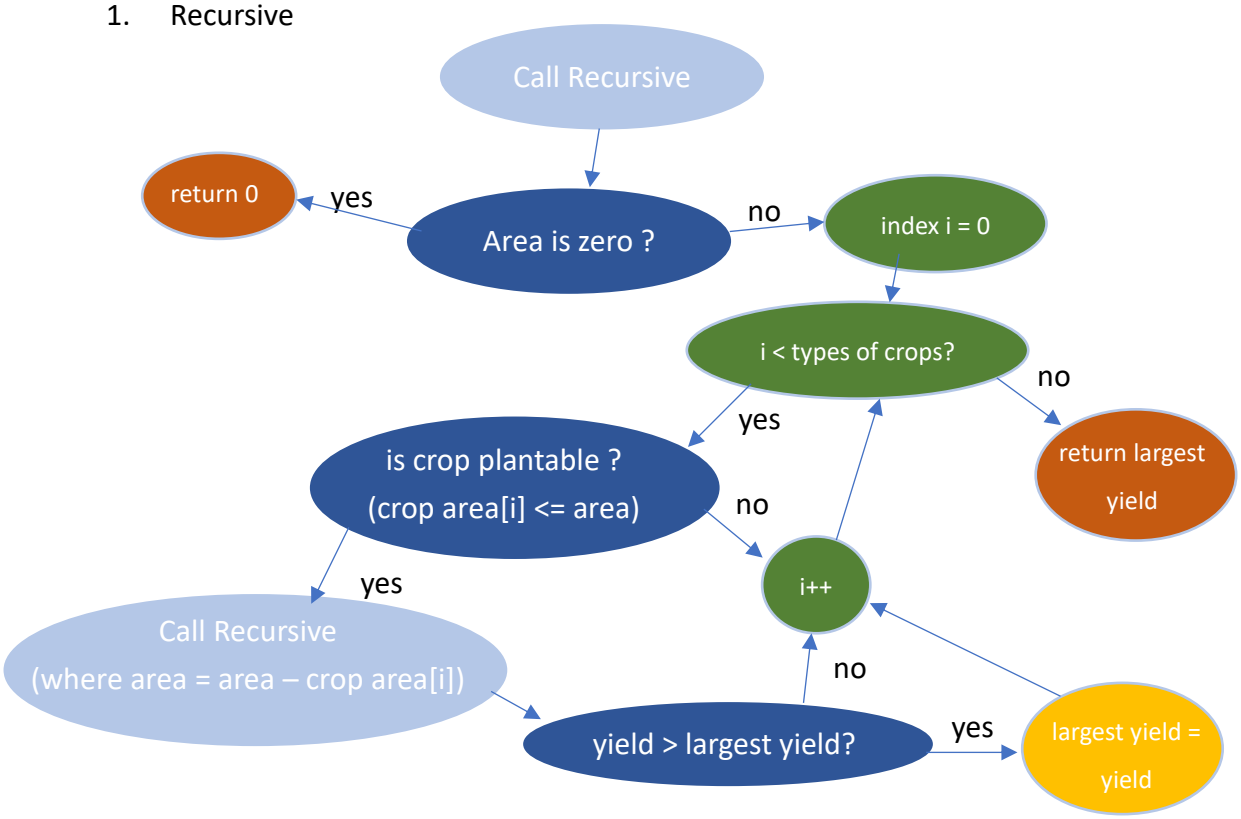
