```c
/* 3) Consider a system which has a finite number of instances of resource types A, B and
      C. A set of processes, P0-P4, compete for these resource instances and are allocated
      according to the snapshot in Fig.1 at a given point in time.

         Allocation          Max         Available
          A   B    C    A    B    C    A    B    C
      P0  0   0    2    0    0    4    1    0    2
      P1  1   0    0    2    0    1
      P2  1   3    5    1    3    7
      P3  6   3    2    8    4    2
      P4  1   4    3    1    5    7
                    Fig1. Snapshot

i)   Determine whether the system is safe.
ii)  If a request from process P2 arrives for (0, 0, 2), can the request be granted immediately?
*/

// What conditions must be satisfied for deadlock to occur.

// Algorithm -
//  In each pass over uncompleted processes : ( Some progress has to be made )
//       Can resource request be fullfilled
//       Request can be fullfilled by allocating resources if they are available
//       And on completion the process releases all its resources

//  So - if progress can be made in every pass, then we have a safe sequence of
//    allocation of resources to aviod deadlock, else we do not.

//  Progress is - during one pass on all waiting processes - Atleast one process is
//    allocated resources and goes to completion

//  This algorithm is also known as Banker's Algorithm : Its a resource allocation and
//  deadlock avoidance algorithm that tests for safety - by simulating the allocation for
//  predetermined maximum possible amounts of all resources, then makes an "s-state"
//  or "safe-state" check to test for possible activities, before deciding whether allocation
//  should be allowed to continue.

#include<stdio.h>
int main()
 {
  int process = 5, resource = 3;
  int i, j, instance, k = 0, count1 = 0, count2 = 0;
  // count1 : number of processes for which required resources were allocated

  int avail[resource] = {1, 0, 2} ; // Available resource Instances of each type: A , B and C
  int max[process][resource] = {0,0,4, 2,0,1, 1,3,7, 8,4,2, 1,5,7}; // Resource requirements of
each process
  int allot[process][resource] = {0,0,2, 1,0,0, 1,3,5, 6,3,2, 1,4,3}; // Resources already allocate
d to each process
  int need[process][resource];  // requiredResources - allocatedResources to each process
  int completed[process] = {0}; // Assume none of process have completed

  for( i=0; i<process; i++ ) // Build the need matrix
     for( j=0; j<resource; j++ )
```

```c
        need[i][j]=max[i][j]-allot[i][j];

  printf("\nPossible Sequence:\n");

  while( count1 != process ) // count1 : how many processes have completed execution
   {
    count2=count1; // Save the present state

    for( i=0; i<process; i++ ) // For each process
      {
       k = 0; // Count the different resource types that can be allocated to present process
       for( j=0; j<resource; j++ ) // For each resource
         {
          if( need[i][j] <= avail[j] ) // Can need be satisfied by currently available resources
            { // increment k as respective resource type can be allocated to present process
             k++;
            }
         } // for each resource completes

       if( k==resource && completed[i]==0 ) // If all resource types can be allocated and if pr
ocess has not already been selected for execution then
         {
          printf("\t p[%d]", i);
          completed[i]=1;      // Mark process as selected for execution

          for(j=0; j<resource; j++) //Free resources held by process, add to available resources
            {
             avail[j]= avail[j] + allot[i][j]; // or avail[j]+=allot[i][j];
            }

          count1++; // count this process has been selected in possible sequence of execution
         } // if( k==resource && completed[i]==0 ) block completes
      } // for each process block completes

    if( count1 == count2 ) // if no progress
      { // For all uncompleted process,  we were unable to allocate resources
       printf("\nStop...After this ...Deadlock\n");
       return 0; // return as no safe sequence of allocation
      }
   }// while ( count1 != process ) block completes

  // Control has come here because "We have a safe sequence of allocating resources to proce
sses such that deadlock does not happen"
  printf("\nSafe Sequence exists");

  return 0;
 } // End of main
// What is the run time of above algorithm, for p processes and r resources as Asymptotic nota
tion ?
// Write proof - Why the above algorithm works ?
```