1. The structure of your program

<https://github.com/MiroslavaRo/PDS_Assignment/blob/master/PDS_Part_A/Program.cs>

1. Evaluation of the tasks according to the following criteria:

* **Is this problem able to be parallelized?**

*Task 01:*

*Task 02:*

* **How would the problem be partitioned?**

*Task 01:*

An array can be divided into a number of threads to consist of sub-arrays, where the largest element of the previous sub-array is lower than the smallest element of the next sub-array. This way the sub-arrays will be sorted in ascending order and can then be merged without any changes.

Make an array of sub-lists and fill each of them by the elements that are lower than split factor\*, multiplied by the index of current sub-list.

\*Split factor equals the largest element in the entire array divided by the number of threads.

var maxNum = array.Max();

List<int>[] subLists = new List<int>[numOfthreads];

var splitFactor = maxNum / numOfthreads;

for (var j = 0; j < numOfthreads - 1; j++)

{

subLists[j] = new List<int>();

for (int i = 0; i < array.Count; i++)

{

if (array[i] <= splitFactor \* (j + 1))

{

var value = array[i];

subLists[j].Add(value);

}

}

array.RemoveAll(a => subLists[j].Contains(a));

}

subLists[subLists.Length - 1] = array;

*Task 02:*

Since the order does not matter, the array can be partitioned by element index. Make an array of sub-lists and fill each of the it until quotient from division of the element’s index in sub-list by the split factor\*, rounded down is equal to or greater than the index of the current sub-list.

\*Split factor equals the length of the entire array divided by the number of threads.

List<string>[] sublists = new List<string>[numOfthreads];

var splitFactor = Math.Ceiling(Convert.ToDouble(list.Count) / Convert.ToDouble(numOfthreads));

for (int i = 0; i < numOfthreads; i++)

{

sublists[i] = new List<string>();

}

for (int i = 0; i < list.Count; i++)

{

int j = Convert.ToInt32(Math.Floor(Convert.ToDouble(i) / Convert.ToDouble(splitFactor)));

var value = list[i];

sublists[j].Add(value);

}

* **Are communications needed?**

Task 01:

No, Communication is not needed.

Threads do not enter space of the same data to sort it and do not exchange data.

Task 02:

Yes.

Threads have to check whether the array is full before inserting a new value.

* **Are there any data dependencies?**

Task 01:

No data dependencies due to the way the array is distributed.

Task 02:

No data dependencies.

* **Are there synchronization needs?**

Task 01:

No, synchronization is not needed due to the fact that the sub-arrays are completely separated.

Task 02:

Yes, synchronization is needed because threads have to check whether the array is full before inserting a new value.

* **Will load balancing be a concern?**

Task 01:

Yes, because the array is not divided into equal parts, but based on the values of the elements.

Task02:

No, the array is divided into almost equal parts (a bit different for an odd number of threads).

1. The test results.

Average value after 10 tests.

\*For 100.000 elements | Time in millisecond

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tasks** | **Threads** | | | | |
| **1** | **2** | **3** | **4** | **6** |
| *Task 01* | 16\_800 | 55\_900 | 28\_500 | 19\_000 | 14\_100 |
| *Task 02* | 40 | 70 | 70 | 105 | 150 |