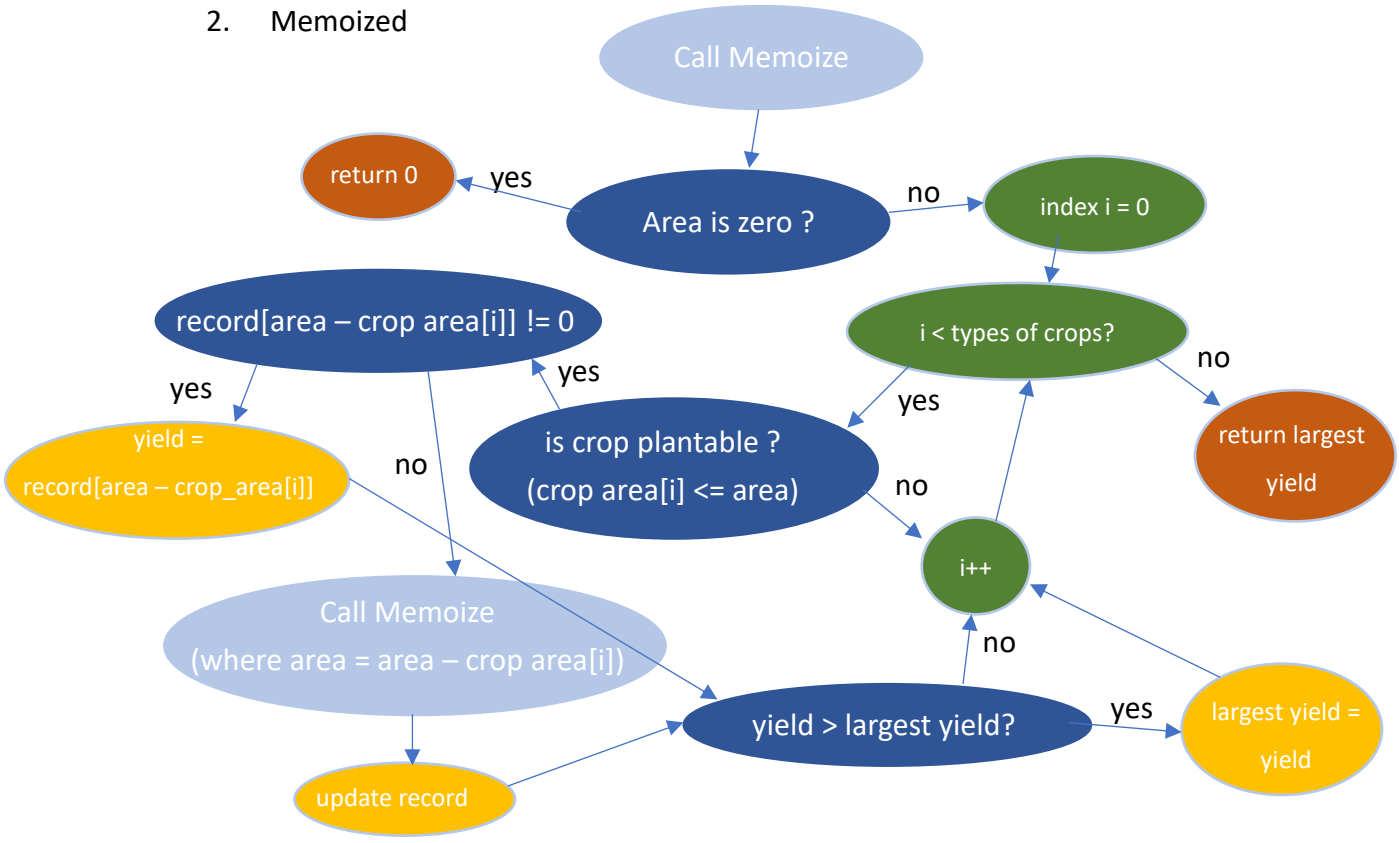
a. Flow Chart

