Math Set Program

Raunac Bhuiyan

Bourns College of Engineering, University of California Riverside

CS 147: GPU Computing and Programming

Prof. D. Wong

June 6, 2022

Overview

This program creates two sets. The sets have a collection of randomized numbers. However, the range of the numbers can vary depending on the size of the set. For example, if the first set contains 25 numbers, then the range of the random numbers will be between -25 and +25. Of course, this also applies to the second set. Furthermore, the sets do not contain any duplicate values. Then, the sets are arranged in ascending order. In other words, least to greatest. After the creation of the sets, logical operations can be performed. In layman's terms the following demonstrates how the operations are performed.

Ex. Set1 = {-3, 0, 8, 11}, Set2 = {0, 5, 8, 14}

Applying the intersection operation Set1 ∩ Set2 yields the new set {0, 8}

Applying the unison operation Set1 U Set2 yields the new set {-3, 0, 5, 8, 11, 14}

Applying the relative operation Set1 - Set2 yields the new set {-3, 11}

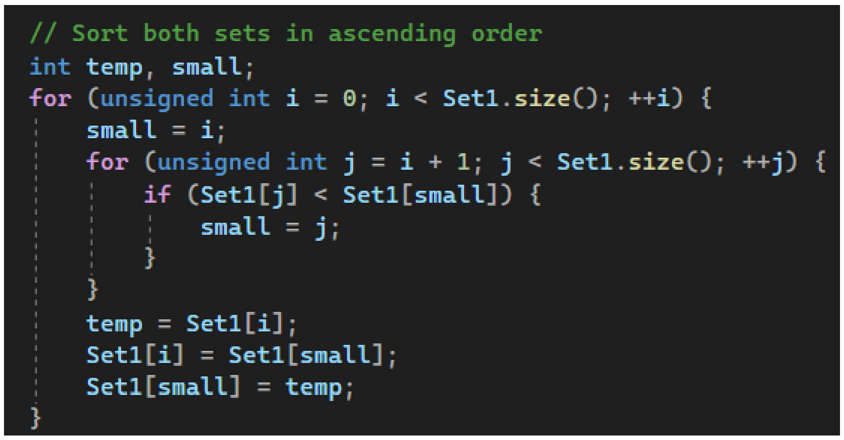
Applying the symmetric operation Set1 Δ Set2 yields the new set {-3, 5, 11, 14}

GPU Application

The Thrust library is similar to the standard C++ STL vector library. In this case, thrust made it convenient to utilize the GPU and parallelize certain parts of the program to make it faster.

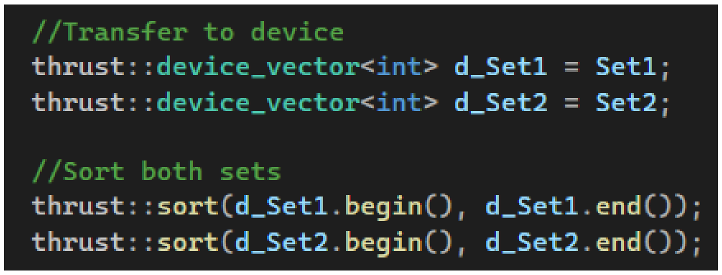
For instance, with the CPU, the selection sort algorithm was used to arrange the sets in ascending order. As shown in **Figure 1,** the code is written with a nested for loop. Nested for loops make the program slower. On the other hand, as shown in **Figure 2**, the vectors are transferred from host to device, then it is sorted. This makes the program faster.

