N-Body Simulation

Submitted by:

Weijia Sun

Zhongze Tang

Zichen Zhu

Team Project Number: 2

Professor:
Shantenu Jha

May 5, 2018

Electrical and Computer Engineering Department

Rutgers University, Piscataway, NJ 08854

1. Introduction

In this project, we introduce the N-Body Simulation and two methods to implement it, brute force and Barnes-Hut algorithm. We analyze the principle, time complexity and space complexity of these two algorithms. Then we do an experiment to test and verify our theory.

2. Background

In physics and astronomy, an N-body simulation is a simulation of a dynamical system of particles, usually under the influence of physical forces, such as gravity. Solving this problem has been motivated by the desire to understand the motions of the Sun, Moon, planets and the visible stars. In the 20th century, Solving the problem of dynamics of globular cluster star system has become a research hotspot. The N-Body problem is even more challenging to solve in general relativity. N-body simulations are widely used tools in cosmology, semiconductors, and fluid dynamics to study complex physical systems. In astrophysics, N-body simulations can be used from investigating the dynamics of few-body systems like the Earth-Moon-Sun system to understanding the evolution of the large-scale structure of the universe. Scientists also apply the same techniques to other pairwise interactions including Coulombic, Biot-Savart, and van der Waals. Direct N-body simulations are used to study the dynamical evolution of star clusters

The classic N-Body problem can be informally stated as:

Given the quasi-steady orbital properties (instantaneous position, velocity and time)[3] of a group of celestial bodies, predict their interactive forces; and consequently, predict their true orbital motions for all future times

3. Algorithms Employed

3.1. Brute Force Method

Basically, N-Body Simulation is an event-driven simulation. Our project mainly base on three physics principles:

- **Pairwise force**: Newton's law of universal gravitation asserts that the strength of the gravitational force between two particles is given by the product of their masses divided by the square of the distance between them, scaled by the gravitational constant G ($6.67 \times 10^{-11}\, Nm^2/kg^2$). The pull of one particle towards another acts on the line between them. Since we are using Cartesian coordinates to represent the position of a particle, it is convenient to break up the force into its x- and y-components (F_x, F_y) as illustrated below

$$\vec{F_i} = -\sum_{j \neq i} \frac{G m_i m_j \left(\vec{r_i} - \vec{r_j}\right)}{\left(\left|\vec{r_i} - \vec{r_j}\right|^2 + \epsilon^2\right)^{3/2}}$$

- **Net force**: The principle of superposition says that the net force acting on a particle in the x- or y-direction is the sum of the pairwise forces acting on the particle in that direction
- **Acceleration**: Newton's second law of motion postulates that the accelerations in the x- and y-directions are given by: $a_x = F_x/m$, $a_y = F_y/m$

A naive solution is to use the all-pairs approach to N-Body simulation, which is a brute-force method that goes through all pair-wise interactions among N bodies.

3.2. Leapfrog Integration

We use the leapfrog finite difference approximation scheme to integrate the above equations numerically: this is the basis for most astrophysical simulations of gravitational systems. In the leapfrog scheme, we discretize time and update the time variable t in increments of the time quantum Δt. We maintain the position (r_x, r_y) and velocity (v_x, v_y) of each particle at each time step. The steps below illustrate how to evolve the positions and velocities of the particles.

1. For each particle: Calculate the net force (F_x, F_y) at the current time t acting on that particle using Newton's law of gravitation and the principle of superposition. Note that force is a vector (i.e., it has direction). In particular, be aware that Δx and Δy are signed (positive or negative). In the diagram above, when you compute the force the sun exerts on the earth, the sun is pulling the earth up (Δy positive) and to the right (Δx positive).
2. For each particle
    a. Calculate its acceleration (a_x, a_y) at time t using the net force computed in Step 1 and Newton's second law of motion: $a_x = F_x/m$, $a_y = F_y/m$.

b. Calculate its new velocity (v_x, v_y) at the next time step by using the acceleration computed in Step 2a and the velocity from the old time step: Assuming the acceleration remains constant in this interval, the new velocity is ($v_x + \Delta t \cdot a_x$, $v_y + \Delta t \cdot a_y$).

c. Calculate its new position (r_x, r_y) at time $t + \Delta t$ by using the velocity computed in Step 2b and its old position at time t: Assuming the velocity remains constant in this interval, the new position is ($r_x + \Delta t \cdot v_x$, $r_y + \Delta t \cdot v_y$).

