REPORT

Partition of vertices:The partition of vertices is done in order i.e the first partition contains vertices from 0 to V/(no of threads) and so on.
If there are 6 vertices and 3 threads(no of threads are equal to no of partitions)
First partition on vertices 0,1
Second partition on vertices 2,3
Third partition on vertices 4,5

Checking of boundary vertex:Boundary vertex is checked by the condition, if vertex in 1 partition and the adjacent vertex of current vertex is in different partition.Here we can easily check whether all the adjacent vertices of given vertex is in same partition or not.If in same partition then the vertex is not boundary.If not,it is boundary.

Coarse Grained lock:In this algo we use only one lock.We check whether the vertex is external or not.if internal allow to perform greedy colouring,if not wait for lock,when the lock is available perform greedy colouring and then signal the lock.

Fine Grained lock:In this algo we use array of mutexes.We check whether the vertex is internal or not.If internal allow to perform greedy colouring.If not,we will enter the adjacent vertices of current vertex and current vertex in an array.Sort them and acquire the lock. Perform greedy colouring and then release the locks in order.Here we are acquiring the locks in increasing order inorder to avoid deadlock.

Input:Input contains two values, no of threads and no of vertices.We generate an upper triangular matrix and copy the lower triangular matrix from it.A 2d matrix is intilazed to 0.here we have initialization to 0 is compulsory because for the indexes (i,i) i.e for the same vertex

we are generating an edge for the same vertex which may lead to unknown error.
We are copying the values from the random generator to matrix and then outputting to file input.txt.Since we have the matrix here,We then generate a graph using addedge function.

Above discussed algorithms we perform greedy colouring for all vertices.

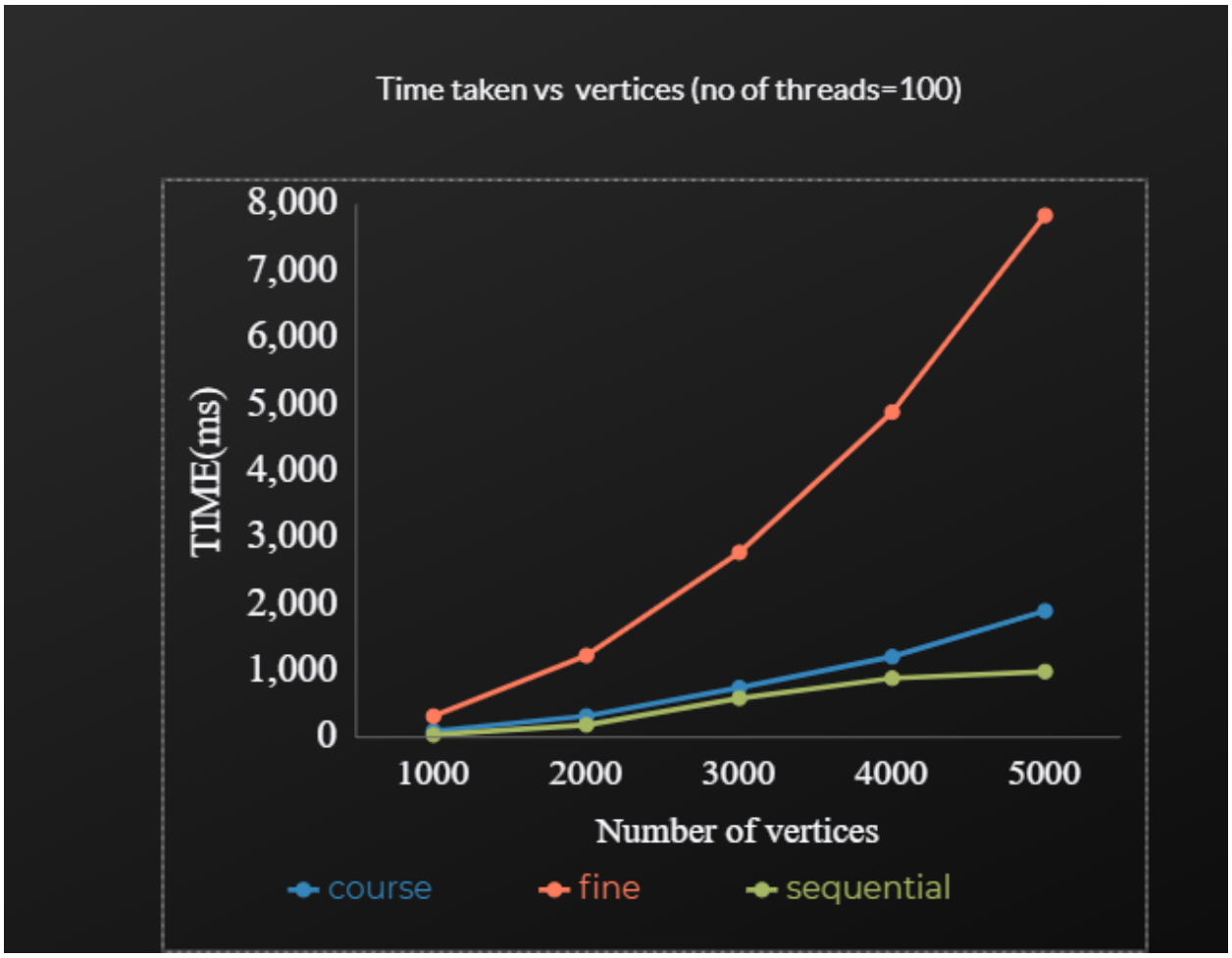
Greedy colouring:

