Code 1

```
from sgp4.api import Satrec, jday
import numpy as np
from datetime import datetime, timedelta
from pyproj import Transformer
import time

# Function to read TLE data from a file
def read_tle(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()
    tle_data = [(lines[i].strip(), lines[i+1].strip(), lines[i+2].strip()) for
i in range(0, len(lines), 3)]
    return tle_data

# Function to write the output to a file
def write_output(file_path, data):
    with open(file_path, 'w') as file:
        for entry in data:
            file.write(",".join(map(str, entry)) + "\n")

# Function to log errors
def log_error(file_path, error_message):
    with open(file_path, 'a') as file:
        file.write(error_message + "\n")

# Function to convert ECEF to LLA
def ecef2lla(i, pos_x, pos_y, pos_z, transformer):
    lon, lat, alt = transformer.transform(pos_x[i], pos_y[i], pos_z[i],
radians=False)
    return lon, lat, alt

# Function to check if a point is within a rectangle defined by four corners
def is_within_bounds(lat, lon, bounds):
    lat_min = min(bounds, key=lambda x: x[0])[0]
    lat_max = max(bounds, key=lambda x: x[0])[0]
    lon_min = min(bounds, key=lambda x: x[1])[1]
    lon_max = max(bounds, key=lambda x: x[1])[1]
    return lat_min <= lat <= lat_max and lon_min <= lon <= lon_max

def main():
    # Record the start time
    start_time = time.time()

    # Read TLE data
    tle_file = '30000sats.txt'
    tle_data = read_tle(tle_file)
```

```
    # Define the observation period (one day) and time intervals (one minute)
    start_time_obs = datetime(2020, 1, 29, 0, 0, 0)
    end_time_obs = start_time_obs + timedelta(days=1)
    time_step = timedelta(minutes=1)

    # Log file for errors
    error_log_file = 'error_log.txt'

    # Initialize the Transformer object
    ecef = "EPSG:4978"  # ECEF (Earth-Centered, Earth-Fixed) coordinate system
    lla = "EPSG:4326"   # WGS84 Latitude/Longitude coordinate system
    transformer = Transformer.from_crs(ecef, lla)

    # Get bounds from user input
    bounds = [
        (16.66673, 103.58196),
        (69.74973, -120.64459),
        (-21.09096, -119.71009),
        (-31.32309, -147.79778)
    ]

    # Process each TLE set
    for satellite_name, tle_line1, tle_line2 in tle_data:
        # Initialize the satellite
        satellite = Satrec.twoline2rv(tle_line1, tle_line2)

        # List to hold the output data for the current satellite
        output_data = []
        Lx = []
        Ly = []
        Lz = []
        Vx = []
        Vy = []
        Vz = []

        # Propagate the satellite position and velocity
        current_time = start_time_obs
        while current_time < end_time_obs:
            year = current_time.year
            month = current_time.month
            day = current_time.day
            hour = current_time.hour
            minute = current_time.minute
            second = current_time.second

            jd, fr = jday(year, month, day, hour, minute, second)
            e, r, v = satellite.sgp4(jd, fr)
```

```
            if e == 0:
                Lx.append(r[0])
                Ly.append(r[1])
                Lz.append(r[2])
                Vx.append(v[0])
                Vy.append(v[1])
                Vz.append(v[2])
            else:
                error_message = f"SGP4 propagation error for {satellite_name}
at {current_time}: error code {e}"
                log_error(error_log_file, error_message)

            current_time += time_step

        # Convert ECEF to LLA and append to output data
        for i in range(len(Lx)):
            lona, lata, alta = ecef2lla(i, Lx, Ly, Lz, transformer)
            if is_within_bounds(lata, lona, bounds):
                output_data.append([lona, lata, alta])

        # Write the output data to a file
        output_file = f'{satellite_name}_positions.csv'
        write_output(output_file, output_data)

    # Calculate the elapsed time
    elapsed_time = time.time() - start_time
    print("Time elapsed:", elapsed_time, "seconds")
    print("Data processing complete.")

if __name__ == "__main__":
    main()
```

```
Time elapsed: 161.0241198539734 seconds
Data processing complete.
```

