Operating Systems

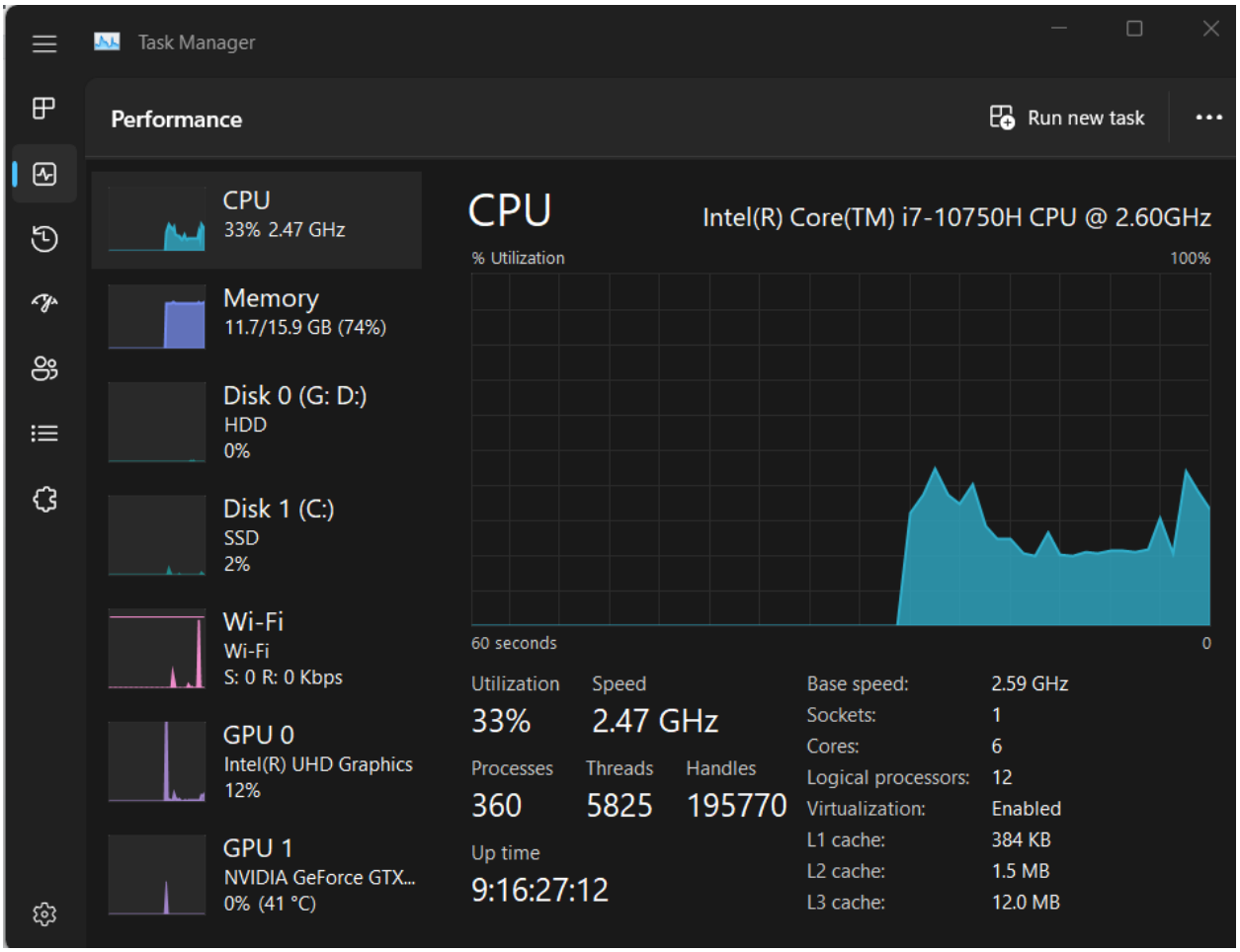
Assignment 3

Syeda Farwa Rizvi
200901098
CS-01-A

Check processor cores of system. For n cores create n threads & divide array among these threads, sort them using merge sort. Take the size of array as input.
Submit:

