1. 排序訂單，透過 bubble sorting，將越快到期越前面，若相同將訂單量越大越前面 (bubble sorting method)

2. 依照訂單順序分配置合適的機台，一張一張訂單分給不同的機台，分配時選擇能讓訂單時，選擇使最小化此訂單的機台延遲時間，若相同，選擇產能最低的機台

在這裡會將每筆訂單丟到 orderDistribute (function)，在orderDistribute 中會計算這筆訂單在每個機台的延遲時間和結束生產此訂單後的機器生產效率，並存在 machineMightDelay (array)和machineFinalPR (array)中，接者比較延遲時間和效能(machineFinalPR \* 理想生產量)。最後，我們將更新機器生產完的效率 (mPR)、生產完的時機點、機器累計延遲時間，並記錄此機器有幾個訂單 (mOA) 和這些訂單的編號 (mO)

3. 安排機台的維修順序，透過 bubble sorting，排出最需要維修的機台到最不需要維修的，先考慮總延遲時間最多的機台 (machineDelayTotal)，再來是訂單量 (sum) 較大的機台，最後是每小時理想產量 (a) 最大的機台

4. 宣告 workerFree，一個 h \* maxProductionTime + 10001 的矩陣，代表每個員工在某一時段是否有空，加 10001 的原因是為了避免最佳的狀況，會使的 maxProductionTime 增加而導致配置記憶體不夠

5. 宣告 repairList，一個 m \* 每台機器訂單量的矩陣，因為每台機台最多只會維修其訂單量次數，而且部會在訂單中維修，所以此矩陣代表的意義是每台機台在某一訂單開始維修前是否需要維修 (1是需要)

6. 針對每個機台，從最需要維修的機台到最不需要維修的，每個機台跑維修至無法再修，在維修次數少於等於維修人員 h 之前，不需考慮人員不足的問題，跑 determineRepair\_1；相反地，維修次數多於維修人員 h 之後，跑 determineRepair\_2

* determineRepair\_1:

此機台有 n 筆訂單，最多維修 n 次，所以跑 n 次大迴圈，其中跑小迴圈計算在哪筆訂單前維修最好，決定要維修時間後，繼續跑大迴圈決定要維修除了這筆還要維修哪一筆。

* 跑此機器的每筆訂單時，先假定訂單 j 要維修 (repairInAdvance[j] = 1，repairInAdvance 內暫時的矩陣)，利用 reCalculate 跑出延遲時間，並和最新的延遲時間比較，找出維修哪筆可以減少"最新總延遲時間"最多，如果有找到維修訂單 j 比較好 ，repairInAdvance[ repairTimePoint ] = 1，也就是說訂單 j 前要維修。最後透過 (workerFreeUpdate)，更新 workerFree，更新工作人員有空與否。
* determineRepair\_2:

和 determineRepair\_1 差別在於，需要檢查是否有否有閒置人力。利用 tryRepair\_Delay 去檢查，在 tryRepair\_Delay 中又有 freeWorker(function)，用來檢查 workerFree (array)在此維修所需的時間段，是否有人力，但是此 function 不會更動到 workerFree，同樣在最後才會透過 (workerFreeUpdate)，更新 workerFree。

* workerFreeUpdate\_1：

對應 determineRepair\_1，根據repairList進行維修更新維修人員，針對機台需要的時間段，把機台順序對應的人員的時間改成沒有空(0)

* workerFreeUpdate\_2：

對應 determineRepair\_2，根據repairList進行維修更新維修人員，針對機台需要的時間段，去找有空的人，並時間改成沒有空(0)。因為前面已鏡檢查過這樣維修是可行的，所以不會出問題。