Fast, adaptive implementation of
the Cooley-Tukey FFT (FFTW)

1.) Locality of data access
   - choose recursive FFT, not iterative FFT
   - DFTum = (DFTx ⊕ Im) ⊙ (In ⊕ DFTm) L (DIT)

\[
\text{(schematic)} = \left( \begin{array}{l}
\text{ fuse and compute DFTum' s} \\
\text{ part of middle} \\
\rightarrow \text{DFT(k, x, t, s)} \\
\text{ in-vector = out-vector, in-stride = out-stride}
\end{array} \right) \cdot L
\]

- stride as parameter
- out-of-place
- interface handles arbitrary recursions
- interface does not handle recursions
   - in FFTW implemented as basic blocks
     (unrolled, optimized code)

Explain why DIT is better than DIF.
2.) Precomputed constants
   - sin/cos are very expensive to compute at runtime
   - Solution: precompute in init(...) function and store in table

3.) Fast basic blocks for small sizes
   - Slides

4.1) Adaptivity
   - Search our relevant algorithm space

Dynamic programming search:
   - Recursively, bottom up, build table of best recursions.

5.) Other cache optimization
   - After the spring break

- Much faster than exhaustive search, but assumes best FFT is independent of context.
**DAG example**

\[ \text{DFT}_2 \cdot \text{diag} (1, c) \]

\[ x_0 \rightarrow y_0 \]

\[ x_1 \rightarrow y_1 \]

loads mult add store

**CSE on transposed DAG**

**DAG transposition:**

\[ \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \begin{pmatrix} 5 & 3 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \]

\[ A = \begin{pmatrix} 5 & 3 \\ 2 & 4 \end{pmatrix} \]

**Example:**

\[ a \rightarrow \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \]

\[ c = 4(2a + 3b) \rightarrow 8a + 12b \]

\[ a = 2.4c \rightarrow 8c \]

\[ b = 3.4c \rightarrow 12c \]

\[ -2 \text{ ops} \] destroys two subexpressions

\[ -2 \text{ ops} \] destroys two subexpressions