Figure 1



```
// Sort both sets in ascending order
int temp, small;
for (unsigned int i = 0; i < Set1.size(); ++i) {
    small = i;
    for (unsigned int j = i + 1; j < Set1.size(); ++j) {
        if (Set1[j] < Set1[small]) {
            small = j;
        }
    }
    temp = Set1[i];
    Set1[i] = Set1[small];
    Set1[small] = temp;
}
```

Figure 2



```
//Transfer to device
thrust::device_vector<int> d_Set1 = Set1;
thrust::device_vector<int> d_Set2 = Set2;

//Sort both sets
thrust::sort(d_Set1.begin(), d_Set1.end());
thrust::sort(d_Set2.begin(), d_Set2.end());
```

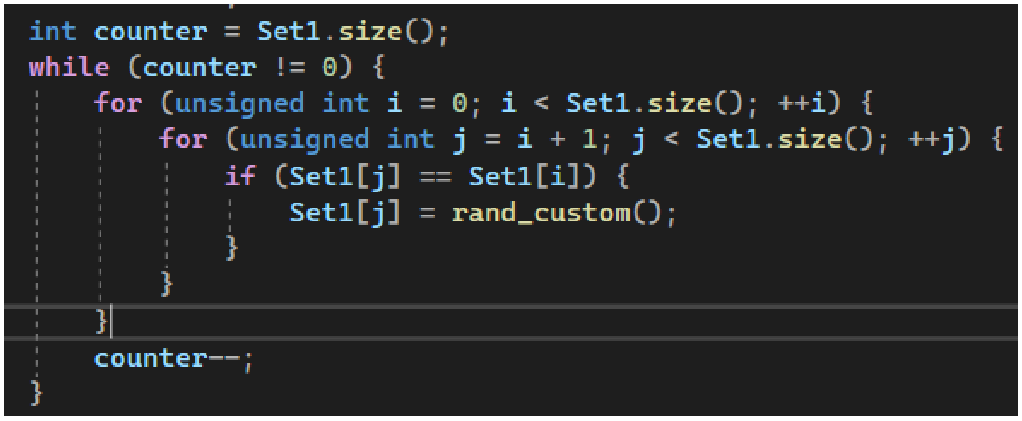
Implementation Details

- How to guarantee Sets will never have duplicate values?

  Suppose initial created Set is {-9, 3, 7, 13, 0, -52, -44, 7, -2, 3}

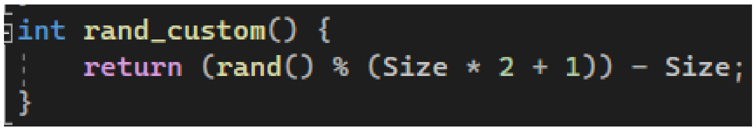
  However, 3 and 7 are repeated. Therefore, iterate through the Set and change 3 and 7 to a different randomized number. Though note that the vectors are iterated depending on the size of the vector. This will ensure a set will never have a duplicate. As shown in **Figure 3** and **Figure 4**, how the randomized numbers are generated and placed into the vectors.

Figure 3



```
int counter = Set1.size();
while (counter != 0) {
    for (unsigned int i = 0; i < Set1.size(); ++i) {
        for (unsigned int j = i + 1; j < Set1.size(); ++j) {
            if (Set1[j] == Set1[i]) {
                Set1[j] = rand_custom();
            }
        }
    }
    counter--;
}
```

Figure 4



```
int rand_custom() {
    return (rand() % (Size * 2 + 1)) - Size;
}
```

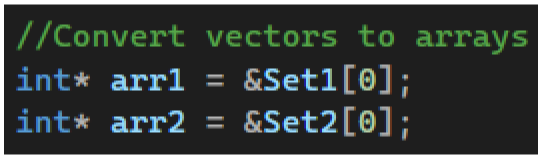
- How are the Sets sorted?

  The implementation details for this part of the program are explained on page 3.

- The operations

  Before printing the sets or applying the operations, as shown in **Figure 5**, both vectors are first converted to an array. The reason is that arrays are faster than vectors.

