Outlook

Re: SE3082 – Assignment Proposal

From Nuwan Kodagoda <nuwan.k@sliit.lk>
Date Mon 11/3/2025 11:55 AM
To SENEVIRATNE D.M.D.G.B it23226128 <it23226128@my.sliit.lk>

[EXTERNAL EMAIL] ***This email has been received from an external source – please review before actioning, clicking on links, or opening attachments.***

Approved. Please proceed.

Best Regards

Nuwan

From: SENEVIRATNE D.M.D.G.B it23226128 <it23226128@my.sliit.lk>
Date: Monday, 3 November 2025 at 11:48 am
To: Nuwan Kodagoda <nuwan.k@sliit.lk>
Subject: Re: SE3082 – Assignment Proposal

Dear Sir,

Thank you for your feedback and suggestions. Based on your recommendation, I would like to propose Quick Sort as my algorithm.

a) Title of the Algorithm - Parallel Quick Sort

b) Problem Domain - Sorting and Searching Algorithms

c) Brief Description of the Algorithm

Quick Sort is a divide-and-conquer sorting algorithm that works by selecting a pivot element from the array and partitioning the other elements into two groups: elements smaller than the pivot and elements larger than the pivot. The algorithm then recursively sorts these two groups. Quick Sort has an average time complexity of $O(n \log n)$ but can reach $O(n^2)$ in the worst case if poor pivots are chosen.

Quick Sort is well-suited for parallelization because after partitioning, the two sub-arrays are completely independent and can be sorted simultaneously. Unlike merge sort where the main work happens during merging, Quick Sort does most of its work during partitioning, which creates natural parallel tasks. Each recursive call on sub-arrays can run on different processors without interfering with each other.

The algorithm offers good parallelization opportunities for all three technologies. With OpenMP, we can use task-based parallelism where each recursive call becomes a separate task that threads can pick up. With MPI, we can distribute sub-arrays to different processes after partitioning, though this requires careful data distribution. With CUDA, we can parallelize the partitioning step itself using GPU threads to compare elements with the pivot simultaneously.

The main challenges include load balancing, as sub-arrays may have unequal sizes after partitioning, and the sequential nature of the partitioning step. Choosing good pivot selection strategies like median-of-three or random pivots helps avoid worst-case performance. Quick Sort generally uses less memory than merge sort since it sorts in-place, making it practical for large datasets in parallel environments.

