In [1]:
```
# The goal of this assignment is to the code the plotting of the velocity dispersion
# of dark matter particles within the Jacobi Radius of M33 over time and plotting
# the jacobi radii itself overtime.
```

In [8]:
```
# numpy provides powerful multi-dimensional arrays
import numpy as np
# units
import astropy.units as u
# import previous HW functions
from ReadFile import Read
from CenterOfMass import CenterOfMass
import matplotlib.pyplot as plt
import os
```

In [9]:
```
# Rj = r*(Msat/2/Mmw)**(1/3)
# This function is a modified function from Lab 4 from class
def jacobi_radius(m_sat,r,m_host):
    """ Function that determines the jacobi radius for a satellite
    on a circular orbit about a host, where the host
    is assumed to be an isothermal sphere halo

    Inputs:
        m_sat: Astropy quantity
            Mass of the satellite galaxy in Msun
        r: Astropy quantity
            Distance of the satellite from the host in kpc
        m_host: Astropy quantity
            Mass of the host galaxy in Msun within r in Msun

    Outputs:
        Rj: astropy quantity
            The radius at which a satellite can be disturbed by
            tidal forces of the host galaxy
    """
    Rj = r*(m_sat/(2*m_host))**(1/3)
    return Rj
```

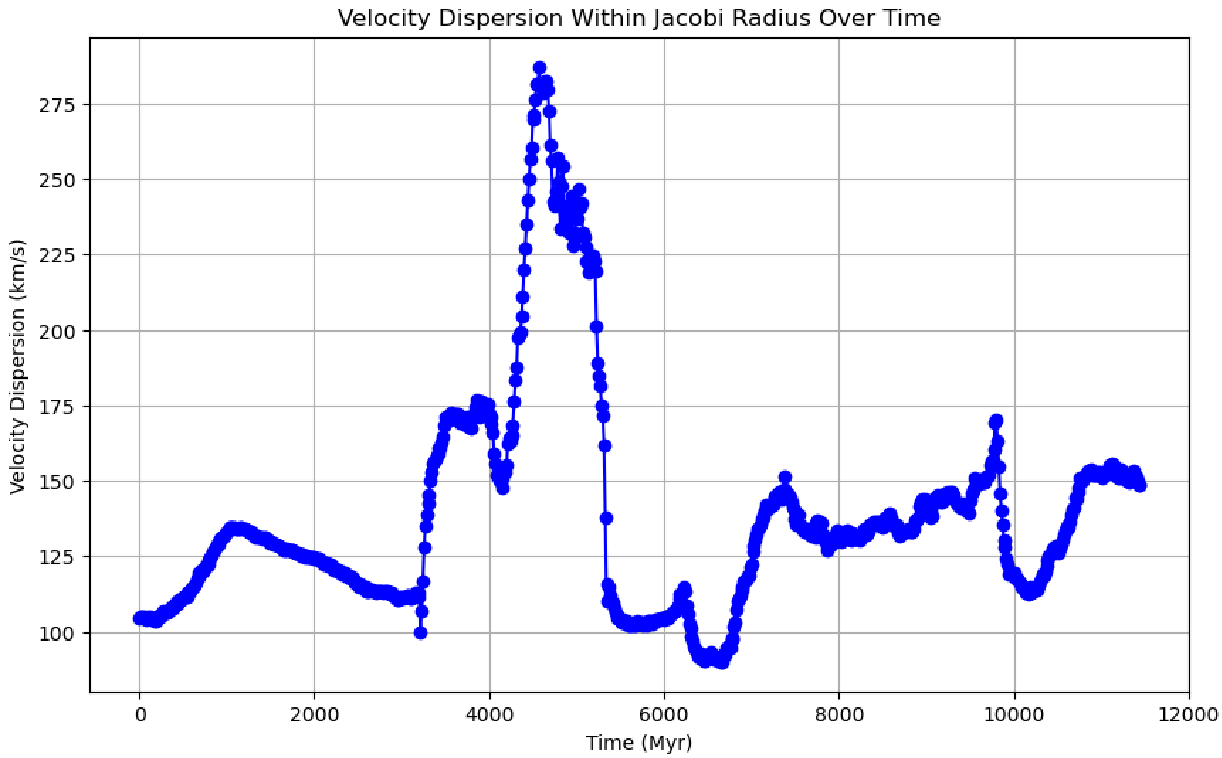
In [10]:
```
def velocity_dispersion_within_radius(vx, vy, vz, positions, r_jacobi, com_pos):
    """
    This function computes velocity dispersion for particles within r_jacobi.

    Inputs:
    vx, vy, vz (N,): vector
        Arrays of velocities in km/s
    positions:
        (N, 3) in kpc
    r_jacobi: scalar
        The jacobi radius in kpc
    com_pos: Astropy 3D array
        The center of mass position in kpc

    Outputs:
    sigma: scalar
        Velocity dispersion in km/s
    """
    # Shift positions to COM frame
    shifted_positions = positions - com_pos
```

```
    distances = np.linalg.norm(shifted_positions, axis=1)


    # Mask for particles within Jacobi radius
    mask = distances <= r_jacobi

    # Returns a nan value if the sum is zero (something went wrong)
    if np.sum(mask) == 0:
        return np.nan

    # Compute velocity dispersion
    selected_velocities = np.vstack((vx[mask], vy[mask], vz[mask])).T
    mean_velocity = np.mean(selected_velocities, axis=0)
    velocity_dispersions = selected_velocities - mean_velocity
    squared_speeds = np.sum(velocity_dispersions**2, axis=1)
    sigma = np.sqrt(np.mean(squared_speeds))  # km/s


    return sigma
```