3. For each particle: Draw it using the position computed in Step 2.

The leapfrog method is more stable for integrating Hamiltonian systems than conventional numerical methods like Euler's method or Runge-Kutta. The leapfrog method is symplectic, which means it preserves properties specific to Hamiltonian systems (conservation of linear and angular momentum, time-reversibility, and conservation of energy of the discrete Hamiltonian). In contrast, ordinary numerical methods become dissipative and exhibit qualitatively different long-term behavior. For these reasons, symplectic methods are extremely popular for N-body simulation in practice.

The simulation can be more accurate as the Δt get smaller, but this comes with the price of computation.

3.3. Barnes-Hut Algorithm

The above we use the brute force method to simulate the motion of N bodies. However, the price of computation significantly increased with the growth of the number of bodies. Basically, the time complexity of brute search is O(N^2). This cannot be used in N-Body simulation when N comes to a large number. So we have to come up with an advanced algorithm.

Barnes-Hut algorithm can simulate the N-Body problem in O(NlogN) time and animate the motion of entire galaxy. It significantly reduces the price of computation with only a little loss of accuracy. The Barnes-Hut algorithm is the most popular algorithm used for N-Body simulation among the scientists.

The basic idea of Barnes-Hut algorithm is to group nearby bodies and treat them as a single body to speeding up the brute force N-Body algorithm. If each group is sufficiently far away from each other, we

can approximate its gravitational effects by using its center of mass. The center of mass of a group is the average position of a body in the group, weighted by mass. The formula for calculating the center of mass is as follows:

