRAY :
[illegible]
```

Conclusion

Input length= 1000	Input length= 10,000	Input length= 100,000
Kernel Time = 2.47 ms	Kernel Time = 21.88 ms	Kernel Time = 22.22 ms
CPU Time= 10 ms	CPU Time= 450 ms	CPU Time= 540 ms



References

- https://en.wikipedia.org/wiki/Odd%E2%80%93even_sort
- https://web.archive.org/web/20111028201105/http://homepages.ihug.co.nz/~aurora76/Malc/Sorting_Array.htm#Exchanging_Sort_Techniques
- https://books.google.ca/books?id=Mo2Q-TEwKGUC&pg=PA322&redir_esc=y#v=onepage&q&f=false
- https://books.google.ca/books?id=F9Y4oZ9qZnYC&pg=PA33&redir_esc=y#v=onepage&q&f=false
- http://liinwww.ira.uka.de/~thw/vl-hiroshima/slides-4.pdf
- https://github.com/Nalaka1693/pthread_odd_even_sort/blob/master/odd-even-sort.c
- http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/networks/nulleinsen.htm
- https://www.geeksforgeeks.org/odd-even-sort-brick-sort/