任務1:

1. 輸入欲排列數字之數量N，若N未介於1~9，則要求重新輸入N(在程式中level等效於說明文件中的N)。
2. 將arrivecount(總共到了幾組)歸零。
3. 將1,2,3,,,N放入動態陣列allowedvec。
4. 將全新的空陣列{}丟入findnext()中。
5. 輸出總共經過幾層(level)，level在先前等於N，因為每遞迴一層會在原數字陣列中多加一筆數字。
6. 輸出總共幾組答案。

任務2:

1. 輸入欲排列數字之數量N，若N未介於2~9，則要求重新輸入N(在程式中level等效於說明文件中的N)。
2. 將arrivecount(總共到了幾組)歸零。
3. 將讀取N比不重複數字放入動態陣列allowedvec。
4. 記錄起始時間
5. 將全新的空陣列{}丟入findnext()中。
6. 輸出耗時(當下時間-起始時間)
7. 輸出總共幾組答案。

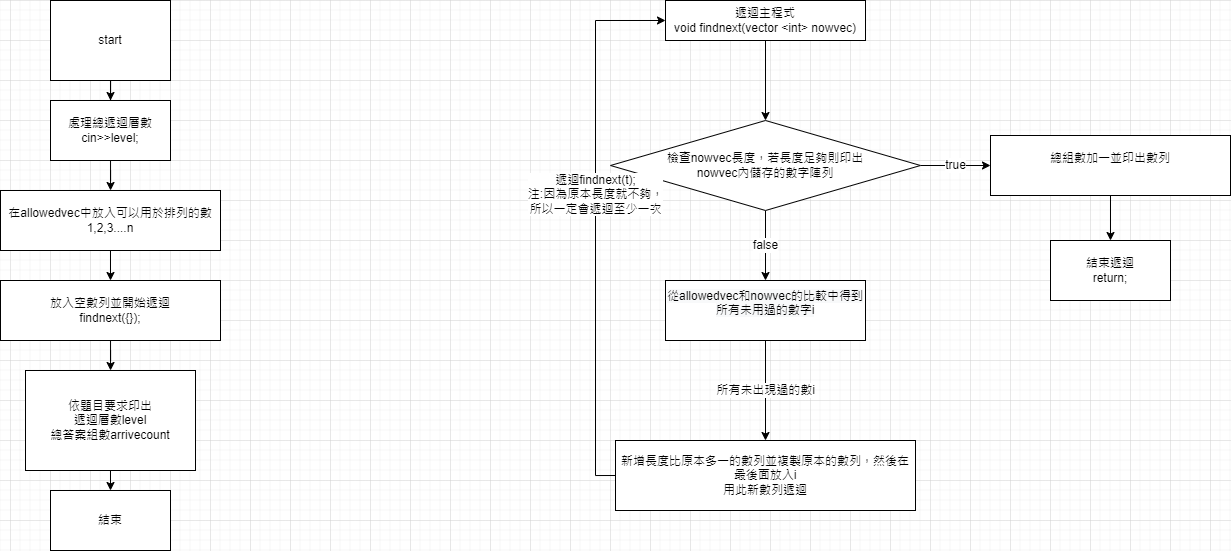
任務3:

1. 輸入欲排列數字之數量N，若N未介於1~9，則要求重新輸入N(在程式中level等效於說明文件中的N)。
2. 將arrivecount(總共到了幾組)歸零。
3. 將1,2,3,4,5,6,7,8,9放入動態陣列allowedvec。
4. 記錄起始時間
5. 將全新的空陣列{}丟入findnext()中。
6. 輸出耗時(當下時間-起始時間)
7. 輸出總共幾組答案。

