Operating System-2
Assignment 1
Parallel Sorting using Multi Threading

Submission Date: 1st February 2021, 9:00 PM

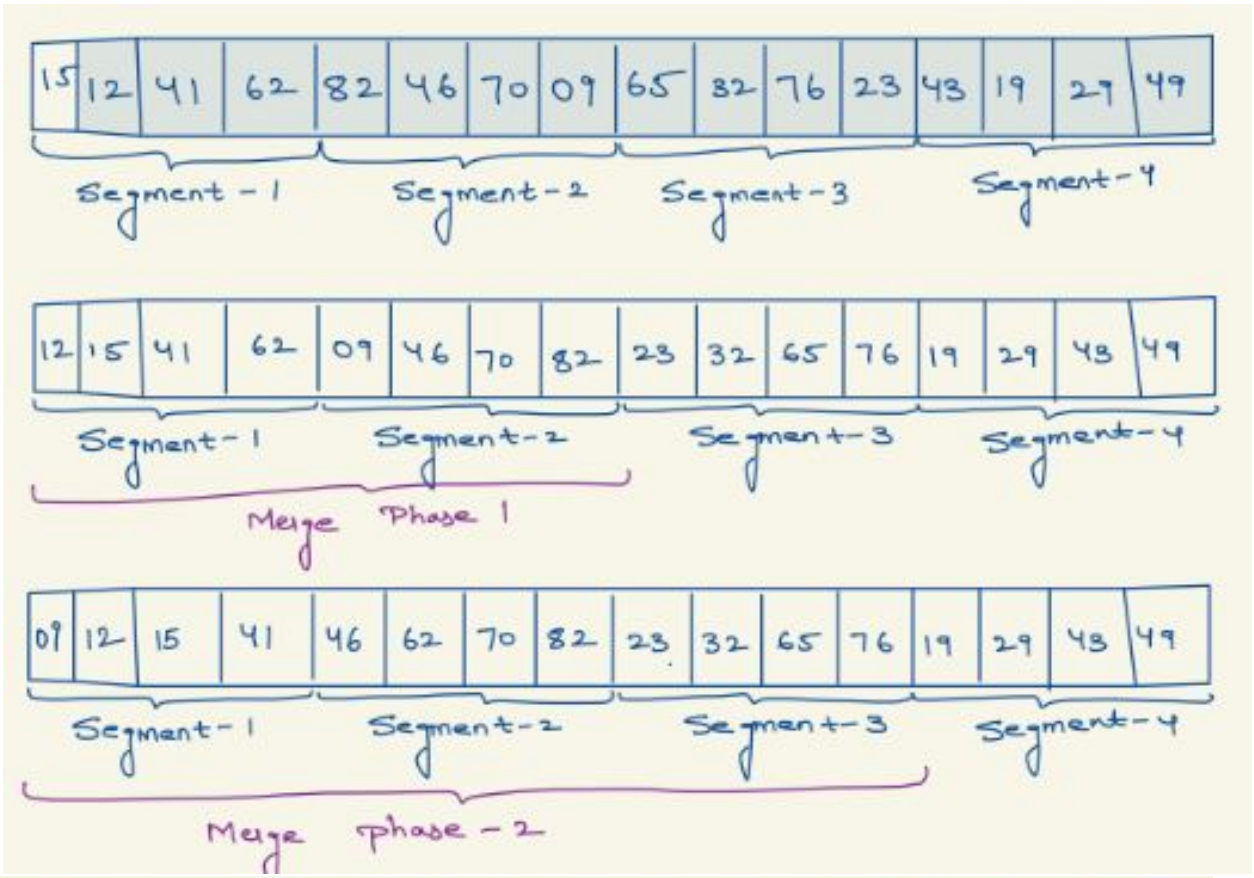
Objective: The objective of this assignment is to perform sorting of numbers efficiently using multiple threads using different s.

