Activity: Extracting Parallelism (Recursive)

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 Fast Exponentiation

Consider this function to compute x^n where n is a positive integer.

```
double expBySquaring(double x, int n) {
  if (n == 0)
    return  1;
  if (n == 1)
    return  x;
  if (n % 2 == 0)
    return expBySquaring(x * x,  n / 2);
  else
    return x * expBySquaring(x * x, (n - 1) / 2);
}
```

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 Dense Matrix Matrix Multiplication Recursively

Consider this algorithm to compute $C = A * B$ when A, B, and C are $n \times n$ matrices where n is a power of 2.

```
Multiply(A, B):
  A11 = A[1..n/2][1..n/2]
  A12 = A[1..n/2][n/2..n]
  A21 = A[n/2..n][1..n/2]
  A22 = A[n/2..n][n/2..n]

  B11 = B[1..n/2][1..n/2]
  B12 = B[1..n/2][n/2..n]
  B21 = B[n/2..n][1..n/2]
  B22 = B[n/2..n][n/2..n]

  C11 = A11*B11 + A12*B21
  C12 = A11*B12 + A12*B22
  C21 = A21*B11 + A22*B21
  C22 = A21*B12 + A22*B22

  return [[C11, C12],[C21, C22]]
```

Note that the $*$ operation are done by recursively calling the Multiply function. And that the $+$ operation is a matrix operation.
Question: What is the complexity of this function? (Hint: use Master theorem)
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 Merge Sort

Question: Recall the merge sort algorithm. (Give the algorithm.)
Question: What is the complexity of this function?
Question: Extract the dependencies.

(Hint: instead of using loop iterations as a task, you can use function calls and function return as tasks. Think that merge sort is recursive! Remember that when working with functions, a name in two different function can represent different underlying variable/memory location.)
Question: Do all tasks have the same processing time?
Question: What is the width?
Question: What is the work?
Question: What is the critical path? What is its length?
Question: How does the schedule of such an algorithm look like when $P = 4$? (What I mean is that what ever the values of n, the schedules have "shapes". What "shape" does any schedule for this problem have? The sketch of what a Gantt chart would look like answer the question.)