Figure 5

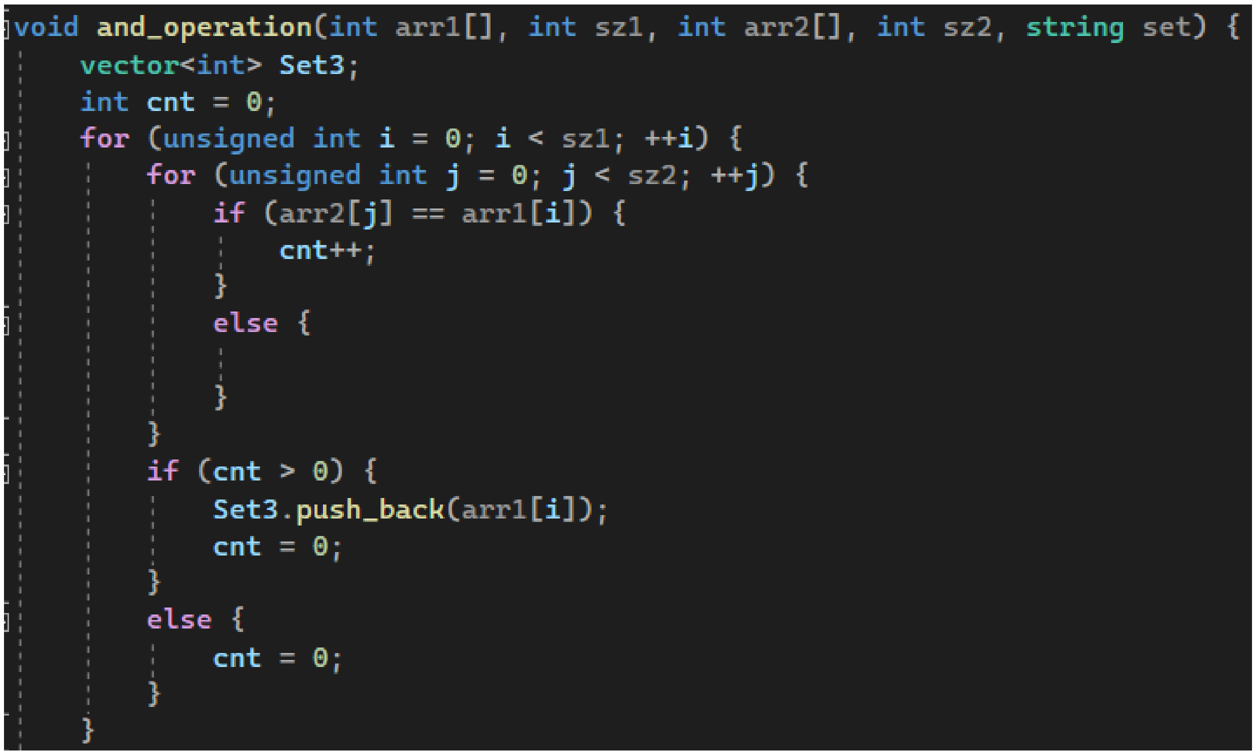


```
//Convert vectors to arrays
int* arr1 = &Set1[0];
int* arr2 = &Set2[0];
```

For the unison U operation, a vector is created. It is populated with the numbers from arr1 and arr2. Duplicate values are removed from the vector. It is arranged in ascending order. Of course, in the GPU version, the vector is first sent to the device memory and then sorted. Finally, the new vector is converted to an array and the new set is printed.

The implementation of the intersection, relative, and symmetric operation are similar to the unison operation. However, the intersection and relative operation do not require sorting. The reason is that since both sets are already sorted in order, unnecessarily sorting again would slow down the program. The following **Figure 6**, demonstrates how the intersection is found between both sets. Moreover, the newly created vector Set3 will contain the intersection values of Set1 (arr1) and Set2 (arr2).

