Question 6

This script implements both a vanilla DFT and a 15-point Cooley-Tukey FFT from scratch and compares their performance.

In [ ]:
```
from pathlib import Path

import numpy as np
import scipy.fft as fft

import matplotlib.pyplot as plt
import seaborn as sns

from a3_config import A3_ROOT, SAVEFIG_CONFIG
```

Construct Signal

In [ ]:
```
import random; random.seed(24)

x_signal = np.array([2 * (random.random() - 0.5) for _ in range(15)])
h_sigref = fft.fft(x_signal)[:7]

t_signal = np.arange(0, 1, 1/15)
f_signal = fft.fftfreq(15, 1/15)[:7]
```

In [ ]:
```
# Export the signal for Question 7
fname = Path(A3_ROOT, "output", "q6_signal_out.npy")
np.save(fname, x_signal)
```

Vanilla DFT

In [ ]:
```
from typing import Any

def dft(x: Any, n: Any = None) -> Any:
    """Compute the 1-D discrete Fourier transform."""
    n = n or len(x)
    X = lambda k: sum(x[i] * np.exp(-2 * np.pi * 1j * i * k / n) for i in range(
    return np.array([X(k) for k in range(n)])

h_sigdft = dft(x_signal)[:7]
```

Cooley-Tukey FFT

In [ ]:
```
def cooley_tukey(x: Any, radix: int = 3) -> Any:
    """Compute the 1-D discrete Fourier transform using the Cooley-Tukey FFT."""
    if (n := len(x)) % radix != 0:
        raise ValueError(f'input length must be multiple of radix')
    n_rows = radix
    n_cols = int(n / radix)
    # Define DFT operation
    X = lambda x, k, n: sum(x[i] * np.exp(-2 * np.pi * 1j * i * k / n) for i in
```

```
    # Reshape into matrix in row-major order
    x = x.reshape(radix, n_cols)
    # Transform columns
    x = np.array([[X(x[:, j], k, n_rows) for k in range(n_rows)] for j in range(
    # Apply twiddle factors
    for j in range(n_cols):
        for i in range(n_rows):
            x[i, j] *= np.exp(-1j * 2 * np.pi * i * j / n)
    # Transform rows
    x = np.array([[X(x[i, :], k, n_cols) for k in range(n_cols)] for i in range(
    # Reshape back into vector in column-major order
    return x.T.reshape(n)

h_sigfft = cooley_tukey(x_signal, radix=3)[:7]
```

In [ ]:

```
sns.set_palette(sns.color_palette('mako', n_colors=3))

# Plot signal and its DFT
fig, axs = plt.subplots(1, 2, figsize=(7.5, 1.5))

sns.lineplot(x=t_signal, y=x_signal, ax=axs[0])
sns.lineplot(x=f_signal, y=np.abs(h_sigref), ax=axs[1], lw=1, label=r'\texttt{sc
sns.lineplot(x=f_signal, y=np.abs(h_sigdft), ax=axs[1], lw=1, label='Direct DFT'
sns.lineplot(x=f_signal, y=np.abs(h_sigfft), ax=axs[1], lw=1, label='Cooley-Tuke

axs[0].set_xlabel("Time (s)")
axs[1].set_xlabel("Frequency (Hz)")

sns.move_legend(axs[1], loc='upper left', bbox_to_anchor=(1, 1))

fig.tight_layout()
fig.savefig(Path(A3_ROOT, "output", "q6_signal.png"), **SAVEFIG_CONFIG)
```

Performance Comparison

In [ ]:

```
import time
from tqdm import trange

N_TRIALS = 10000

time_start = time.time()
for _ in trange(N_TRIALS):
    h_signal = fft.fft(x_signal)
time_elapsed = time.time() - time_start
print(f'scipy.fft ({N_TRIALS} trials): {time_elapsed * 1000 / N_TRIALS:.5f} ms')

time_start = time.time()
for _ in trange(N_TRIALS):
    h_signal = dft(x_signal)
time_elapsed = time.time() - time_start
print(f'DFT ({N_TRIALS} trials): {time_elapsed * 1000 / N_TRIALS:.5f} ms')

time_start = time.time()
for _ in trange(N_TRIALS):
    h_signal = cooley_tukey(x_signal, radix=3)
time_elapsed = time.time() - time_start
print(f'Cooley-Tukey FFT ({N_TRIALS} trials): {time_elapsed * 1000 / N_TRIALS:.5
```