In [11]:

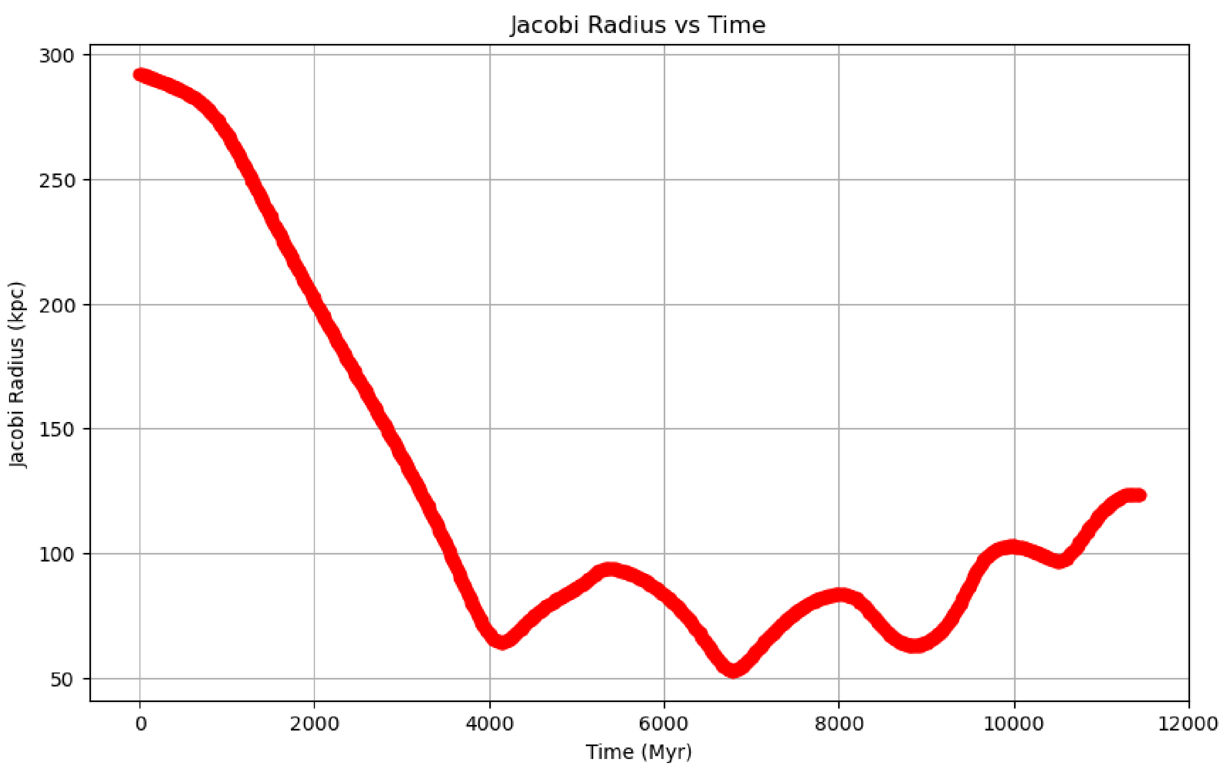
```
def plot_velocity_dispersion(start_num, end_num, m_sat, m_host, prefix="M33_", suffix=
    """
    This function calculates and plots the velocity dispersion of a satellite galaxy
    within its Jacobi radius over time using simulation snapshot files.

    Inputs:
        start_num: int
            Starting index of the snapshot files.
        end_num: int
            Ending index of the snapshot files.
        m_sat: Astropy quantity
            Mass of the satellite galaxy (e.g., M33) in solar masses.
        m_host: Astropy quantity
            Mass of the host galaxy (e.g., M31) in solar masses.
        prefix: str
            File name prefix (default is "M33_").
        suffix: str
            File extension (default is ".txt").
        folder: str
            Folder where the snapshot files are stored.

    Output:
        A plot showing how the velocity dispersion within the Jacobi radius evolves wi
    """
    # Define the number of steps
    num_steps = end_num - start_num + 1
    times = np.zeros(num_steps)
    dispersions = np.zeros(num_steps)

    # Loop over the range and fill the arrays
    for i in range(num_steps):
        filename = os.path.join(folder, f"{prefix}{start_num + i:03d}{suffix}")
        time, total, data = Read(filename)

        com = CenterOfMass(filename, ptype=1)
        com_pos = com.COMdefine(com.x, com.y, com.z, com.m)
        com_pos = np.array(com_pos) * u.kpc
        com_v = com.COMdefine(com.vx, com.vy, com.vz, com.m) * u.km / u.s
```

```
        r = np.linalg.norm(com_pos)
        r_jacobi = jacobi_radius(m_sat, r, m_host)

        positions = np.column_stack((data['x'], data['y'], data['z'])) * u.kpc
        positions_com = positions - com_pos

        # Gather velocities
        vx = data['vx'] * u.km / u.s
        vy = data['vy'] * u.km / u.s
        vz = data['vz'] * u.km / u.s

        # Subtract of com_v to get relative velocities
        selected_vx = (vx - com_v[0]).to_value(u.km / u.s)
        selected_vy = (vy - com_v[1]).to_value(u.km / u.s)
        selected_vz = (vz - com_v[2]).to_value(u.km / u.s)

        # Call the vel_dispersion function
        sigma = velocity_dispersion_within_radius(
            selected_vx,
            selected_vy,
            selected_vz,
            positions,
            r_jacobi,
            com_pos
        )

        # Store the values
        times[i] = time.value
        dispersions[i] = sigma

    # Plot valid data
    valid = ~np.isnan(dispersions)

    plt.figure(figsize=(10, 6))
    plt.plot(times[valid], dispersions[valid], marker='o', linestyle='-', color='b')
    plt.xlabel('Time (Myr)')
    plt.ylabel('Velocity Dispersion (km/s)')
    plt.title('Velocity Dispersion Within Jacobi Radius Over Time')
    plt.grid(True)
    plt.show()

plot_velocity_dispersion(start_num=0,end_num=801,m_sat=18.7e10 * u.Msun,m_host=192e10
```