Figure 6

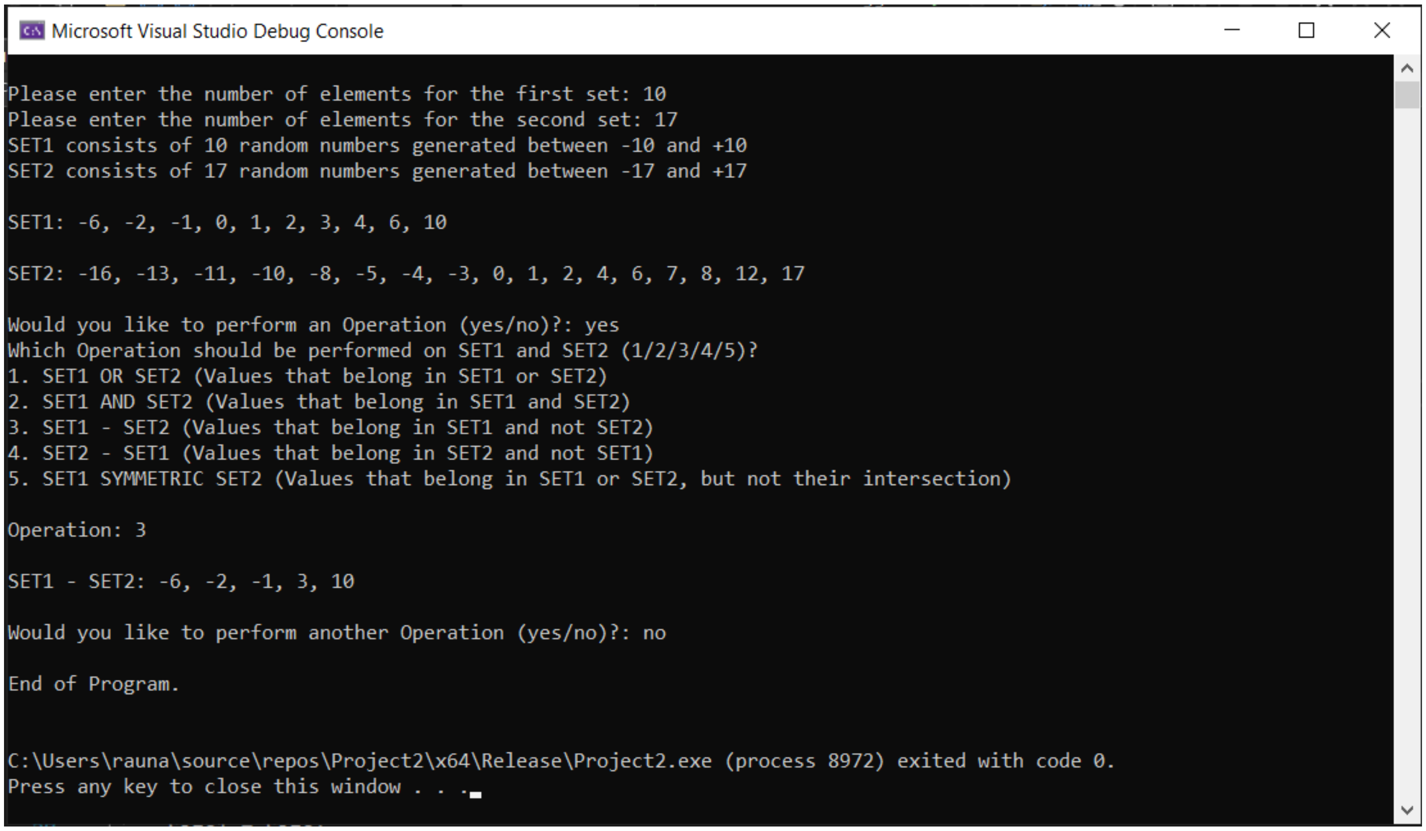


```
void and_operation(int arr1[], int sz1, int arr2[], int sz2, string set) {
    vector<int> Set3;
    int cnt = 0;
    for (unsigned int i = 0; i < sz1; ++i) {
        for (unsigned int j = 0; j < sz2; ++j) {
            if (arr2[j] == arr1[i]) {
                cnt++;
            }
            else {

            }
        }
        if (cnt > 0) {
            Set3.push_back(arr1[i]);
            cnt = 0;
        }
        else {
            cnt = 0;
        }
    }
```