d) Detailed Pseudocode

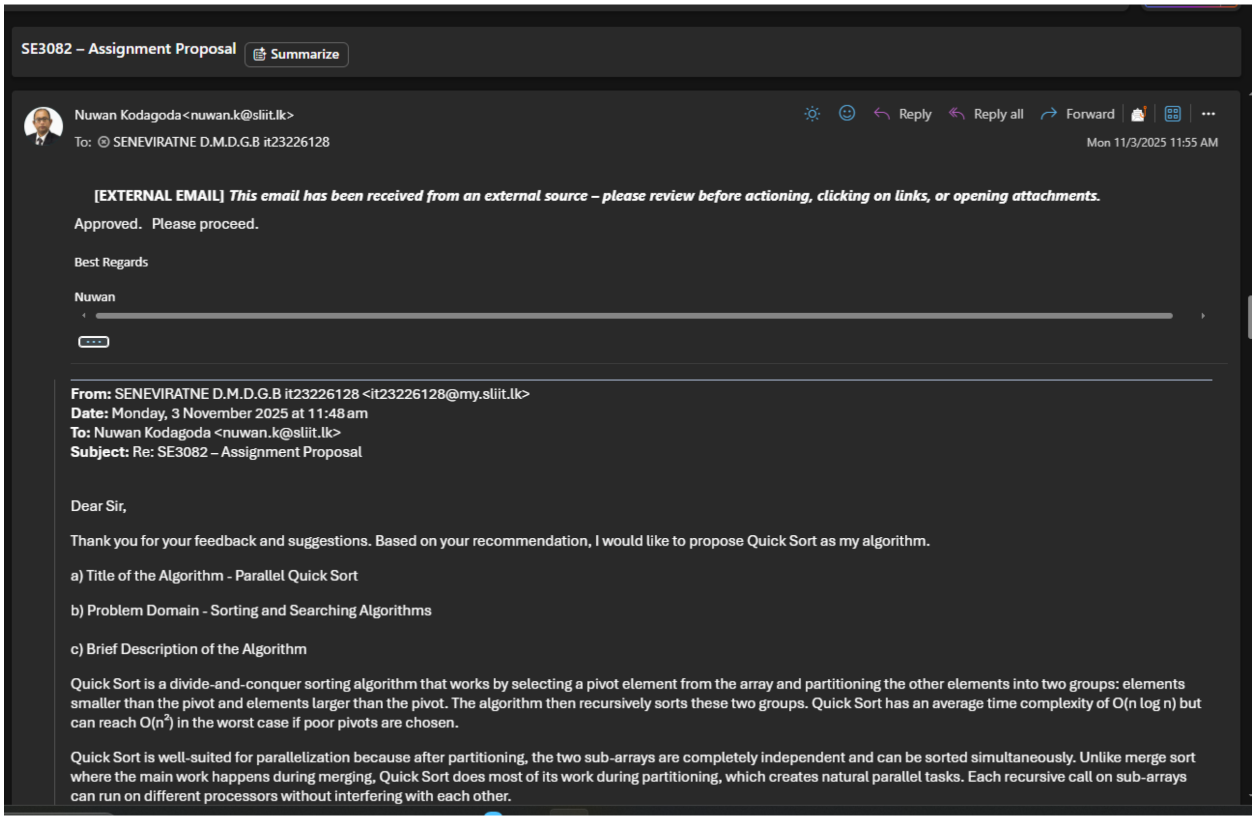
ALGORITHM: Serial Quick Sort

```
FUNCTION quickSort(array, low, high)
  IF low < high THEN
    // Partition the array and get pivot position
    pivotIndex = partition(array, low, high)

    // Recursively sort left sub-array
    quickSort(array, low, pivotIndex - 1)

    // Recursively sort right sub-array
    quickSort(array, pivotIndex + 1, high)
  END IF
END FUNCTION

FUNCTION partition(array, low, high)
  // Choose last element as pivot
  pivot = array[high]
  i = low - 1

  // Rearrange elements around pivot
  FOR j = low TO high - 1 DO
    IF array[j] <= pivot THEN
      i = i + 1
      SWAP array[i] with array[j]
    END IF
  END FOR

  // Place pivot in correct position
  SWAP array[i + 1] with array[high]
  RETURN i + 1
END FUNCTION

MAIN PROGRAM
  Get array size from user
  Create array with random numbers

  Start timer
  Call quickSort(array, 0, size - 1)
  Stop timer

  Print execution time
  Check if array is sorted correctly
END MAIN
```

Thank you for your time and consideration. I look forward to your approval to proceed with this algorithm.

Best regards,

D.M.D.G.B.Seneviratne
IT23226128
BSc (Hons) in Computer Science - Year 3 Semester 1



Nuwan Kodagoda<nuwan.k@sliit.lk>
To: SENEVIRATNE D.M.D.G.B it23226128
Mon 11/3/2025 11:55 AM

[EXTERNAL EMAIL] ***This email has been received from an external source – please review before actioning, clicking on links, or opening attachments.***

Approved. Please proceed.

Best Regards

Nuwan

From: SENEVIRATNE D.M.D.G.B it23226128 <it23226128@my.sliit.lk>
Date: Monday, 3 November 2025 at 11:48 am
To: Nuwan Kodagoda <nuwan.k@sliit.lk>
Subject: Re: SE3082 – Assignment Proposal

Dear Sir,

Thank you for your feedback and suggestions. Based on your recommendation, I would like to propose Quick Sort as my algorithm.

a) Title of the Algorithm - Parallel Quick Sort

b) Problem Domain - Sorting and Searching Algorithms

c) Brief Description of the Algorithm

Quick Sort is a divide-and-conquer sorting algorithm that works by selecting a pivot element from the array and partitioning the other elements into two groups: elements smaller than the pivot and elements larger than the pivot. The algorithm then recursively sorts these two groups. Quick Sort has an average time complexity of O(n log n) but can reach O(n^2) in the worst case if poor pivots are chosen.

Quick Sort is well-suited for parallelization because after partitioning, the two sub-arrays are completely independent and can be sorted simultaneously. Unlike merge sort where the main work happens during merging, Quick Sort does most of its work during partitioning, which creates natural parallel tasks. Each recursive call on sub-arrays can run on different processors without interfering with each other.