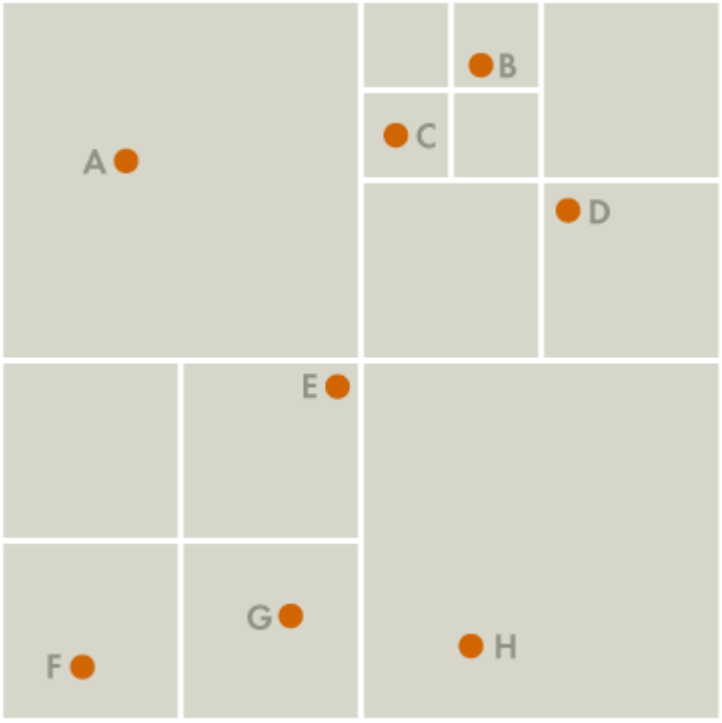
$$\mathrm{m} = m_1 + m_2$$

$$\mathrm{x} = (x_1 \cdot m_1 + x_2 \cdot m_2)/m$$

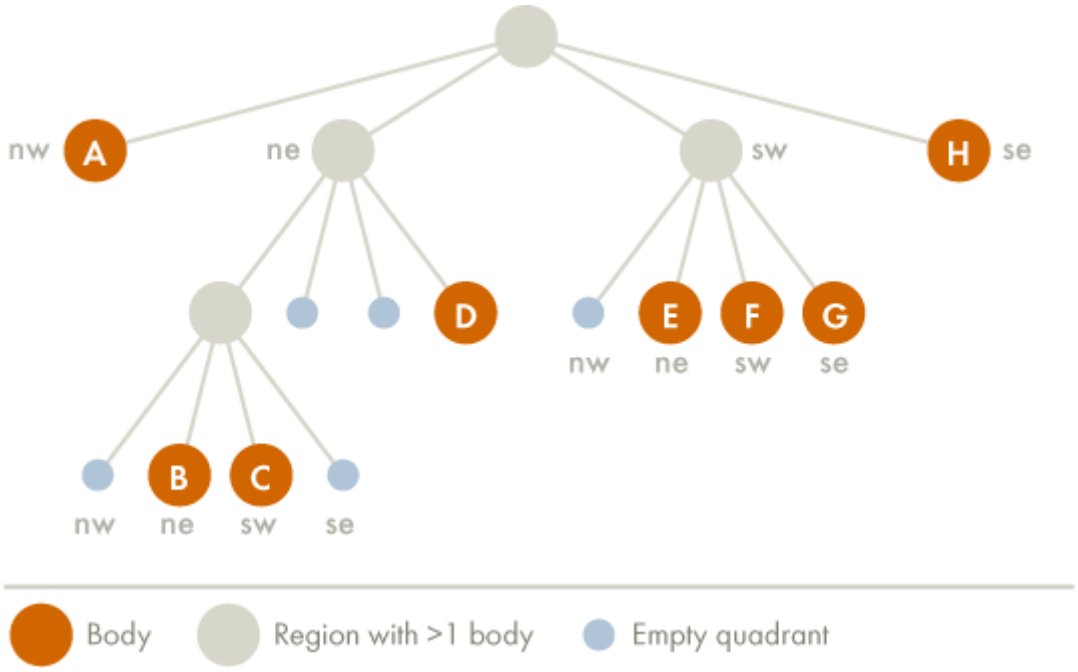
$$\mathrm{y} = (y_1 \cdot m_1 + y_2 \cdot m_2)/m$$

The Barnes-Hut algorithm groups the bodies which are close enough together. It divides the set of bodies into the group by storing them in a so-called quad-tree. Quad-tree is similar to a binary tree or a multi-branch tree. Each node in quad-tree has four children, some of which may be empty. Each node is a quadrant of the parent area.

The topmost node represents the whole area; each child node represents the corresponding quadrants of the area. As shown in the figure below, The quad-tree recursively subdivide the area into quadrants until each subdivision contains 0 or 1 body.



The external node means a single body. The internal node means the group of bodies under this node, and store the center-of-mass and the total mass of all its child node. Here is one example for one quad-tree with eight bodies:



To calculate the net force on one body, we should start from the root node, traversing the node of the tree. If the center-of-mass of the specific node is sufficiently far away from the body, we can treat the bodies under that node as one whole group, whose position is the center-of-mass and whose mass is the group's total mass. The algorithm is optimized algorithm since we do not need to traverse through all the node in the universe.

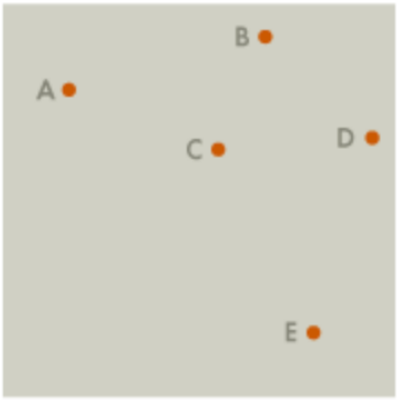
If the internal node is not sufficiently far away, we should traverse each of node in its' subtrees recursively. To determine whether it is sufficiently far away, we can use the quotient s/d as a standard. s means the width of the area represented by the internal node, and d is the distance between two bodies' center of mass. Second, compare s/d against the threshold θ, normally 0.5. By adjusting the θ value, you can control the speed and the accuracy of the simulation here. If θ equals to 0, the algorithm will become the brute force.