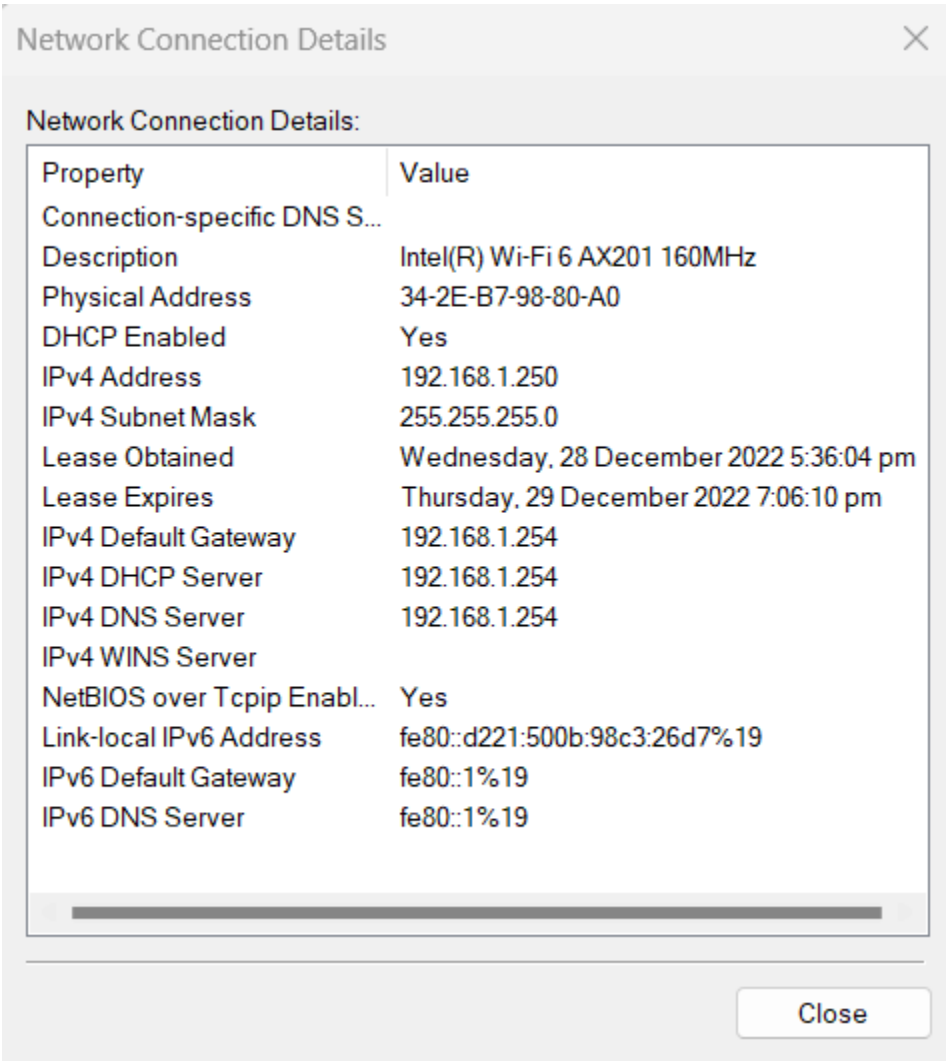
1. multi threaded mergesort cpp file
2. Screenshot of available cores
3. Screenshot of MAC addresses of system
4. Reference link of merge sort code
5. Git hub link

Number of Cores



Using 6 Cores as shown above.

MAC Address



Physical Address: 34-2E-B7-98-80-A0

Code:

- Reference of Merge Sort and Merge functions: https://www.programiz.com/dsa/merge-sort

```
#include <iostream>
#include <thread>
#include <vector>
#include <pthread.h>
using namespace std;
```

```
//                          MERGE SORTING
// merge two subarrays L and M into arr
void merge(int* arr, int start, int limit, int end)
{
// sizes of the two subarrays to be merged
// create L <- A[start...limit] & M <- A[limit+1...end]
int size1 = limit - start + 1;
int size2 = end  - limit;

// create temp arrays
int* left = new int[size1];
int* right = new int[size2];

// copy data to temp arrays
for (int i = 0; i < size1; i++)
        left[i] = arr[start + i];
for (int i = 0; i < size2; i++)
        right[i] = arr[limit + 1 + i];

// merge the temp arrays back into arr[l..end]
//Maintain current index of sub-arrays and main array
int i, j, k;
i = 0;
j = 0;
k = start;
//until we reach either end of left or right, pick larger and place correct position at A[start...limit]
while (i < size1 && j < size2)
{
        if (left[i] <= right[j])
        {
        arr[k] = left[i];
        i++;
        }
        else
        {
```

```
        arr[k] = right[j];
        j++;
        }
        k++;
}

// copy remaining elements of left[], if any and put in A[start...end]
while (i < size1)
{
        arr[k] = left[i];
        i++;
        k++;
}

// copy remaining elements of right[], if any
while (j < size2)
{
        arr[k] = right[j];
        j++;
        k++;
}
}
// Merge sort
void mergeSort(int* array, int start, int end)
{
 if (start < end)
  //limit is the point where the array is divided into two subarrays
 {
        // find the middle point
        int limit = start + (end - start) / 2;
        // sort the two halves
        mergeSort(array, start, limit);
        mergeSort(array, limit + 1, end);
        // merge the sorted subarrays
        merge(array, start, limit, end);
 }
```

```
}
//                              MULTITHREADING
// divides the array into six equal parts & merge sort
void multithreaded(int* array, int total_size)
{
 // to get the size of every part that is used
 int partSize = total_size / 6;
 //six threads created
 thread t1(mergeSort, array, 0, partSize - 1);
 thread t2(mergeSort, array, partSize, 2 * partSize - 1);
 thread t3(mergeSort, array, 2 * partSize, 3 * partSize - 1);
 thread t4(mergeSort, array, 3* partSize, 4 * partSize - 1);
 thread t5(mergeSort, array, 4* partSize, 5 * partSize - 1);
 thread t6(mergeSort, array, 5 * partSize, total_size - 1);

 // waiting time for completion of threads:
 t1.join();
 t2.join();
 t3.join();
 t4.join();
 t5.join();
 t6.join();
// displaying sorted array
cout << "Sorted array: ";
for (int i = 0; i < total_size; i++)
cout << array[i] << " ";

}

//                              MAIN
int main()
{
 // take array size as input from user
 int sizeArray;
 cout << "Enter size of array here:  ";
 cin >> sizeArray;
```

```
 int* array = new int;
 for (int i=0;i<sizeArray; i++){
   cout<<" for "<<i<<" enter: ";
   cin>>array[i];
 }
multithreaded(array, sizeArray);

 return 0;
}
```