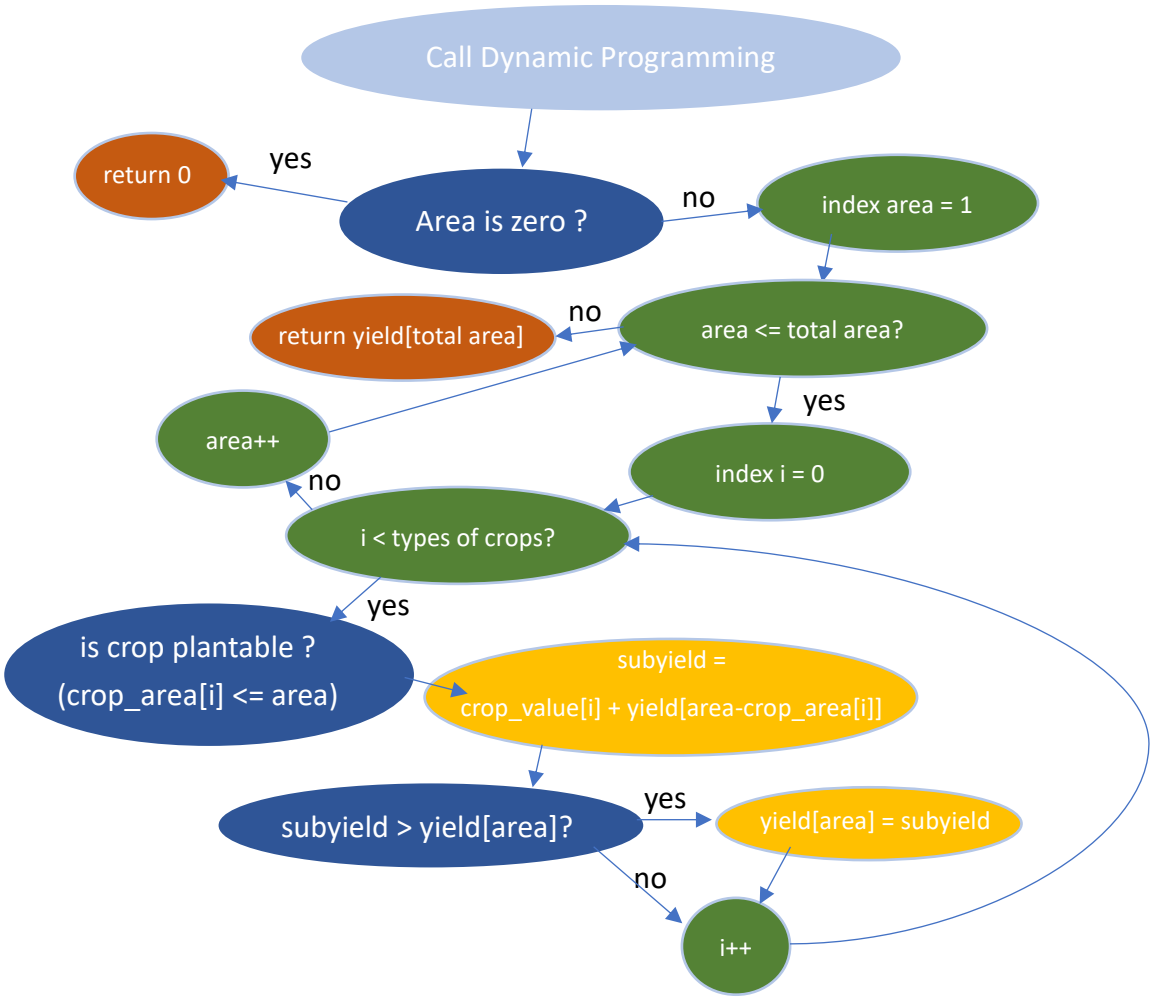
1. Recursive



2. Memoized



3. Dynamic Programming



b. Design Concept of Algorithm

1. Recursive
   - i. 依序分別假設(在面積允許的情況下)每種作物必種下一株
   - ii. 對剩下的面積做遞迴呼叫尋找最佳種植方式
   - iii. 最後比大小回傳最大值

在設計 recursive 的演算法要把握最大的原則就是遞迴的呼叫，把問題變成子問題然後在找到解答後層層回傳得到答案，是一個 top-down 的 approach

2. Memoized
   i. 依序分別假設(在面積允許的情況下)每種作物必種下一株
   ii. 檢查剩下的面積在先前遞迴呼叫是否計算過，有則跳到 iv.
   iii. 對剩下的面積做遞迴呼叫尋找最佳種植方式，並將回傳值記錄下來
   iv. 最後比大小回傳最大值