How to run the Application

Figure 7 demonstrates the application of the program. User enters the size for both sets. In this case, since the size for SET1 is 10, the set will contain 10 random numbers ranging between -10 and +10. Likewise, this also applies to SET2 with size 17. Note that both sets are placed in ascending order. The user can then choose which of the five operations to apply. In this case, the relative operation SET1 - SET2 will return all the numbers unique to SET1. Finally, the user can choose to apply another operation or end the program.

Figure 7



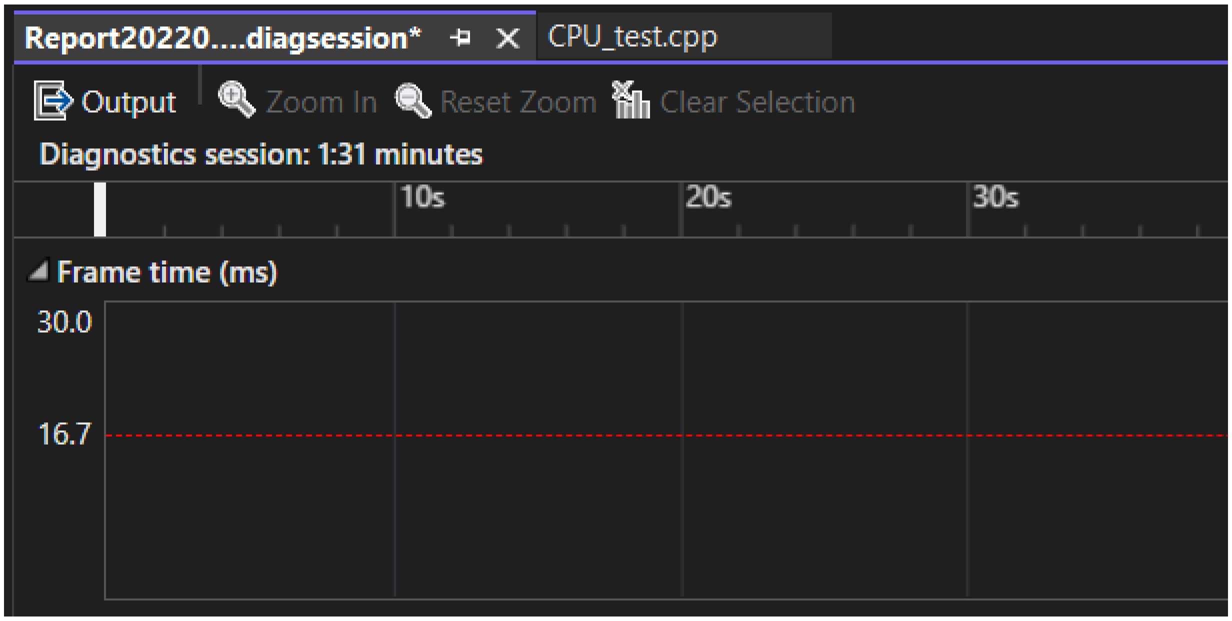
```
Please enter the number of elements for the first set: 10
Please enter the number of elements for the second set: 17
SET1 consists of 10 random numbers generated between -10 and +10
SET2 consists of 17 random numbers generated between -17 and +17

SET1: -6, -2, -1, 0, 1, 2, 3, 4, 6, 10

SET2: -16, -13, -11, -10, -8, -5, -4, -3, 0, 1, 2, 4, 6, 7, 8, 12, 17

Would you like to perform an Operation (yes/no)?: yes
Which Operation should be performed on SET1 and SET2 (1/2/3/4/5)?
1. SET1 OR SET2 (Values that belong in SET1 or SET2)
2. SET1 AND SET2 (Values that belong in SET1 and SET2)
3. SET1 - SET2 (Values that belong in SET1 and not SET2)
4. SET2 - SET1 (Values that belong in SET2 and not SET1)
5. SET1 SYMMETRIC SET2 (Values that belong in SET1 or SET2, but not their intersection)

Operation: 3

SET1 - SET2: -6, -2, -1, 3, 10

Would you like to perform another Operation (yes/no)?: no

End of Program.


C:\Users\rauna\source\repos\Project2\x64\Release\Project2.exe (process 8972) exited with code 0.
Press any key to close this window . . .
```

