Assignment 3: Parallel and Distributed Computing

Odd-even algorithm:

The odd even algorithm compares every 2 consecutive numbers in the array. The numbers are then swapped if first number is greater than the second. For example, if the array 9,3,1,4 was sorted using the odd even algorithm:

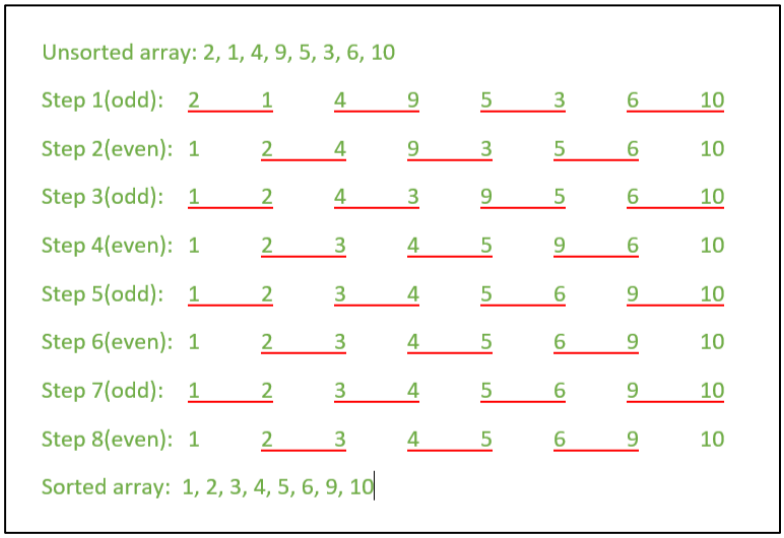
9 compared with 3
9 > 3
9 and 3 swap

1 compared with 4
1<4
Numbers remain the same
New array: 3,9,1,4

This algorithm organises the numbers into an ascending order array. The algorithm runs in two phases:

- Odd phase: Every odd element is compared with the next even element
- Even phase: Every even element is compared with the next odd element.

The below diagram illustrates this process for a larger array:



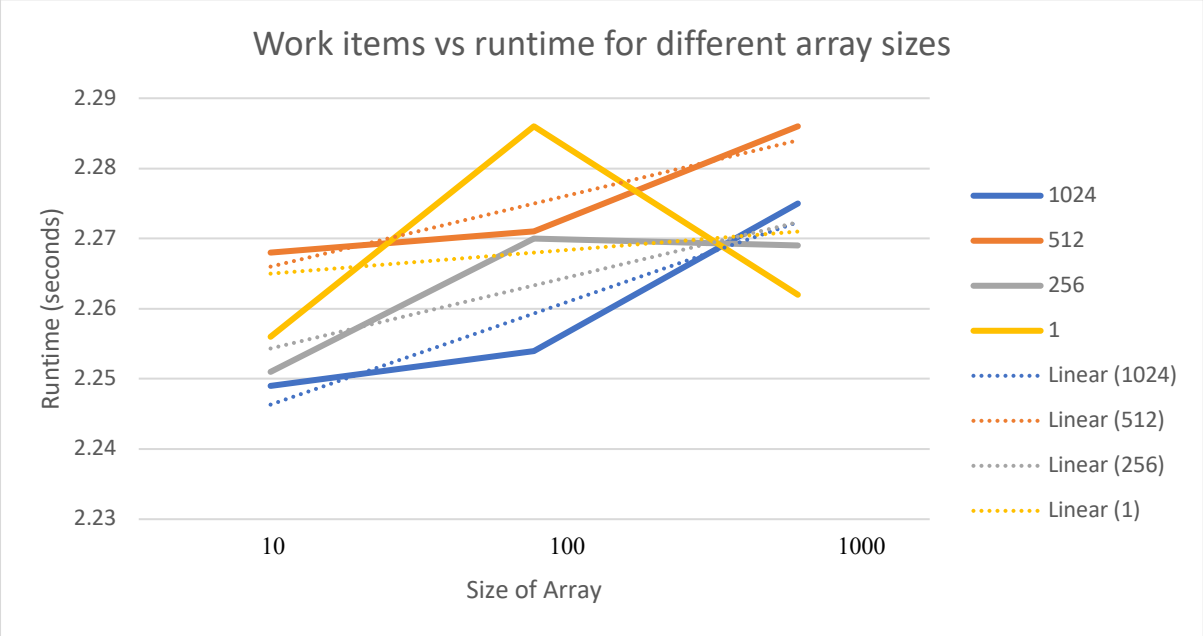
In a sequential program each step a comparison will need to be completed before the next begins. However, in a parallel program all comparisons for each step can happen at the same time, before moving onto the next step.

Table

For each work item, the average was calculated through the runtimes of 10 tests. Arrays of size 10, 100 and 1000 were tested:

Array size	Parallel work items			
	1	**256**	**512**	**1024**
10	2.256	2.251	2.286	2.249
100	2.286	2.270	2.260	2.254
1000	2.262	2.269	2.268	2.275

Graph



As we can see from the above graph the larger number of work items being used to sort an array the faster the array is sorted. I am not sure what happened when I was running my tests with 512 work items, but the data average calculated showed that these work items took the longest to complete the sorting. It was expected that 512 work items would have the second fastest runtimes after 1024 because more work items means that the array can be further spilt among processors. I believe if I had completed more tests, 100 for each set of work items, then the outlier data would be mitigated, and we would see the following rank of runtimes (ranked from fastest to slowest).

1. 1024 work items
2. 512 work items
3. 256 work items
4. 1 work item
5. Sequential