Here We initialize the colour array to -1.except for index 0.Colour of vertex greater than or equal 0 means vertex is coloured.Less than 0 means graph is not coloured.
We then check for vertices which are coloured ,use an another array available ,initialize to true if coloured.If for any vertex for which available is false ,apply that colour to given vertex and then initialize the available to false for coloured vertices.

Time taken:Calculate the time for algorithm between creation of threads and joining the threads.

Max no of colors:Max no colours is determined by the highest value of colour array+1.

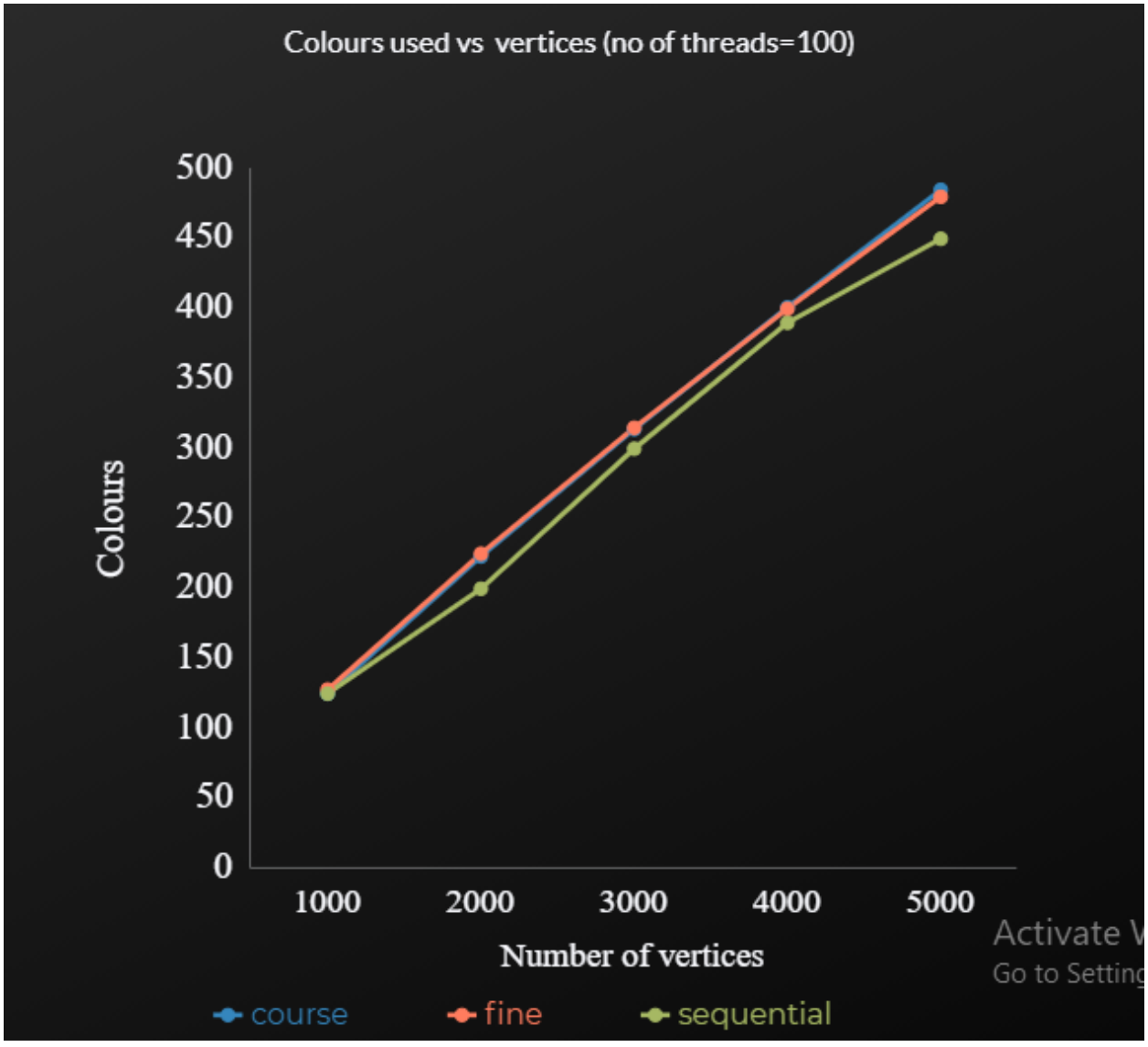
Graph1:
Time in milliseconds vs no of vertices
No of threads fixed to 100
V increase from 1000 to 5000