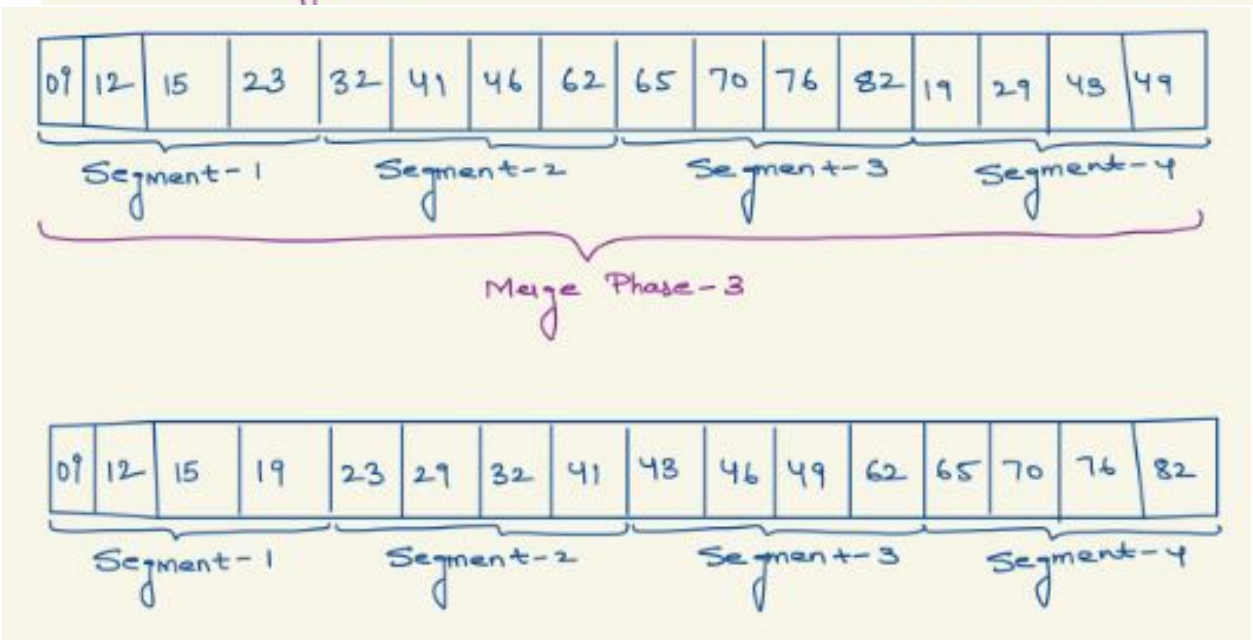
Sorting Technique: In this assignment, you will experiment with multi-threading the performance impact using real-time measurements. Given an array of size 2^n of random ~~double~~ long values. You have to divide the array into 2^p segments of equal size, 2^{p-n}. Sort each segment using a sorting algorithm S of your choice using 2^p (slave threads) threads. You must then merge each of the 2^p segments. **You have to sort and perform the merge using following two methods:**

Method 1: In this method once all the threads have performed the sorting (using S) and then exit. The main thread will take the burden of sorting them. Specifically, the steps performed by the main thread and the slave threads created by the main thread are as follows:

1. The main thread will create slave threads as the number of segments. These slave thread will sort the segments. After sorting the slave threads will exit.
2. The main thread will then merge the segments one by one in phases. In the first phase, the main thread will merge segment one and segment two and so on.
3. The main thread ensures that the resulting merged segment is also sorted. Note that merging two sorted segments of size $O(p), O(q)$ can be done in $O(p+q)$ time.
4. Merging & sorting process will decrement the number of segments by one. After merging, if the number of segments is greater than 1, then main thread will go to Step 3.

As you can see, the slave threads perform the sorting while the main thread merges all the segments. Further this, method does not take significant advantage of parallelism as the main thread still has to major task of merging all the segments in a sorted manner. The following figures illustrate the idea.



