```
    for pos_i in range(len(state.lattice)):
        for pos_j in range(len(state.lattice)):
            interaction_energy += -state.J*state.lattice[pos_i,pos_j]*(state.lattice[pos_i,(pos_j+1)%state.N] + state.lattice[(pos_i-1)%state.N,pos_j] + state.lattice[(pos_i+1)%state.N,pos_j] + state.lattice[pos_i,(pos_j-1)%state.N])
    interaction_energy *= 0.5 # avoid double counting

    magnetisation_energy = -state.H*calculate_magnetisation(state)

    return interaction_energy+magnetisation_energy


# In[ ]:


# Function to normalize data by number of lattice sites
def normalize_data(data, N):
    return data/(N**2)


# In[ ]:


# Metropolis algorithm
def Metropolis_algorithm(N_range, J, H, T_range, no_of_sweeps):
    # arrays to store magnetisation and energy time series
    M = np.zeros((len(N_range),len(T_range),no_of_sweeps))
    E = np.zeros((len(N_range),len(T_range),no_of_sweeps))
    M_normalized = np.zeros((len(N_range),len(T_range),no_of_sweeps))
    E_normalized = np.zeros((len(N_range),len(T_range),no_of_sweeps))
    run_time = np.zeros((len(N_range),len(T_range)))

    # Monte Carlo sweeps
    for N_index in range(len(N_range)):
        for T_index in range(len(T_range)):
            state = Ising(N_range[N_index],J,H)

            t0 = time.time()
            for sweep in range(no_of_sweeps): # run!
                monte_carlo_sweep(state,T_range[T_index])

                M[N_index,T_index,sweep] = calculate_magnetisation(state)
                E[N_index,T_index,sweep] = calculate_energy(state)
                M_normalized[N_index,T_index,sweep] = normalize_data(M[N_index,T_index,sweep],state.N)
                E_normalized[N_index,T_index,sweep] = normalize_data(E[N_index,T_index,sweep],state.N)

                if sweep%1000 == 0: # output checkpoints
                    print('N = {0}, T = {1}, sweep = {2}'.format(N_range[N_index],T_range[T_index],sweep))
            t1 = time.time()
            run_time[N_index,T_index] = t1-t0

    return M, E, M_normalized, E_normalized, run_time


# In[ ]:


# Function to plot evolution of data over time
def plot_time_series(data, N_range, N_index_range, T_range, T_index_range, start_point, stop_point, xname, yname):
    for N_index in N_index_range:
        for T_index in T_index_range:
```