AS observed in graph sequential and fine grained are closely plotted whereas fine grained values are very high.

As the no of vertices increases automatically time taken to color the graph also increases.Whereas for fine grained values are very far.It is due to high overhead of locks and also if the graph is external we have to acquire locks on all adjacent vertices.It may lead to more time.

Graph2:
Max no of colours vs vertices
No of threads fixed to 100
V increase from 1000 to 5000

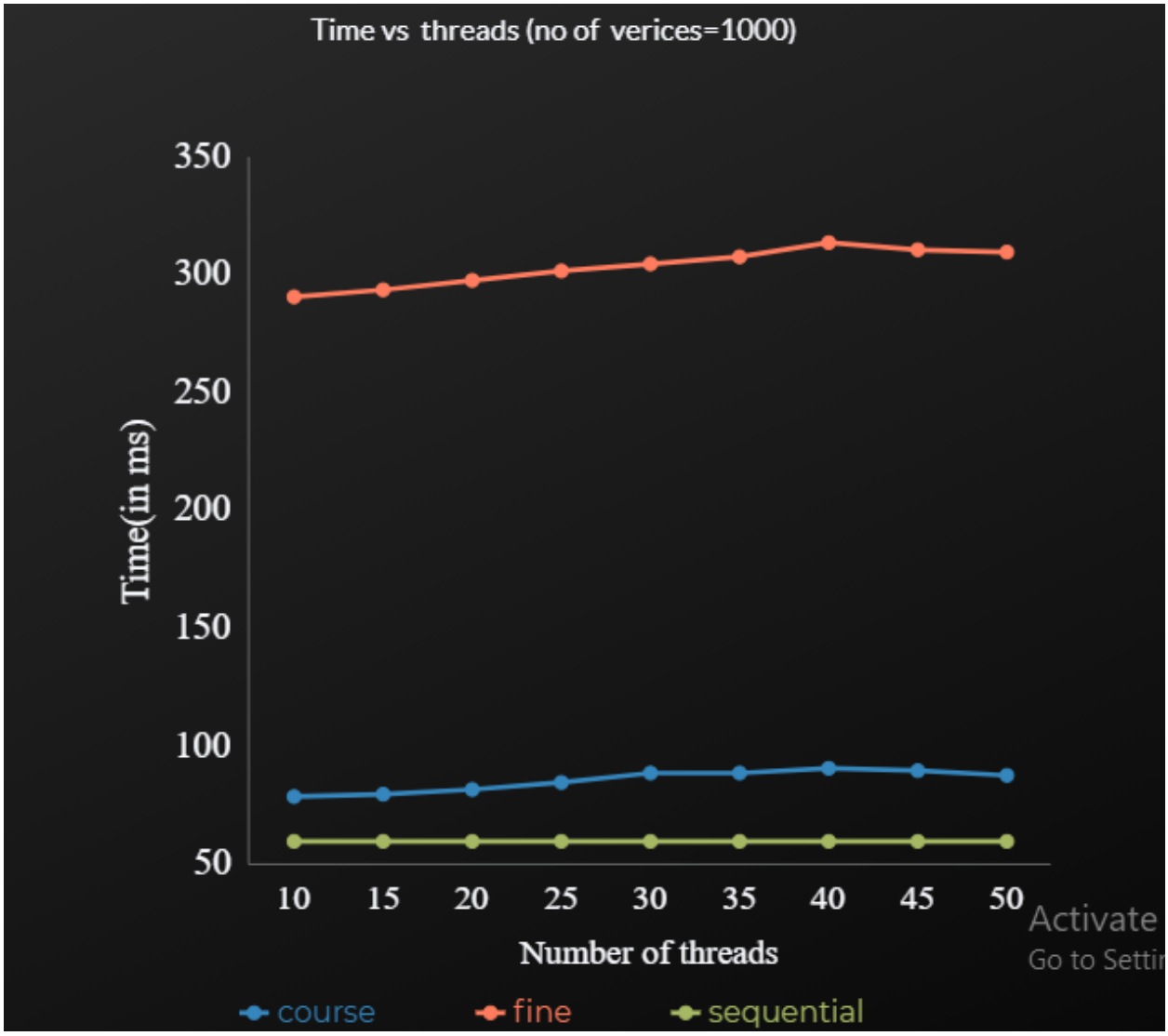


Here graphs are increasing and also closely plotted.
As vertices are increasing colours.Here sequential has lesser no of colours required because we don’t have

synchronization here and also we just see the adjacent vertices of given vertex and then colour it .

