3.11

- 2 ways to see it:
  - Either count directly with the help of slide 24 lecture 5: tasks for the first loop n(n-1)/2 to compute the L[j,k] but also U[k,j] + the "splitting" of the element of the diagonal (n) + the loop on the smaller square matrix (size k at every iteration).

$$2\frac{n(n-1)}{2}+n+\sum_{i=1}^{n-1} i^2=\sum_{i=1}^{n} i^2$$

  - Or recursively: at a given iteration every element of the sub-matrix of size k is touched, hence k^2 tasks, and you add the count for the previous iteration, and you have $t(m)=t(m-1)+m^2$, or the sum of squares directly.

3.12 & 3.13

- 3.12) Maximum degree of concurrency is given by
  - Either the first loop: 2(m-1) tasks in parallel (m-1 for L and U),
  - Or the second loop $(m-1)^2$ tasks in parallel (sub-matrix).
  - There is a dependency between the first and the second loop so it is the $\max(2(m-1), (m-1)^2)$.
- 3.13) Critical path length: Let's check the dependencies. Every element in the diagonal (except the first) needs an update from the second loop of the algorithm (on the sub-matrix) but its coefficient are computed by the first loop. That gives us a sub-path of length 2 between every "split" of the diagonal element to its L and U parts. The critical path length is then 2(m-1)+1 = 2m-1.

3.15, 3.17 & 3.21

- 3.15 & 3.17) See chapter 13.
- 3.21) We have to find the most balanced and imbalanced combinations for keeping 2 processors busy. The best is perfect balanced where we have a speedup of 2. The worst tries to keep one processor idle for the longest possible time with no task available where we get a speedup of 1.5.