Activity : Extracting Parallelism (2D cases)

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 $n \times m$ 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?
Question: Extract the dependencies.
Question: What is the width?
Question: What is the work?
Question: What is the critical path? What is its length?

2 Knapsack

The Knapsack problem aims at finding the best set of objects to pack in a bag. Often the following dynamic 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]]);
      }
    }
  }
}
```

(You can assume weight is positive.)

Question: What is the complexity of this function?
Question: Extract the dependencies.
Question: What is the width?
Question: What is the work?
Question: What is the critical path? What is its length?

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?
Question: Extract the dependencies.
Question: What is the width?
Question: What is the work?
Question: What is the critical path? What is its length?