**Report: OS-2 Assignment1**

Name: Naitik Malav Roll No. : CS19BTECH11026

***Low-level design of Asgn1\_CS19BTECH11026\_mth1.c :***

1. At the very first I have scanned values of *n* and *p* from *inp.txt* and also declared *ArraySize=2n*, *TotalThreads=2p*.
2. initialTime and finalTime is used to calculate the time taken to sort and merge.
3. *temp* pointer of type *struct info* is used to store the address of double *Array* and value of indexes low and high.
4. Creating and joining all 2p threads using *pthread\_create* and *pthread\_join*. I have passed arguments in form of *struct info* in *threadFunc* with the help of *temp* pointer.

**Description of *threadFunc*:** I have declared low and high, which are basically the starting and ending indexes of particular segment of array which is assigned to a single thread and gets sorted by insertion sort. The value of low and high are calculated as-

1. low = thread\_part\*( ArraySize/ TotalThreads),
2. high = (thread\_part+1)\*(ArraySize/TotalThreads)-1

Using for-loop moving temp->ptr to lowth index. Using insertion sort I have sorted this particular part and set temp->ptr back to 0th index of Array using *setArrayPointer* function.

5. Now according to the method-1 described in question, I have declared pivot variable. And passing 0th, pivot and high index in merge function.

merge(temp->ptr, 0, pivot, pivot+(ArraySize/TotalThreads));

Here 0 is the lowth index, and high = pivot+(ArraySize/TotalThreads). Note that pivot is incremented by *ArraySize/TotalThreads*each time it is called.

6. Calculating the finalTime and printing sorted array along with the *Time Taken*.

Compilation Screenshot:

Graphical user interface, text

Description automatically generated

***Low-level design of Asgn1\_CS19BTECH11026\_mth2.c :***

1. Follow the step1 to step4 as described above.
2. In the method-2 once all the slave threads have performed the sorting, they will then exit. The main thread will create k threads to merge 2k segments.
3. 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.
4. After merging the segments in a sorted manner, the slave threads will exit. If the number of segments is greater than 1, then main thread will go to Step-3. With the help of basic maths and while-loop I have implemented above steps.

Note that I have declared array of type struct info *slaveTemp[k].* Every element of this array story different values of low and high, as per the question statement.

**Description of *slaveThreadFunc*:** In this function I have called merge function and passed 0, low, mid, high into it. I have calculated mid using-

mid = low+(high-low)/2

Eg: With the help of above procedure, I am passing the value of low and high to slaveThreadFunc.

1. Suppose n=4, p=3, then in this case values of low, high are: (0, 3), (4, 7), (8, 11), (12, 15)
2. After first loop, total segments gets halved so low and high are: (0, 7), (8, 15)
3. And at last, low and high: (0, 15)

Compilation Screenshot:

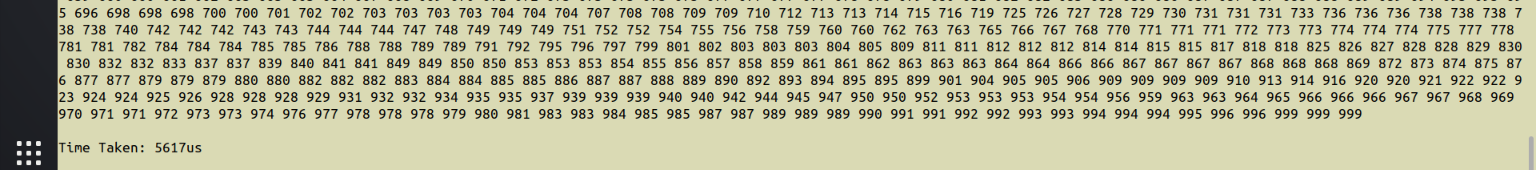
Text

Description automatically generated

Examples:

1. n=10, p=6

A screenshot of a computer

Description automatically generated with medium confidence

***Graphs:***

* In first graph I have fixed the value of ArraySize=210 and changing values of p from 2 to 6. The y-axis will include time taken by the program to terminate (in microseconds) for the three methods. The x-axis should be the parameter p 2, 3, 4, 5, 6 where the number of segments are equal to 2p.

(Grey line is the Time-Taken by Insertion sort)

Below are the graph details, each time is in microseconds. Each time is calculated by taking average of 10 output-times of same category.

|  |  |  |  |
| --- | --- | --- | --- |
| p | method1 | method2 | Insertion Sort |
| 2 | 176 | 319 | 1079 |
| 3 | 359 | 635 | 1079 |
| 4 | 576 | 1050 | 1079 |
| 5 | 1115 | 2122 | 1079 |
| 6 | 2162 | 3831 | 1079 |
|  |  |  |  |

* In second graph I have fixed the value of TotalThreads=24 i.e. p=4 and changing values of n from 7 to 12. The x-axis is the parameter n, i.e. 7, 8, 9, 10, 11, 12. The y-axis will include time taken by the program to terminate (in microseconds) for the three methods.

Below are the graph details, each time is in microseconds. Each time is calculated by taking average of 10 output-times of same category.

|  |  |  |  |
| --- | --- | --- | --- |
| n | method1 | method2 | InsertionSort |
| 7 | 833 | 1870 | 22 |
| 8 | 845 | 1904 | 35 |
| 9 | 866 | 1935 | 127 |
| 10 | 895 | 1947 | 500 |
| 11 | 1000 | 1983 | 2080 |
| 12 | 1098 | 2018 | 8085 |