Results

To determine the speed of the program with and without the GPU, the performance profiler in Visual Studio is used. Both versions create two sets with size 5000. Then the sets are randomized with no duplicate values and arranged in ascending order. Finally, all five operations are performed on the sets (i.e. unison, intersection, relative twice, and symmetric). As shown in **Figure 8**, the normal or CPU version of the program is very slow. On the other hand, **Figure 9** shows that the program is significantly faster when utilizing the thrust library or GPU.

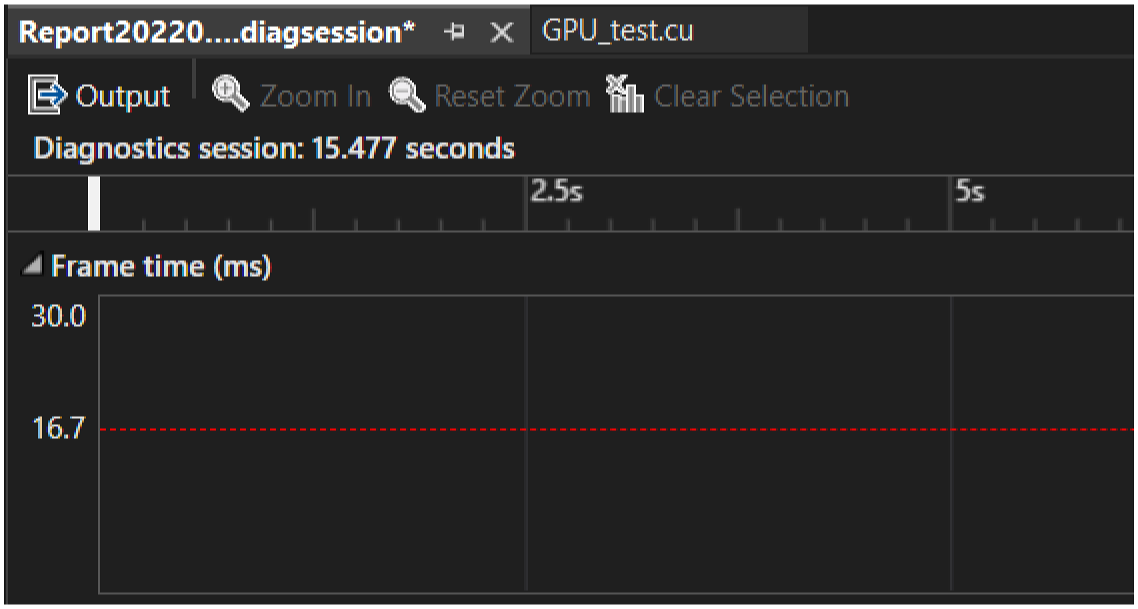
To emphasize the speed difference, not utilizing the GPU as shown in **Figure 8** when both Sets at size of 5000 elements it took more than 1 minute to complete. However, as shown in **Figure 10**, when utilizing the GPU with both Sets at size of 20,000 elements it took approximately 34.28 seconds to complete.

