Name :- Kiran Bagwe and Mukesh Dasari
UNCC ID :- 801223392 and 801208218

Extracting dependency from code is an almost automatic process. You need to choose a granularity. But once that is chosen, the entire analysis follows.
In the whole activity, you should express the metrics in complexity notation as a function of the parameters of the functions.

1 Coin Collection (from Midterm Spring 2018)

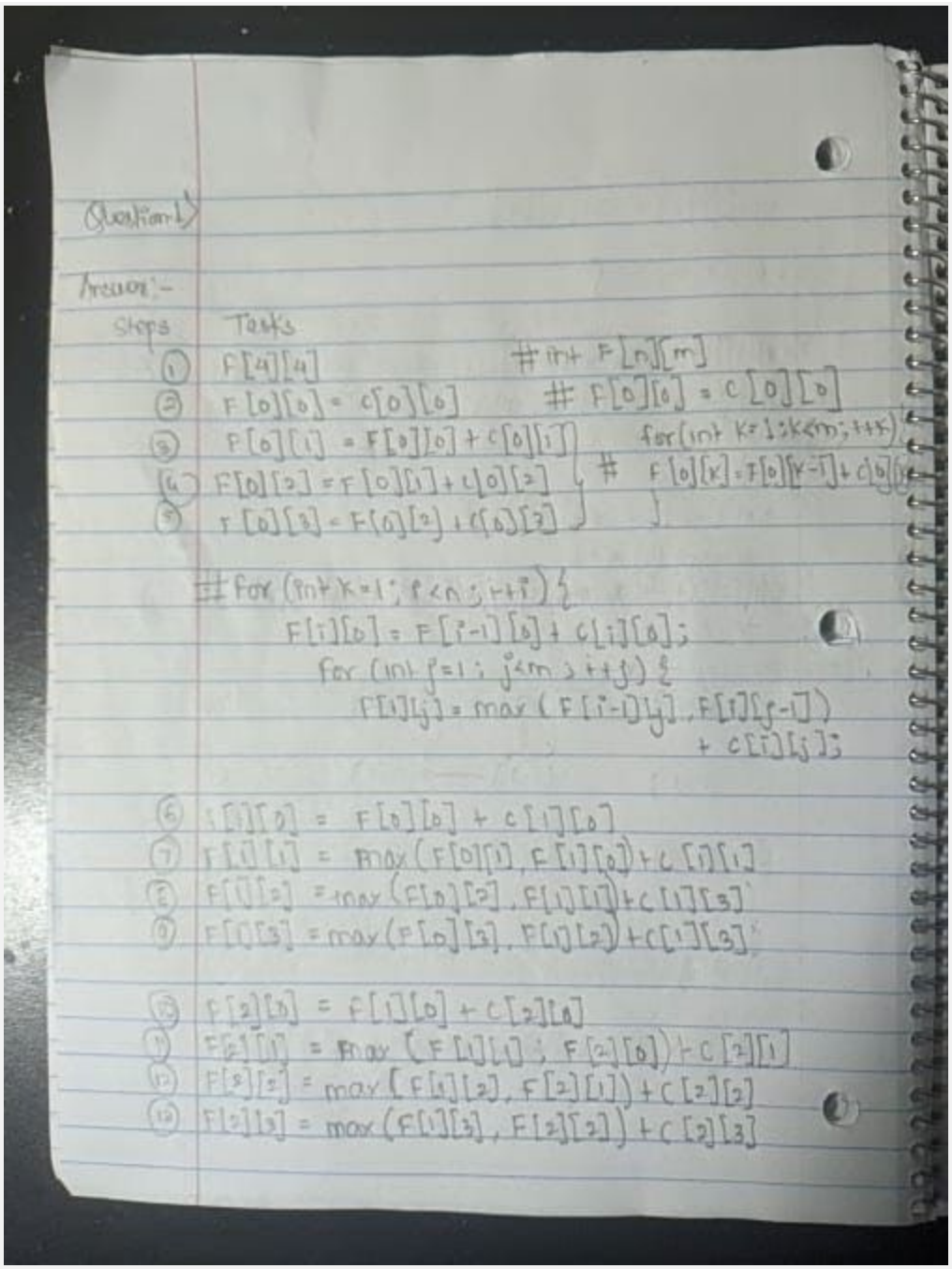
The Coin Collection problem is defined as follows:
Several coins are placed on an n × m board with at most one coin per cell of the board. A robot is initially located at the upper left cell of the board. The robot can only move to the right or down; it can not move up or left. When the robot visits a cell where a coin is located, it picks it up. At most, how many coins can the robot collect?

