Problem Sheet 4 - Exercise 1

Part (b)

The solutions of this exercise are based on the predefined template that was provided on the website. In the following, you can find the modifications that I performed to obtain the results shown below.

```
  for iz in range(0,2):
    for iy in range(0,2):
      for ix in range(0,2):
        id = self.nodelist[nodeid].childrenids[ix,iy,iz]
        cm, m = self.computemultipoles(id)
        if m != 0: # Ignore cells with mass 0 as they do not contribute
          mass = mass + m
          for i in range(0,3):
            cenm[i] = (m * cm[i] + (mass - m) * cenm[i])/mass
```

Code for the computation of the center of mass and mass in the tree algorithm.

```
if angle < anglemax or self.nodelist[nodeid].leaf == 1:
  #
  # This is a small enough branch or it is a leaf.
  # In either case just use the node's cm and mass.
  # But avoid self-attraction!
  # Particle mass is 1.0
  distance = (self.nodelist[nodeid].cm[0] - self.particles[pid][0])**2 \
           + (self.nodelist[nodeid].cm[1] - self.particles[pid][1])**2 \
           + (self.nodelist[nodeid].cm[2] - self.particles[pid][2])**2
  # avoiding self-attraction
  if distance != 0:
    for i in range(0, 3):
      force[i] = self.nodelist[nodeid].mass * \
      np.power(distance + 0.001**2, -3/2) \
      * (self.particles[pid][i] - self.nodelist[nodeid].cm[i])
```

Code for the computation of the partial force on the particle.

```
def exact_force(self, pid):
  force = [0., 0., 0.]
  for id in range(0, len(self.particles)):
  # avoid self-interaction:
  if id != pid:
    dist = np.power((self.particles[pid][0] - self.particles[id][0]) ** 2
        + (self.particles[pid][1] - self.particles[id][1]) ** 2
        + (self.particles[pid][2] - self.particles[id][2]) ** 2
        + 0.001 ** 2, -3/2)
    for j in range(0, 3):
      force[j] = force[j] + dist * (self.particles[pid][j] - self.particles
          [id][j])
  return force
```

Code for the computation of the exact forces on a particle.

```
def all_exact_forces(self):
  nrp = len(self.particles)
  for pid in range(nrp):
    self.forces[pid][:] = self.exact_force(pid)
```

Code for the computation of the exact forces on a particle.

Part (c)

The following piece shows how the average arithmetic error $\langle \eta \rangle$ is calculated.

```
def distance(particle1, particle2):
  squared_dist = (particle1[0] - particle2[0]) ** 2 \
          + (particle1[1] - particle2[1]) ** 2 \
          + (particle1[2] - particle2[2]) ** 2
  return np.sqrt(squared_dist)


for i in range(0, len(particles)):
  avgerror = avgerror + distance(fapprox[i], fexact[i]) \
        / distance(fexact[i], [0., 0., 0.])

avgerror = avgerror / nparticles
print("The average error was computed to be:", avgerror)
```

Code for the computation of the relative average error.

The algorithm to count the number of node-particle interactions per particle was written in a way that it only counts interactions with nodes with non-zero mass. See the attached code for further information (look at the function gforce).

Part (d)

The following tables show the computation time, average arithmetic error and average nodes per particle for several numbers of particles and different opening angles.

Time [s]	0.2	0.4	0.8	Brute Force
5000	298.61	79.62	17.30	131.78
10000	773.83	187.43	38.39	526.40
20000	1941.86	434.21	84.68	2097.46
40000	4789.59	1001.59	187.85	8297.63

Table 1: Calculation time for different number of particles and opening angles in comparison to the brute force algorithm. One can easily see the logarithmic scaling for the tree algorithm and the quadratic scaling in the brute-force case.

Interactions [1]	0.2	0.4	0.8
5000	1886.8	578.3	133.6
10000	2603.9	705.5	152.7
20000	3480.8	849.4	173.73
40000	4433.9	997.220225	194.6

Table 2: Average Number of interactions for different number of particles and opening angles.

η [1]	0.2	0.4	0.8
5000	0.0003	0.0029	0.0154
10000	0.0003	0.0025	0.0131
20000	0.0003	0.0022	0.0112
40000	0.0003	0.0022	0.0100

Table 3: Average relative error per particle for different number of particles and opening angles in comparison to the brute force algorithm.

Part (e)

The plot of the data points and fit function is displayed in Figure 1. I decided to fit a function of the type $y = a \cdot x^b$ to the given function with the parameters $a = 0.0027$ and $b = 1.209$. From this, the estimated computation time for 10^{10} particles is approximately 105.3 Years.

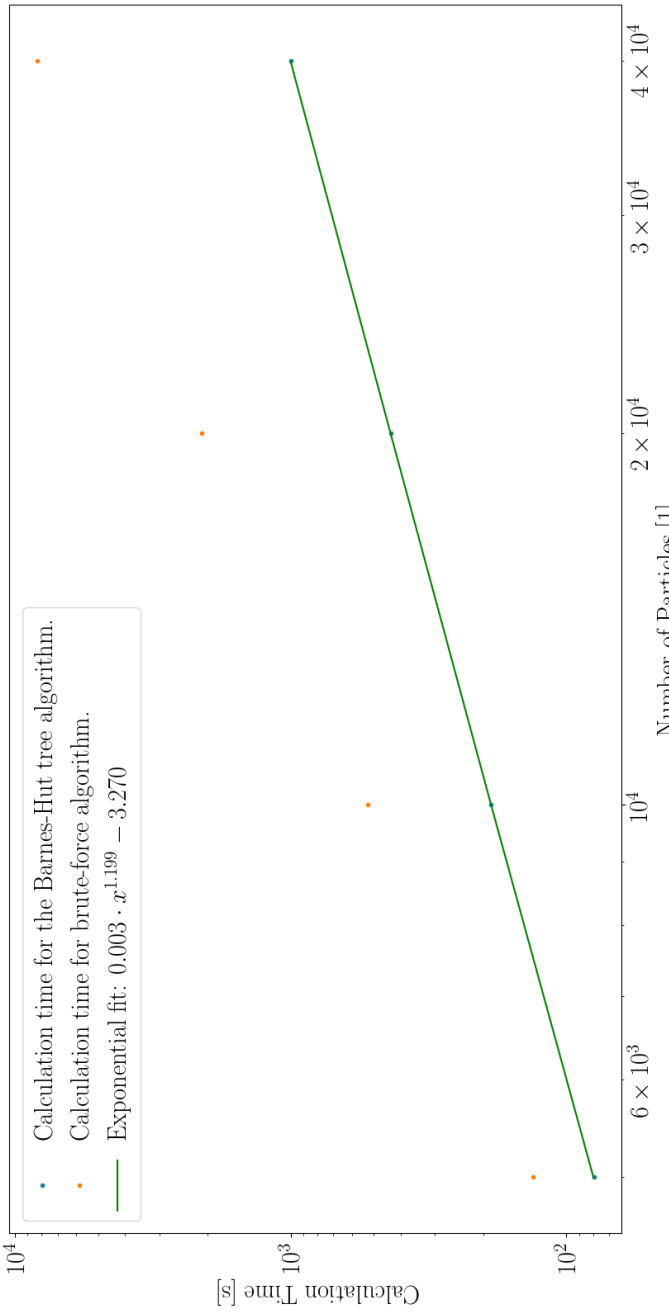

Figure 1: Plot of the calculation time for the brute force and the tree algorithm with opening angle $\theta = 0.8$