Question 1

Question 1.a

We have n task to complete and p worker. Let suppose the dependency graph in the best case allow doing p task at the same time. Then the time required will be

$$roundUp(\frac{n}{p})$$

Question 1.b

Basically we start calculating the cost of a node to be the cost of all the node where it lead to plus 1. Each node could start with a value of 1. Then we can visit all the node with the same weight (starting with the smallest first) at the same time (if p is large enough) or in multiple time

```
function schedule(G(V, E), p)
    G = reverse(G)
    while(queue does not contains all node in V)
        Node n = V.pop()
        dfs(G,n)

    List schedule = new List():
    List current = new List():
    int value = 1;
    while(queue is not empty)
        NodeWeight (n, w) = queue.pop();
        if(w != value || current.size() >= p)
            schedule.push(current);
            current.clear();
            value = w;
        current.push(n);
    schedule.push(current);
    return schedule

function dfs(G(V, E), n)
    if(queue contains e)
        return
      int weight = 1;
    ForAll(Node e in getAjacentOf(n))
           bfs(G, e)
           weight += queue.get(e).value
      queue.push(n, weight)

function reverse(G(V, E))
    Graph reverse = new Graph();
    reverse.setV(V)
    forAll(Edge e in E)
       Edge newEdge;
       newEdge.from = e.to;
       newEdge.to = e.from
       reverse.addEdge(newEdge)
    return reverse
```

This algorithm is however not optimal as we can only do node of the same weight at the same time and it might be faster to do different weight if the dependency are already completed.

Question 2

Question 2.b

The worst complexity will be were an edge could be added from all vertices to all the other. We would have $n!$ Edges. The algorithm is looping in n^2 in the vertices list and check if the edges could be added. The check edge function is of complexity k where k is the current number of edges. Then the complexity for n vertices will be