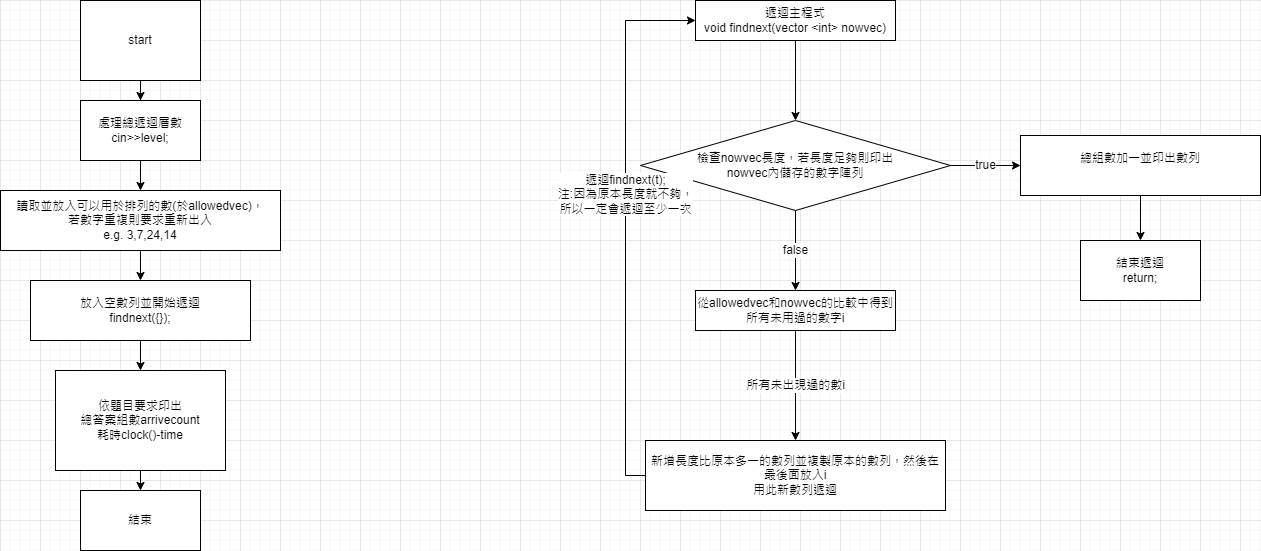
findnext(遞迴):

1. 確認現在的陣列長度是否等於level，若為真則輸出陣列內容跟組數(要加一)且程式結束(return)。
2. 將現在的陣列和allowedvec比較，找出所有沒有在現在的陣列出現的數。
3. 將所有沒出現的數分次放在現在的陣列後面然後再次呼叫findnext。

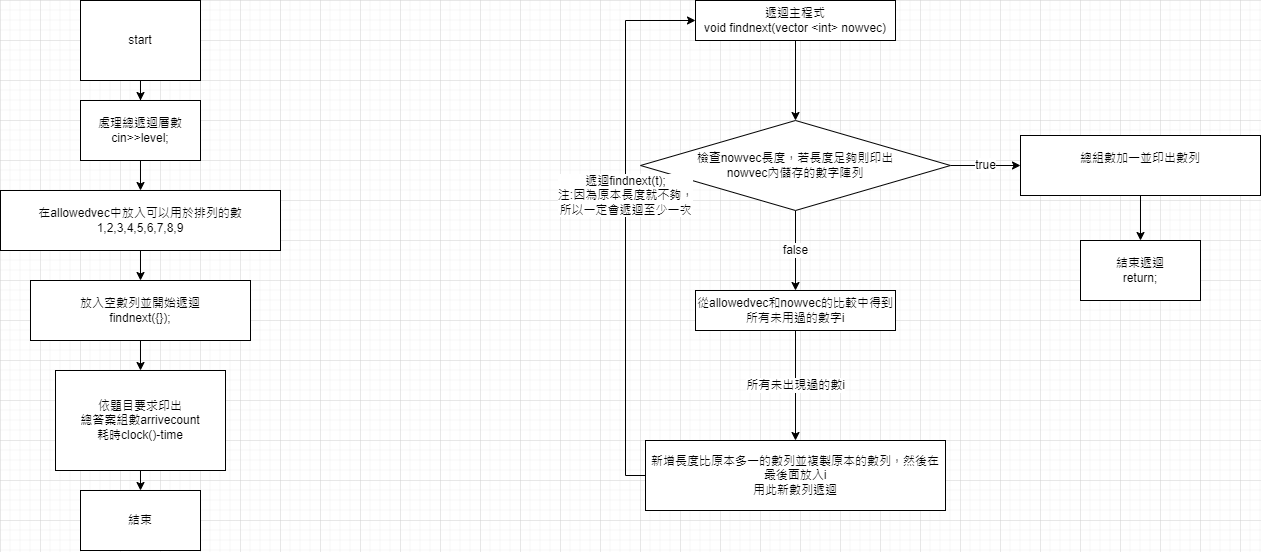
任務1流程圖



任務2流程圖



任務3流程圖



心得:此次作業要完成可謂是非常簡單，在使用熟悉的std內建搜尋後，此程式的主程式只花了不到20分鐘就完成，但是為了要求自己，我盡力的優化此程式的流程，試過許多降低時間複雜度的辦法，甚至使用多線程技術去解此程式，雖然確有進步，但在刪掉print的部分後，整個程式效率有了一個數量級的提升，可見cout在優化之後依舊很慢(printf更慢)。