   Memoize 的做法和 recursive 很類似，一樣也是按照 top-down approach，唯一的差別在於會把運算過的 recursion 答案記錄下來，在做呼叫前先檢查之前有沒有做過一樣的運算，有的話就利用記錄下的答案，沒有的話就依樣遞迴呼叫並更新紀錄

3. Dynamic Programming
   i. 從面積為一開始，到總面積，分別對每個面積算出最佳解如下
   ii. 每算一總面積，就把扣掉當下作物面積的最佳解加上作物價值
   iii. 把得到的答案和種植其他作物的最大值做比對，如果大於則取代之
   iv. 完成一個面積所有的作物種植比對就將最大值更新到該面積最佳解的紀錄裏面

   Dynamic Programming 用的是 bottom-up 的方式去解最佳解，先解面積較小的最佳解，而每次計算面積最佳解的時候就是利用先前已經運算過較小面積的最佳解去求得

c. Implementation Comparison

在實際跑的執行時間比較表如下：

	Runtime (case1)	**Runtime (case2)**
Recursive	0.842 sec	2.441
Memoize	0.001 sec	0.001 sec
Dynamic Programming	0.001 sec	0.001 sec

很明顯能看到在時間上 Memoize 和 Dynamic Programming 都比 recursive 快許多，主要原因是因為 recursive 花費了很多時間在做已經做過的運算，top-down approach 無法保證不會再遇到先前算過一模一樣的子問題，所以導致很常都多花了時間在做不必要的運算，而 memoize 和 DP 分別用紀錄和 bottom-up 的方式去避免這樣的問題發生。DP 用的 bottom-up approach 是先從小問題開始算，然後再依照子問題去解母問題，這樣就不會有重複運算的問題，因為不同面積最佳解只會算一次；而 memoize 用查表的方式去紀錄

recursive 已經算過的面積最佳解，雖然有可能多一點 size 的 overhead 去建立 array，但整體效率還是高於 recurisve 很多，甚至理論上會比 DP 快一些，原因是因為 memoize 不見得對每種面積都會去做運算，而是只做「有需要」的面積最佳解，所以在運算時間上又小了 DP 一些。

d. Problems in Implement Time and Discussion

這次因為作業要求 case1, case2 要在五分鐘內完成，用 memoize 和 dynamic programming 都可以輕易過，但是在 recursive 的部分就遇到了一些問題，程式跑了很久都沒辦法跑完，經過進一步的分析，我發覺問題出在於很多遞迴呼叫其實呼叫的參數都一樣只是順序組合不一樣，例如：a→b→c、c→b→a、a→c→b 等呼叫順序其實都是一樣的。為了解決這樣的問題，我在 function 多加了一個參數，確保當下進行呼叫的 crop index 不能小於已經使用過的 crop index，這樣就可以排除重複的問題，進行這項修改後果然在幾秒內就能跑完。

e. Bonus Part

在 auxiliary cases 中，會發現和 original case 的作物種類和 cost 都一樣，只是每一項面積（包括總面積）都乘以了一個相同倍數 k，在 DP 中原先的 space complexity 就會從 W 變成 Wk（因為要計算的面積種類數多了 k 倍，存各面積最佳解的 array 也會放大 k 倍），time complexity 會從 O(nW)變成 O(nkW)。

要讓 time complexity 和原先的 original case 相同的辦法就是在進行 DP 前找出所有作物面積以及總面積的**最大公因數**，並把他們做 resize 的動作，這樣在進行 DP 的時候一樣可以在 space complexity W 和 time complexity O(nW+n) = O(nW)（+n 因為需要掃過每種作物一次做 resize）下完成，雖然兩者 big O 看似一樣，但是其實 execution time 和 overhead 上還是有差異，後者不用浪費那麼多空間和時間。