In [6]:
```
def plot_jacobi_radius(start_num, end_num, m_sat, m_host, prefix="M33_", suffix=".txt'
    """
    Loops through multiple files with a specified prefix in a folder, calculates the J
    for the satellite galaxy, and plots the results.

    Inputs:
        start_num, end_num: int
            Range of file numbers
        m_sat: Astropy quantity
            Mass of the satellite galaxy in Msun
        m_host: Astropy quantity
            Mass of the host galaxy in Msun within r in Msun.
        prefix: str
            The prefix for the filename.
        suffix: str
            The type of file.
        folder: str
            Optional subfolder where files are located.

    Outputs:
        A plot of Jacobi radius over time.
    """
    # Number of time steps
    num_steps = end_num - start_num + 1

    # Fill empty numpy arrays
    times = np.zeros(num_steps)
    jacobi_radii = np.zeros(num_steps)

    # Loop through the file range
    for i in range(num_steps):
        filename = os.path.join(folder, f"{prefix}{start_num + i:03d}{suffix}")

        # Read the data file
        time, total, data = Read(filename)
```

```
        # Extract positions
        positions = np.column_stack((data['x'], data['y'], data['z']))  # in kpc

        # Compute the distance of the satellite from the host galaxy
        r = np.linalg.norm(np.mean(positions, axis=0)) * u.kpc

        # Calculate Jacobi radius
        r_jacobi = jacobi_radius(m_sat, r, m_host)

        # Store the values directly in the arrays
        times[i] = time.value  # Time in Myr
        jacobi_radii[i] = r_jacobi.value  # Jacobi radius in kpc

    # Filter out any NaN values
    valid_indices = ~np.isnan(jacobi_radii)

    # Plotting the results
    plt.figure(figsize=(10, 6))
    plt.plot(times[valid_indices], jacobi_radii[valid_indices], marker='o', linestyle=
    plt.xlabel('Time (Myr)')
    plt.ylabel('Jacobi Radius (kpc)')
    plt.title('Jacobi Radius vs Time')
    plt.grid(True)
    plt.show()

start_num = 0
end_num = 801
m_sat = 18.7e10 * u.Msun
m_host = 192e10 * u.Msun

plot_jacobi_radius(start_num, end_num, m_sat, m_host, folder="M33")
```



In [7]:
```
# Take a effective mass between M31 and MW for the future
```

```
In [ ]:
```