This problem can be solved by the following method:

```
void RobotCoin(int n, int m, //size of the board
               int C[n][m] //Is there a coin in (i,j)
               ) {

  int F[n][m]; //How many coins can be collected while on (i,j)

  F[0][0] = C[0][0];

  for (int k=1; k<m; ++k) {
    F[0][k] = F[0][k-1] + C[0][k];
  }

  for (int i=1; i<n; ++i) {
    F[i][0] = F[i-1][0] + C[i][0];
    for (int j=1; j<m; ++j) {
      F[i][j] = max (F[i-1][j], F[i][j-1]) + C[i][j];
    }
  }

  return F[n-1][m-1];
}
```

Question:What is the complexity of this function?
Answer :- The complexity of this function is 0(n*m)

Question:Extract the dependencies.



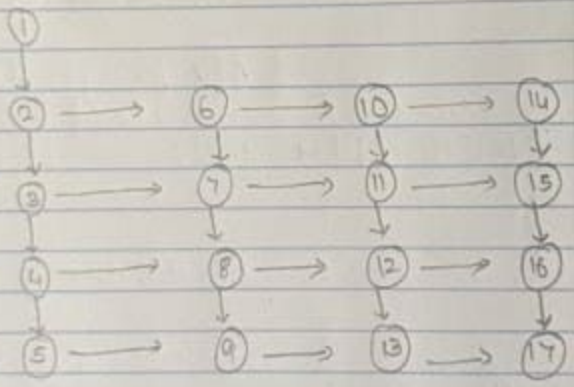
(14) $F[3][0] = F[2][0] + c[3][0]$

(15) $F[3][1] = \max(F[2][1], F[3][0]) + c[3][1]$

(16) $F[3][2] = \max(F[2][2], F[3][1]) + c[3][2]$

(17) $F[3][3] = \max(F[2][3], F[3][2]) + c[3][3]$

Graph -



Question:What is the width?
Answer:- Since F[4][4] is only an independent task, the width is 1
Question:What is the work?
Answer:- The work is sum of processing time required to complete n*m tasks.i.e 0(n*m)
Question:What is the critical path? What is its length?
Answer:- Critical Path is 0(n+m) and the length is the sum of processing time of all the task within the critical path.

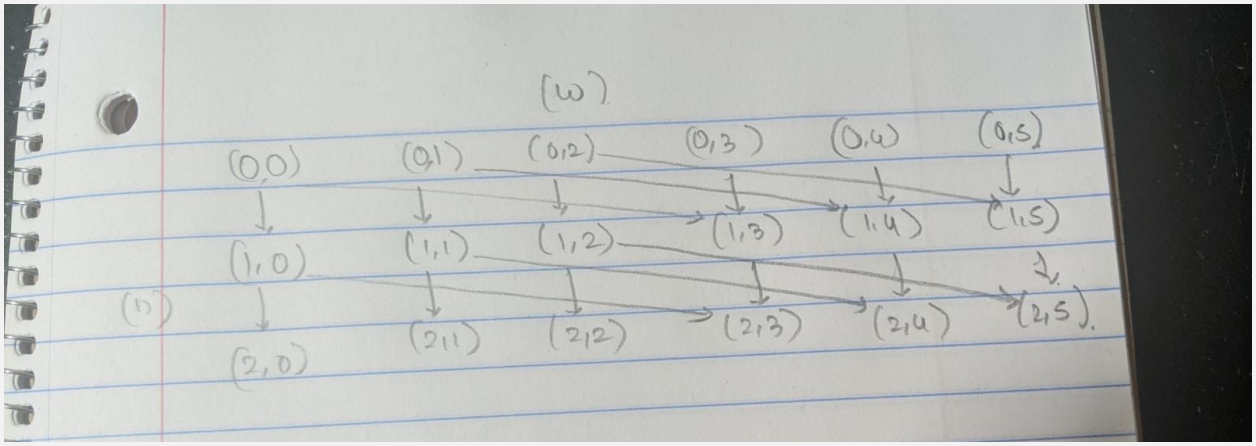
2 Knapsack

The Knapsack problem aims at finding the best set of objects to pack in a bag. Often the following dynamic a programming algorithm is used to solve the problem.

```
void knapsack (int n, int W, int value[], int weight[], int val[][]) {
  for (int a = 0; a<=W; ++a) {
    val[0][a] = 0;
  }

  for (int i=1; i<=n; ++i) {
    for (int j=0; j<=W; ++j) {
      val[i][j] = val[i-1][j];
      if (weight[i-1] <= j) {
        val[i][j] = max (val[i-1][j], value[i-1]+val[i-1][j-weight[i-1]]);
      }
    }
  }
}
```