Figure 8



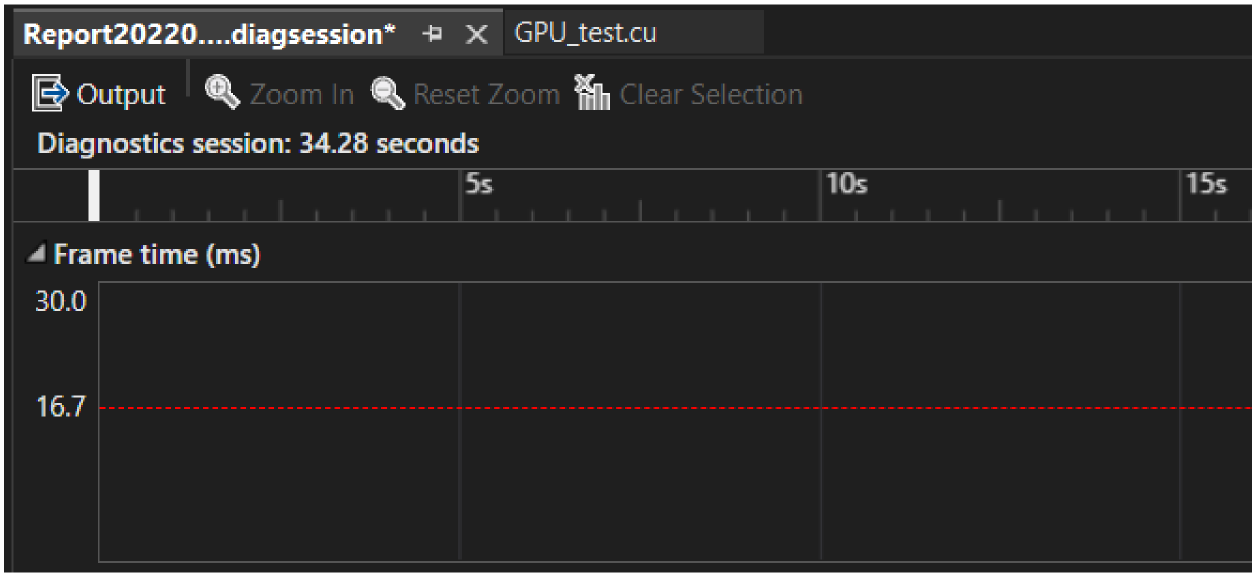
Diagnostics session: 1:31 minutes

Figure 9



Diagnostics session: 15.477 seconds

Figure 10



Diagnostics session: 34.28 seconds

Challenges

- Randomize the Sets
    - To ensure there is never a duplicate value in a set, the vector needs to be iterated based on the size. For example, since the following set has 10 elements, there will be 10 separate iterations to ensure no duplicate values are present.

      Initial Set: {-9, 3, 7, 13, 0, -52, -44, 7, -2, 3} n = 10

      After n iteration: {-9, 3, 7, 13, 0, -52, -44, 0, -2, -52}

      …………..

      After n - 1 iteration: {-9, 3, 7, 13, 0, -52, -44, 1, -2, -5}

      Note: Set is not yet sorted

- Sending the Sets from Host to Device memory
    - Instead of having to use cudaMalloc, the thrust library solves this issue by using the "device_vector." This sends the host vector to the GPU, and does all of the memory allocation 'under the hood.'

Contribution

As shown in **Table 1**, the entire project is created by Raunac Bhuiyan. Though it is important to acknowledge that feedback from the Teacher Assistant (TA) Nafis Mustakin was very helpful. In fact, the TA recommended the use of the Thrust library.

Table 1

Task	Breakdown
Program Implementation	Raunac Bhuiyan - 100%
Project Report	Raunac Bhuiyan - 100%

References

1. Thrust CUDA Toolkit: https://docs.nvidia.com/cuda/thrust/index.html#abstract
2. Video Demonstration: https://www.youtube.com/watch?v=mYciZDzV794

*Note: The video is also linked in the Github final-project-sort repository.