Pedram Safaei
CS477
HW5
1.

a) There is a recursive relation on the below function so it counts as a recursive formula not a recursive implementation we are going to use a base case of 0, loop through our container which is a group of vectors at this case, and we start initializing this container m, so m[i][j] would be whatever is at i-j in our map's max member (it is going to be a number of drones which is initialized to 0 at the beginning) plus the minimum of the xi that we have contained in a vector and f[j] which we have also decided to contain in a vector, at the end we get the max element of these group of vectors by looping through them and comparing them (I am just assuming the user will use the max element function from the STL so I am not implementing a method for finding this max element). The final time complexity would be O(n^2) which is not bad considering that without dynamic programming the time complexity would be exponential.

```
void Solution_Recursive_Relation(map<int, Drones>VALUE,
vector<vector> m, vector Function_Value, vector Xi_Value )
{
   i;
   j;
   maxPtr;


   for( i = 1; i < Function_Value.size(); i++ )
   {
      m[i][0] = 0;

      for(j = 1; j <= i; j++ )
      {
         m[i][j] = VALUE[i-j].MAX + min( Xi_Value[i],
Function_Value[j] );
      }

      maxPtr = max_element(m[i].begin(), m[i].end());
      j = distance(m[i].begin(), maxPtr);

      // here we save optimal
      MAP_VALUE[i];
      MAP_VALUE[i].MAX = *maxPtr;
   }

}
```

b) I am using the numbers that were given to us on the HW

```
void Solution_Recursive_Relation( std::unordered_map<int, Drones>
&MAP_VALUE, std::vector<std::vector<int>> &m, std::vector<int>
&Function_Value, std::vector<int> Xi_Value )
{
   unsigned int i;
   unsigned int j;
   std::vector<int>::iterator maxPtr;
// Im using STL just to save time

   // loop through
   for( i = 1; i < Function_Value.size(); i++ )
   {
      m[i][0] = 0;
      for(j = 1; j <= i; j++ )
      {
         m[i][j] = MAP_VALUE[i-j].MAX + std::min( Xi_Value[i],
Function_Value[j] );
      }
      // find max for second
      maxPtr = max_element(m[i].begin(), m[i].end());
      j = distance(m[i].begin(), maxPtr );

      // save optimal
      MAP_VALUE[i];
      MAP_VALUE[i].MAX = *maxPtr;
   }

}
```

```
================================================================================
                      ########## Part B ##########
================================================================================

########## Part B SOLUTION ##########

Max number of drones taken out in  4 seconds is 5.

--------------------------------------------------------------------------------

########## Part B TABLE ##########

 i : After How many seconds
 j : How many seconds[of charge]
          j:
            1 2 3 4

          4  5 3 2 1
          3  3 3 4
i:        2  2 2
          1  1
```

C) we basically add
MAP_VALUE[i].PREVIOUS = i – j;
to the end of the algorithm from part B

```
void Solution_Recursive_Relation( std::unordered_map<int, Drones>
&MAP_VALUE, std::vector<std::vector<int>> &m, std::vector<int>
&Function_Value, std::vector<int> Xi_Value ){
   unsigned int i;
   unsigned int j;
   std::vector<int>::iterator maxPtr; // Im using STL just to save
time

   // loop through
   for( i = 1; i < Function_Value.size(); i++ ){
      m[i][0] = 0;
      for(j = 1; j <= i; j++ ){
         m[i][j] = MAP_VALUE[i-j].MAX + std::min( Xi_Value[i],
Function_Value[j] );
      }
      // find max for second
      maxPtr = max_element(m[i].begin(), m[i].end());
      j = distance(m[i].begin(), maxPtr );

      // save optimal
      MAP_VALUE[i];
      MAP_VALUE[i].MAX = *maxPtr;

      // part C
      MAP_VALUE[i].PREVIOUS = i - j;

      //part D
      MAP_VALUE[i].TIME = MAP_VALUE[i-j].TIME;
      MAP_VALUE[i].TIME.push_back(i);
   }

}
```

```
================================================================
                  ########## Part C ##########
================================================================

########## Part C Print out table of Values ##########

Table Reconstructed to show where each subproblem's solution came from:
we require to fire at i, as well as during value in braces of (i,j) [of course recursively]
          j:
             1    2    3    4

         4   5{3} 3{2} 2{1} 1{0}
         3   3{2} 3{1} 4{0}
i:       2   2{1} 2{0}
         1   1{0}
```

D) Again we only have to add a line to our algorithm from part B

```
which is
      MAP_VALUE[i].TIME = MAP_VALUE[i-j].TIME;
      MAP_VALUE[i].TIME.push_back(i);

// solve solution, recursive relation solution
void Solution_Recursive_Relation( std::unordered_map<int, Drones>
&MAP_VALUE, std::vector<std::vector<int>> &m, std::vector<int>
&Function_Value, std::vector<int> Xi_Value )
{
   unsigned int i;
   unsigned int j;
   std::vector<int>::iterator maxPtr; // Im using STL just to save
time

   // loop through
   for( i = 1; i < Function_Value.size(); i++ )
   {
      m[i][0] = 0;
      for(j = 1; j <= i; j++ )
      {
         m[i][j] = MAP_VALUE[i-j].MAX + std::min( Xi_Value[i],
Function_Value[j] );
      }
      // find max for second
      maxPtr = max_element(m[i].begin(), m[i].end());
      j = distance(m[i].begin(), maxPtr );

      // save optimal
      MAP_VALUE[i];
      MAP_VALUE[i].MAX = *maxPtr;

      // part C
      MAP_VALUE[i].PREVIOUS = i - j;

      //part D
      MAP_VALUE[i].TIME = MAP_VALUE[i-j].TIME;
      MAP_VALUE[i].TIME.push_back(i);
   }

}
```

```
================================================================
                    ########## Part D ##########
================================================================

########## Part D printout of the solution ##########

To take down 5 drones in 4 seconds, we must fire at seconds: 3 4
================================================================

ps@RedPS:~/Desktop/CS477/HW5$
```

2.
I'll show one, the others will be computed just like below
C and B
3*5*2 = 30
30 + 0 = 30
C and A
4*3*2 = 24
24+30 = 54
so our final result would be C*B*A

	A	B	C
C	54	30	0
B	60	0	-
A	0	-	-

3.

LCS = Longest Common Subsequence

A) <u>True</u>
Let's say we start X and Y with A then A has been used as a prefix to the LCS of some sequence of x1 and Y1 which is the sequence – first character of their sequence)this way we are not reducing the possible LCS of our X and Y

B) <u>True</u>
This is the same as what we discussed earlier in part A if we add A to LCS of x2 and Y2 which are sequences with no last character of A then that will produce LCS result for X and Y