✓ 2m 41s completed at 23:39

Code 2

```python
from sgp4.api import Satrec, jday
from datetime import datetime, timedelta
from pyproj import Transformer
import multiprocessing
import time

error_log_file = 'error_log.txt'

# Function to read TLE data from a file
def read_tle(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()
    return [(lines[i].strip(), lines[i+1].strip(), lines[i+2].strip()) for i
in range(0, len(lines), 3)]

# Function to write the output to a file
def write_output(file_path, data):
    with open(file_path, 'w') as file:
        for entry in data:
            file.write(",".join(map(str, entry)) + "\n")

# Function to log errors
```

```python
def log_error(file_path, error_message):
    with open(file_path, 'a') as file:
        file.write(error_message + "\n")

# Function to convert ECEF to LLA
def ecef2lla(pos_x, pos_y, pos_z, transformer):
    lon, lat, alt = transformer.transform(pos_x, pos_y, pos_z, radians=False)
    return lon, lat, alt

# Function to check if a point is within a rectangle defined by four corners
def is_within_bounds(lat, lon, bounds):
    lat_min = min(bounds, key=lambda x: x[0])[0]
    lat_max = max(bounds, key=lambda x: x[0])[0]
    lon_min = min(bounds, key=lambda x: x[1])[1]
    lon_max = max(bounds, key=lambda x: x[1])[1]
    return lat_min <= lat <= lat_max and lon_min <= lon <= lon_max

# Function to propagate satellite positions
def propagate_satellite(args):
    tle_data, propagation_start_time, end_time, time_step, transformer, bounds
= args
    satellite_name, tle_line1, tle_line2 = tle_data
    satellite = Satrec.twoline2rv(tle_line1, tle_line2)

    output_data = []
    Lx = []
    Ly = []
    Lz = []

    current_time = propagation_start_time
    while current_time < end_time:
        year = current_time.year
        month = current_time.month
        day = current_time.day
        hour = current_time.hour
        minute = current_time.minute
        second = current_time.second

        jd, fr = jday(year, month, day, hour, minute, second)
        e, r, v = satellite.sgp4(jd, fr)

        if e == 0:
            Lx.append(r[0])
            Ly.append(r[1])
            Lz.append(r[2])
        else:
            error_message = f"SGP4 propagation error for {satellite_name} at
{current_time}: error code {e}"
```

```
            log_error(error_log_file, error_message)

        current_time += time_step

    for i in range(len(Lx)):
        lona, lata, alta = ecef2lla(Lx[i], Ly[i], Lz[i], transformer)
        if is_within_bounds(lata, lona, bounds):
            output_data.append([lona, lata, alta])

    output_file = f'{satellite_name.strip().replace(" ", "_")}_positions.csv'
    write_output(output_file, output_data)

    return f"Processed {satellite_name}"

def main():
    # Record the start time
    start_time = time.time()

    # Read TLE data
    tle_file = '30sats.txt'
    tle_data = read_tle(tle_file)


    # Define parameters
    propagation_start_time = datetime(2020, 1, 29, 0, 0, 0)
    end_time = propagation_start_time + timedelta(days=1)
    time_step = timedelta(minutes=1)
    bounds = [
        (16.66673, 103.58196),
        (69.74973, -120.64459),
        (-21.09096, -119.71009),
        (-31.32309, -147.79778)
    ]
    transformer = Transformer.from_crs("EPSG:4978", "EPSG:4326")

    # Log file for errors
    error_log_file = 'error_log.txt'

    # Create multiprocessing pool
    pool = multiprocessing.Pool()

    # Propagate satellite positions
    args = [(tle, propagation_start_time, end_time, time_step, transformer,
bounds) for tle in tle_data]
    results = pool.map(propagate_satellite, args)

    # for result in results:
    #     print(result)
```

```
    # Record the end time
    end_time = time.time()

    # Calculate the elapsed time
    elapsed_time = end_time - start_time
    print("Time elapsed:", elapsed_time, "seconds")
    print("Data processing complete.")

if __name__ == "__main__":
    main()
```

```
Time elapsed: 144.19086909294128 seconds
Data processing complete.
```

✓ 2m 25s completed at 23:43

code 3