Graph3:

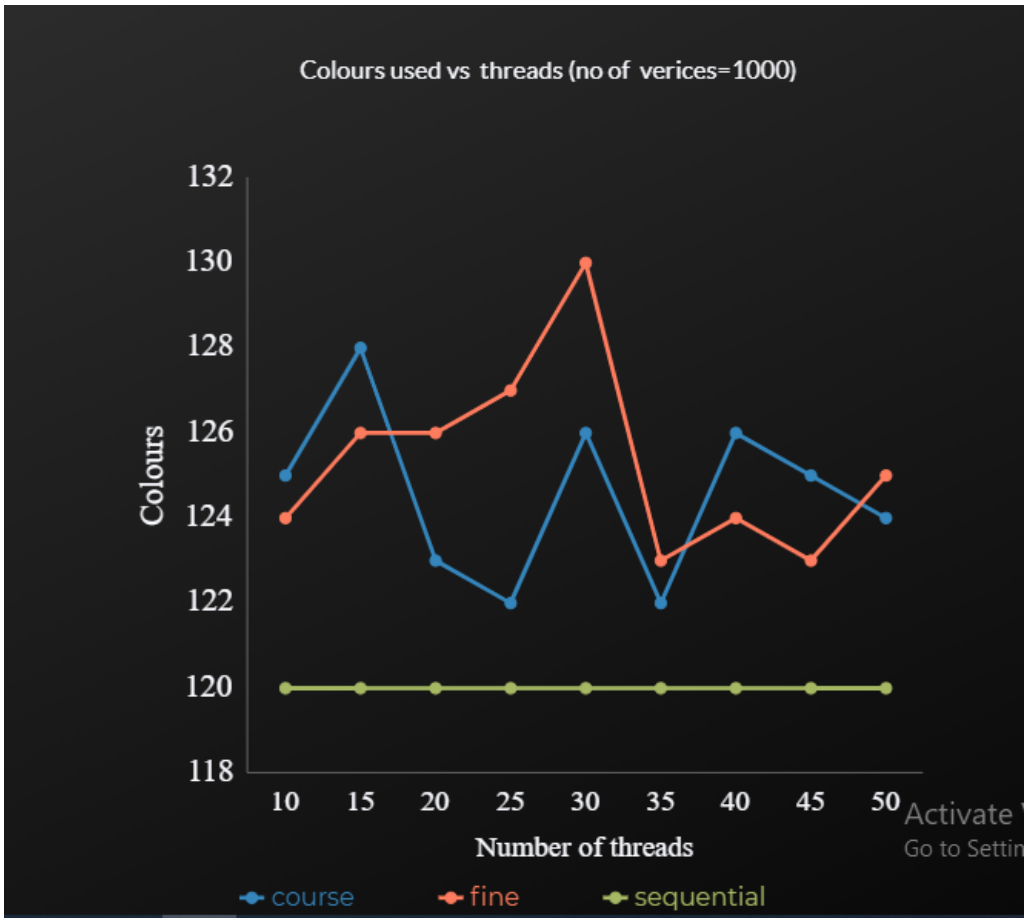
Time taken vs threads
No of Vertices fixed to 1000
Threads increase from 10 to 50



In this graph sequential is constant because it has nothing to do with threads.As compared to coarse grained fine grained has higher values as i described in above graph.
For higher values we observe a small decrease in time values because as threads increase we do have only a small partition means lesser no of vertices for the threads to calculate.Synchronization plays a major part here.

Graph4:

Colours vs threads
No of Vertices fixed to 100
Threads increase from 10 to 50



From the above graph sequential is constant because it has nothing to do with threads.Here we have mixed results as vertices are constant and we have to apply greedy colouring.It increases for some values and decreases for others.

Moreover since the graphs are randomly generated there might be chance for mixed results as the coloring is chosen by how the vertices are connected.For a connected graph we have different colour for every vertex.Since every vertex is joined to every other vertex there is only option to use different color for every vertex.