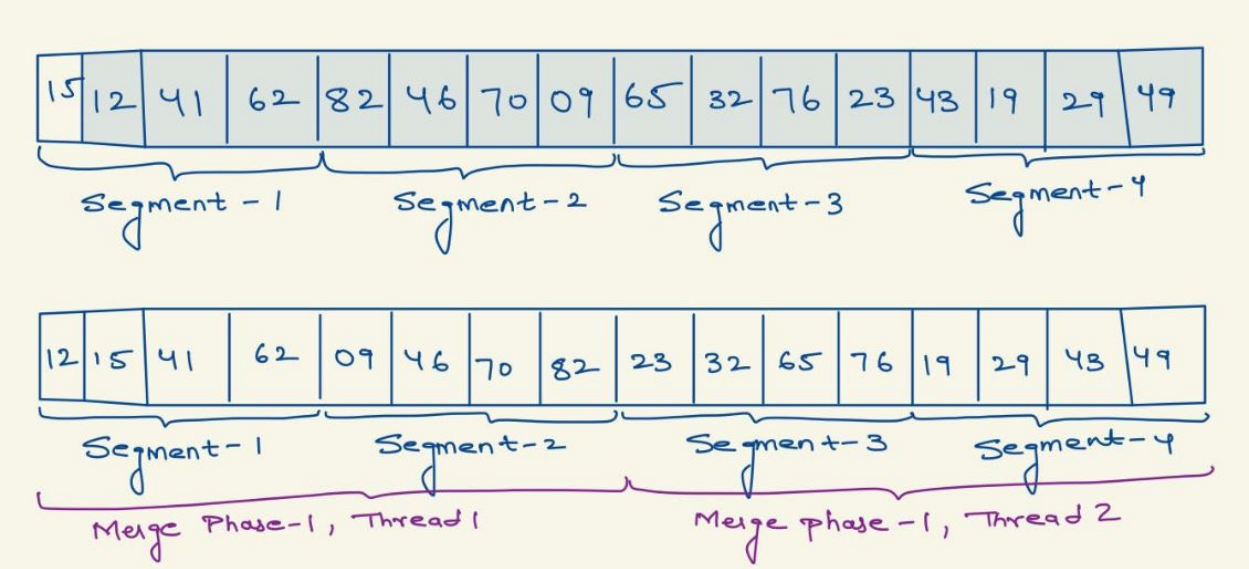


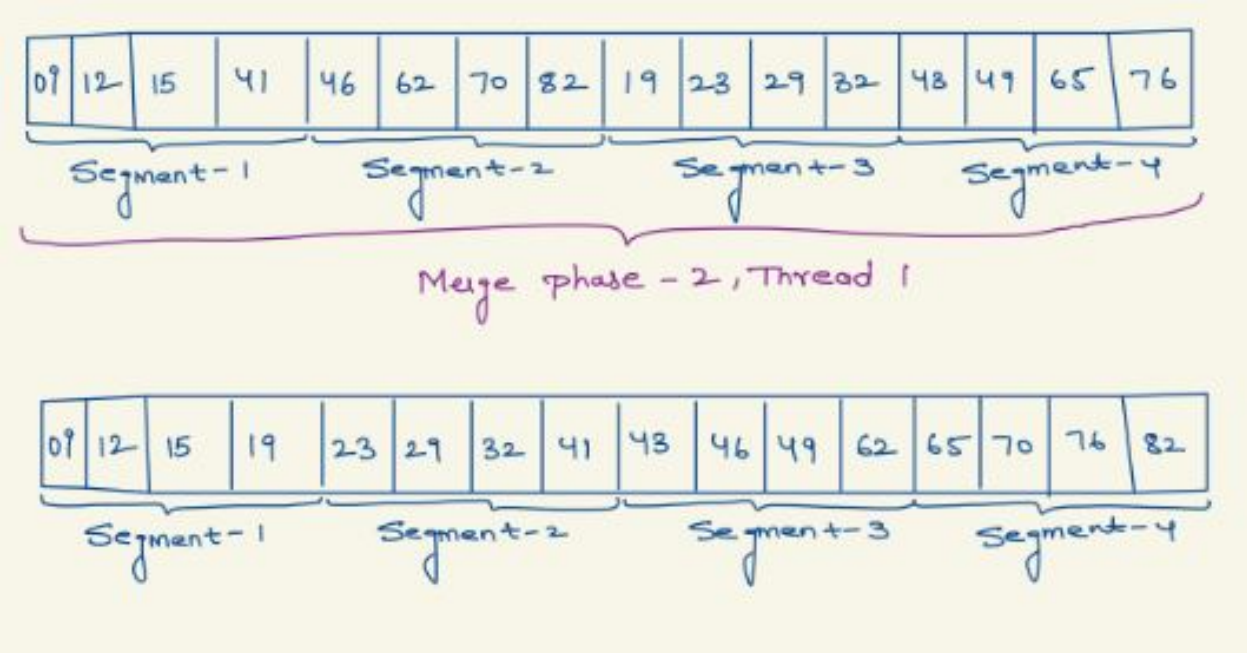
Method 2: In this method once all the slave threads have performed the sorting, they will then exit. The main thread will then again create $(k/2)$ threads to merge k segments. The steps performed by the main

thread and the slave threads are as follows:

1. The main thread will create as many slave threads as the number of segments. These slave thread will sort the segments using the sorting algorithm S. After sorting the slave threads will exit.
2. The main thread will then create another set of slave threads which are half the number of segments. Each slave thread will then merge two consecutive sorted segments instead of the main thread (unlike Method 1). Again note that merging two sorted segments of size $O(p), O(q)$ can be done in $O(p+q)$ time.
3. After merging the segments in a sorted manner, the slave threads will exit.
4. If the number of segments is greater than 1, then main thread will go to Step 2.

The following example figures illustrate the idea of merging: In first phase for merging of segment 1 and 2 thread 1 will perform the merging operation while thread 2 will perform the merging of segments 3 and 4. Again thread 1 will perform merging of segment 1,2 and 3,4. This is illustrated using the figures below.





Input: The input to the program will be a file (named inp.txt) where the first digit will be n where size of the array is 2^n. Second digit in the file will be parameter p where the number of segments are 2^p.

Example: inp.txt

5 2 // Size of Array 2^5, Number of segments 2^2

Output: Output.txt should contain the initial array before sorting and Array after sorting in the next line. Time taken in microseconds for sorting in third line.

Example: output.txt

8 15 22 13 16 27 // Array Before sorting
8 13 15 16 22 27 // Array After sorting
Time taken: 563 Microseconds.

Report: As a part of this assignment, you have to prepare a report which will describe the low-level design of your program and give an analysis of its output. As a part of the report, you have to measure the performance of the algorithms by following the above mentioned steps of 1 and 2. Calculate the time taken by method 1 and 2. To make it interesting, you can compare the time taken by the sequential algorithm S for the same input. Don't include the timing of reading and writing to file in this.

You have to create two graphs showing the time taken by these solutions for different values as follows:

- Create a graph showing the time taken by these three solutions - 1, 2 and sequential algorithms - for different values of the number of threads and fixing the size of the array to be from 2^{10} to 2^{15}. The x-axis should be the parameter p : 2, 3, 4, 5, 6 where the number of segments are equal to 2^p. The y-axis will include time taken by the program to terminate (in micro/milliseconds) for the three methods.

  Here, since the size of the array is fixed, the time taken to sort by the sequential algorithm will also be same for all the segments. Hence, the curve for the sequential algorithm will be a straight line parallel to x-axis.

- Next, create another graph showing the times using both the methods 1, 2 and the sequential algorithm while fixing the p to 4, i.e., the number of segments to be 16. The x-axis should be the parameter n, the array size, i.e. 7, 8, 9, 10, 11, 12.

  Here, the size of the array is changing. Hence, the time taken to sort by the sequential algorithm will also change.

Report should include the detailed analysis of the graphs and have a conclusion on which is faster. Note again that each graph will have three curves one corresponding to: 1, 2 and sequential execution using the algorithm S.

You have to implement all the three methods - 1, 2 and sequential algorithm. Your report should include analysis for both the Graphs.

Deliverables:

1. Report describing the low-level design of your program and analysis of output and graph. This file should be named as Assgn1_⟨Roll No⟩_Report.pdf
2. Prepare a README file that contains the instructions on how to execute your submitted file. The file should be named as Assgn1_⟨Roll No⟩_README.txt
3. Name the source code file in the following format: Asgn1_⟨Roll No⟩_mth1.c and Asgn1_⟨Roll No⟩_mth2.c
4. Upload the source code, report and README as a zip archive file and name it as Asgn1_⟨Roll No⟩.zip

Please see the instructions given above before uploading your file. Your assignment will NOT be evaluated if there is any deviation from the instructions posted there.

The policy for grading this assignment will be

- Design as described in report and analysis of the obtained results: 50%
- Execution: 40%
- Code documentation and indentation: 10%.

As mentioned before, all assignments for this course has the late submission policy of a penalty of 10% each day after the deadline.

Kindly remember that all submissions are subjected to plagiarism checks.