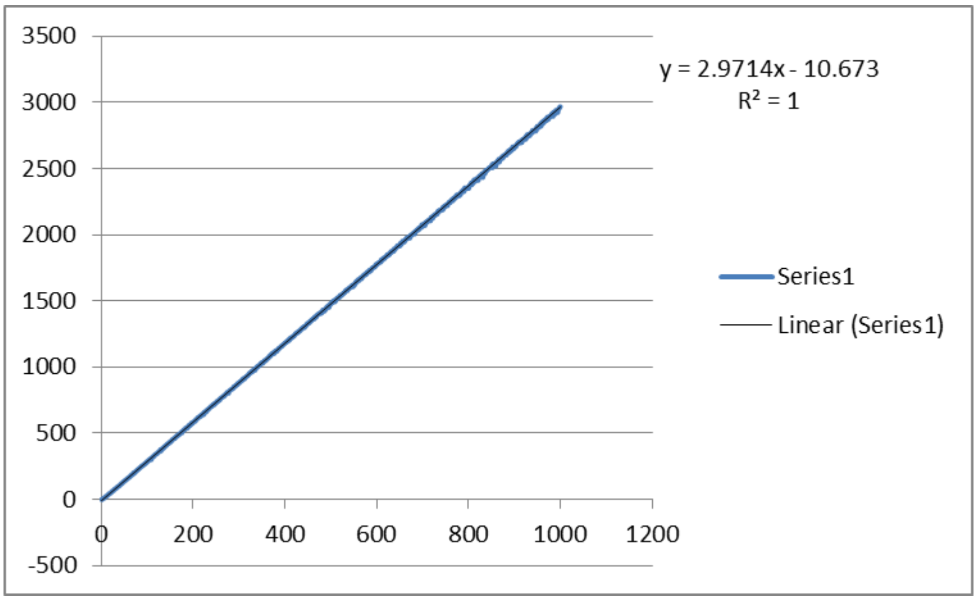
$$k * n = 1 + 2 + 3 + .. + n!$$

$$k = \frac{1}{2}(n-1)!\,(n! + 1)$$

Then the complexity is

$$n^2 * \frac{1}{2}(n-1)!\,(n!+1) = O(n^{n+2})$$

Question 2.c



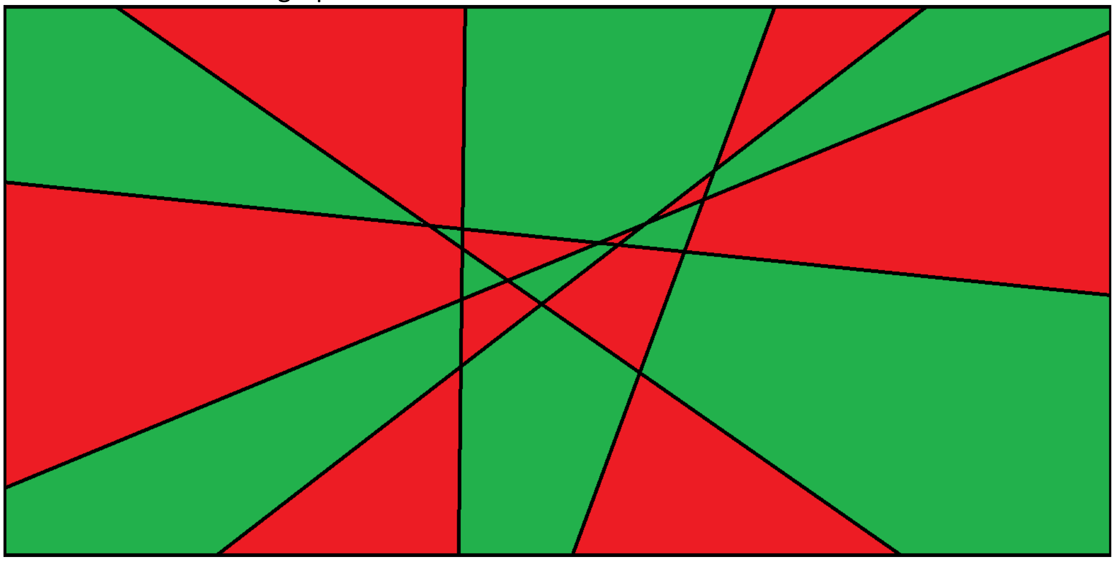
As we can see the number of edges is linear and equals to $3n$

Question 3

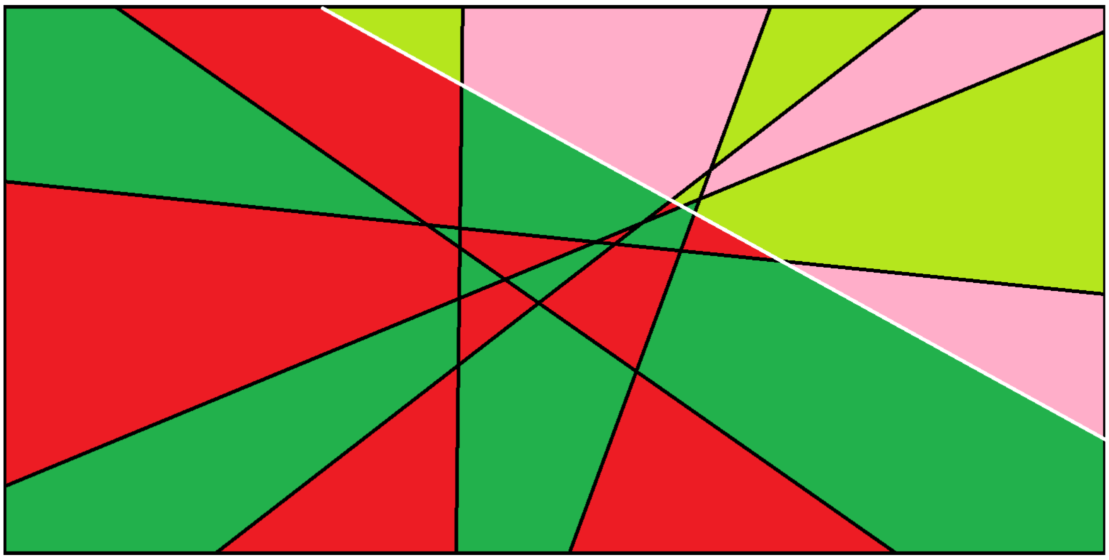
Let first prove the graph is two colourable
Base case: One line → two colors, one on each side
Induction: Let suppose we have n lines and the graph is 2-colorable.
If we add a new line we separate the graph in two parts, each part is two colourable. But as we cut some region in two → transformed in two region that touch each other and have the same colors and there are both in a different part. Then if we invert the colors of all the region in one part we still have this part is still 2-colorable and the connected region at the separation are now of different colors. Then the graph is 2-colorable.
Then as the graph is 2-colorable it's bipartite.

Example:

We have a 2-colorable graph with n lines



Let draw another line and invert all the colors on one side



And we get a two colourable graph with n+1 lines