Question:What is the complexity of this function?
Answer: Complexity of algorithm is O(nW)

Question:Extract the dependencies.
Answer:
Value → [1, 2, 1, 4]
Weight → [3, 1, 2, 6]
N → 2
W → 5



Question:What is the width?
Answer:As mentioned in first for loop, the value of a depends upon the value on W which is indirectly the reason for independent task, Hence the width is W
Question:What is the work?
Answer:The work will be sum of processing time required to complete n*W task i.e O(n*W)
Question:What is the critical path? What is its length?
Answer:The critical path would be length of the diagonal as shown in the diagram above(n -- > W)

3 Bubble Sort

The bubble sort algorithm can be written like this:

```
void bubblesort(int* A, int n) {
  for (int i=0; i<n; ++i) {
    for (int j=1; j<n; ++j) {
      if (A[j] < A[j-1]) {
        int temp = A[j];
        A[j] = A[j-1];
        A[j-1] = temp;
      }
    }
  }
}
```

Question:What is the complexity of this function?
Answer: The complexity of this function is O(n^2)
Question:Extract the dependencies.
Answer:

[3, 6, 1, 4, 7]

N=5

I=0 J=0	I=0 J=1 A[0,1] R A[0,0] R Temp w A[0,1] w A[0,0] w	I=0 J=2 A[0,2] R A[0,1] R Temp w A[0,2] w A[0,1] w	I=0 J=3 A[0,3] R A[0,2] R Temp w A[0,3] w A[0,2] w	I=0 J=4 A[0,4] R A[0,3] R Temp w A[0,4] w A[0,3] w
I=1 J=0	I=1 J=1 A[1,1] R A[1,0] R Temp w A[1,1] w A[1,0] w	I=1 J=2 A[1,2] R A[1,1] R Temp w A[1,2] w A[1,1] w	I=1 J=3 A[1,3] R A[1,2] R Temp w A[1,3] w A[1,2] w	I=1 J=4 A[1,4] R A[1,3] R Temp w A[1,4] w A[1,3] w
I=2 J=0	I=2 J=1 A[2,1] R A[2,0] R Temp w A[2,1] w A[2,0] w	I=2 J=2 A[2,2] R A[2,1] R Temp w A[2,2] w A[2,1] w	I=2 J=3 A[2,3] R A[2,2] R Temp w A[2,3] w A[2,2] w	I=2 J=4 A[2,4] R A[2,3] R Temp w A[2,4] w A[2,3] w
I=3 J=0	I=3 J=1 A[3,1] R A[3,0] R Temp w A[3,1] w A[3,0] w	I=3 J=2 A[3,2] R A[3,1] R Temp w A[3,2] w A[3,1] w	I=3 J=3 A[3,3] R A[3,2] R Temp w A[3,3] w A[3,2] w	I=3 J=4 A[3,4] R A[3,3] R Temp w A[3,4] w A[3,3] w
I=4 J=0	I=4 J=1 A[4,1] R A[4,0] R Temp w A[4,1] w A[4,0] w	I=4 J=2 A[4,2] R A[4,1] R Temp w A[4,2] w A[4,1] w	I=4 J=3 A[4,3] R A[4,2] R Temp w A[4,3] w A[4,2] w	I=4 J=4 A[4,4] R A[4,3] R Temp w A[4,4] w A[4,3] w

Here only row values are dependent. So the dependency graph would be,
00 → 01 → 02 → 03 → 04
10 → 11 → 12 → 13 → 14
20 → 21 → 22 → 23 → 24
.
Question:What is the width?
Answer: So, each row is independent of other rows and which are comparisons of each number with rest in the
So width would be N.

Question:What is the work?
Answer: Total work is N*(N-1)

Question:What is the critical path? What is its length?
Answer: Critical path is each row. In the above example 00 → 01 → 02 → 03 → 04 can be the critical path.