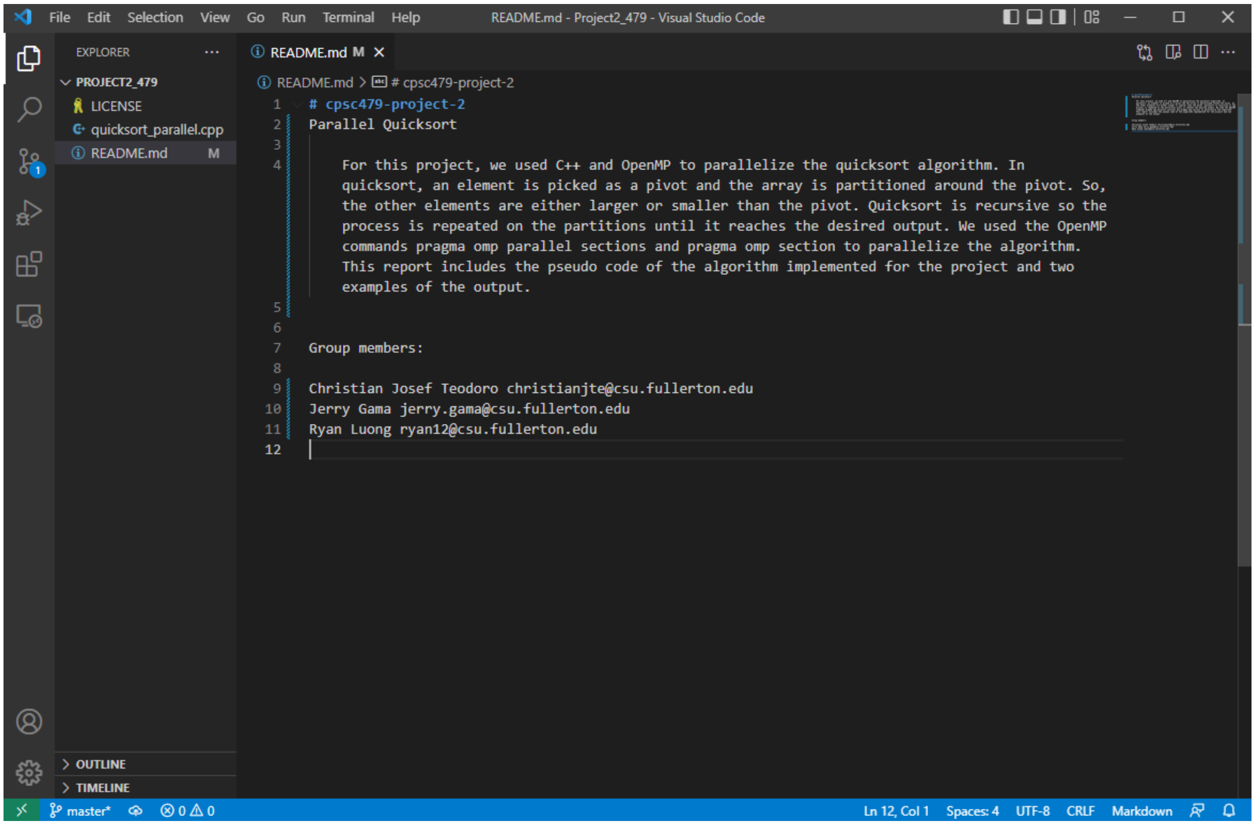
Project 2

Group Members:

Christian Josef Teodoro christianjte@csu.fullerton.edu
Jerry Gama jerry.gama@csu.fullerton.edu
Ryan Luong ryan12@csu.fullerton.edu



Summary:

For this project, we used C++ and OpenMP to parallelize the quicksort algorithm. In quicksort, an element is picked as a pivot and the array is partitioned around the pivot. So, the other elements are either larger or smaller than the pivot. Quicksort is recursive so the process is repeated on the partitions until it reaches the desired output. We used the OpenMP commands pragma omp parallel sections and pragma omp section to parallelize the algorithm. This report includes the pseudo code of the algorithm implemented for the project and two examples of the output.

Pseudocode:

```
elaspedTime = 0

int partition(array, first, last) {

    pivot = first array element

    count = 0

    for (i = first + 1; i <= last; i++) {
```

```
        if array[i] <= pivot
            add 1 to count
    }
    pivotIndex = first + count
    swap array[pivotIndex] and arr[last]
    i = first, j = last
    while i is less than pivotIndex and j is greater than pivotIndex {
        while array[i] is less than or equal to pivot
            add 1 to i
        while array[j] is greater than pivot
            subtract 1 to j
        if i is less than pivotIndex and j is greater than pivotIndex
            swap array[i] and array[j]
            add 1 to i
            subtract 1 to j
}
    return pivotIndex
}
void quickSort(array, first, last) {
    if first < last {
        p = partition(array, first, last)
        Start parallelizing {
            parallelize this block of code
            begin, end
            start timing (begin)
            quicksort(array, first, p - 1)
            end timing (end)
            elaspedTime += end - begin
}
```

```
      parallelize this block of code {
      begin, end
      start timing (begin)
      quicksort(array, p + 1, last)
      end timing (end)
      elaspedTime += end - begin
}
}
}
}
main function () {
  arraySize = 0
  randomize seed
  ask for arraySize
  initialize array based on arraySize
  for i to arrarySize {
    rando = random number between 1 to arraySize
    array[i] = rando
}
  quickSort(array, 0, n-1)
  if elapsedTime is less than 60 minutes {
    print the array elements
    print the elapsed time
}
  else
    end the program
}
}
```

The code was compiled and run using:

g++ -fopenmp quicksort_parallel.cpp -o quicksort

./quicksort

n = 100,000

```
99959
99961
99961
99961
99961
99962
99962
99962
99963
99964
99965
99965
99966
99967
99967
99968
99970
99972
99972
99973
99973
99973
99975
99975
99977
99979
99979
99980
99981
99981
99982
99983
99984
99985
99985
99985
99986
99987
99988
99989
99992
99992
99992
99996
99997
99998
99998
99998
100000
Elapsed time: 1.284895student@tuffix-vm:~/project2$
```

n = 1m

```
999958
999958
999958
999959
999959
999961
999961
999961
999961
999962
999962
999962
999963
999963
999964
999965
999967
999968
999969
999969
999971
999971
999971
999973
999974
999975
999975
999977
999977
999978
999978
999981
999982
999982
999985
999985
999987
999987
999989
999990
999991
999992
999992
999993
999994
999995
999998
999999
1000000
Elapsed time: 11.998142student@tuffix-vm:~/project2$
```