3.3.1. Constructing the quad-tree

To construct the quad-tree, we have to insert all the bodies into the quad-tree. When inserting a body b into the quad-tree rooted at node x, we have to follow these steps:

1. If node x is null now, insert the new body b here.
2. If node x is an internal node, update the total mass and the center-of-mass of x and then insert the body b into the corresponding subdivision recursively.
3. If node x is an external node, assume it contains a body called c, then this area has two bodies b and c now. Subdivide the area then insert both b and c into the corresponding quadrant recursively. Due to b and c may still end up in the same quadrant, we should keep subdividing the region until the requirement is met. Finally, update the center-of-mass and total mass of x

As the following figures, here is an example for constructing a quad-tree with five bodies in it. For here, we use the convention that the branches represent northwest, northeast, southwest, and southeast quadrants from left to right correspondingly.

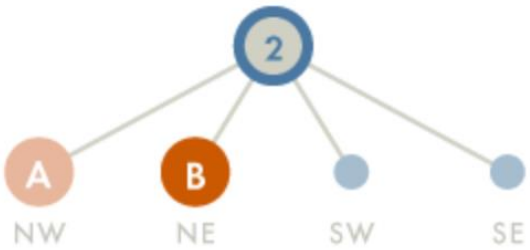

B
A
C
D
E

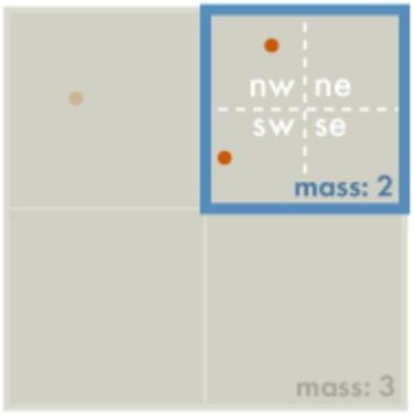

NW
NE
SW
SE

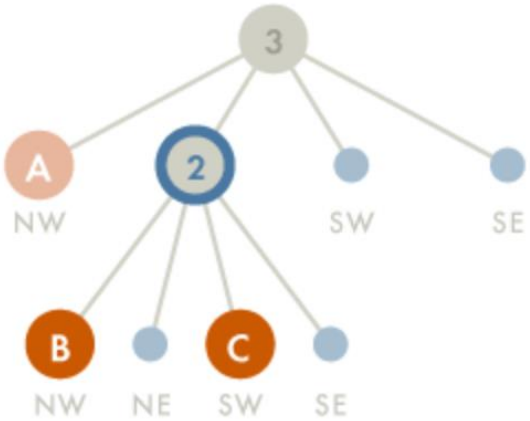

nw
ne
sw
se
mass: 1

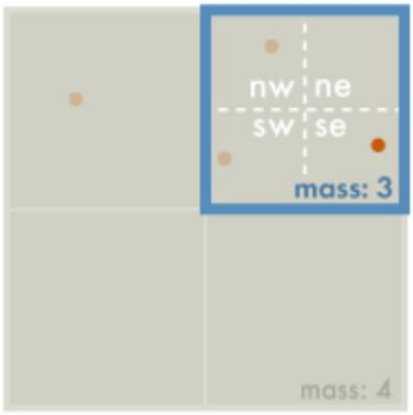

1
A
NW
NE
SW
SE

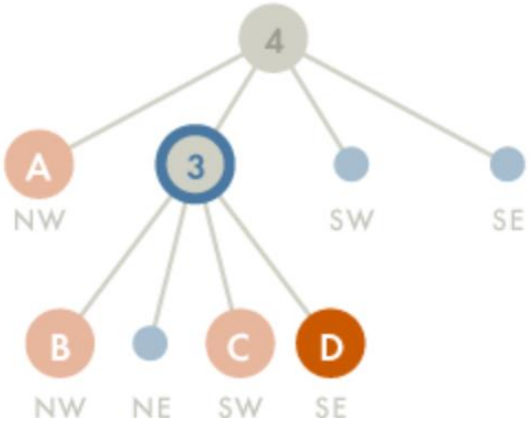

nw ne
sw se
mass: 2

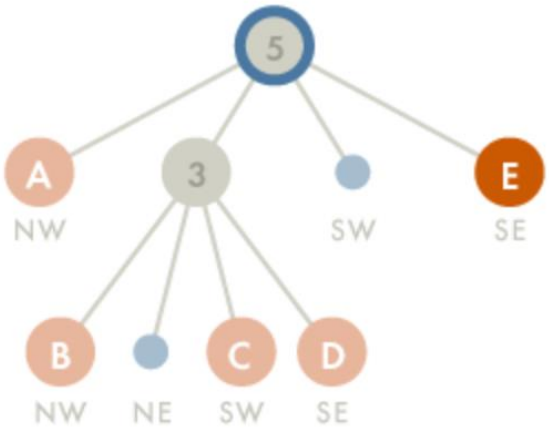

2
A
B
NW
NE
SW
SE


nw ne
sw se
mass: 2
mass: 3


3
A
2
NW
SW
SE
B
C
NW
NE
SW
SE


nw ne
sw se
mass: 3
mass: 4


4
A
3
NW
SW
SE
B
C
D
NW
NE
SW
SE



For this quad-tree, the root node contains the center-of-mass and total mass of all the bodies here. The internal node (mass equals to 3) contain the center-of-mass and total mass of the bodies b, c, and d.

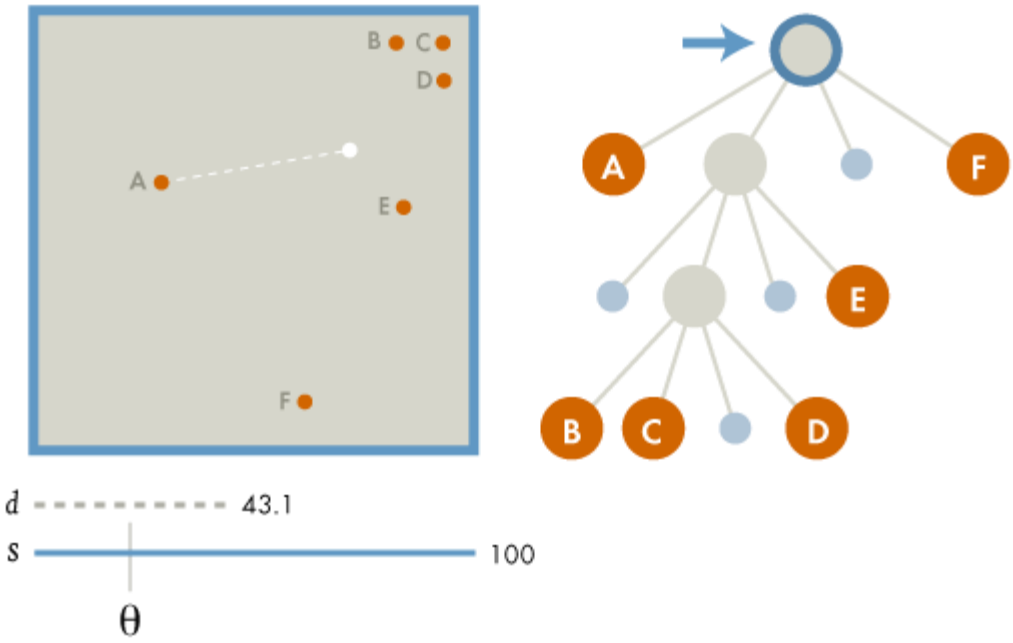
3.3.2. Calculating the force acting on a body

To calculate the net force acting on a body, say that b, follow the following procedure.

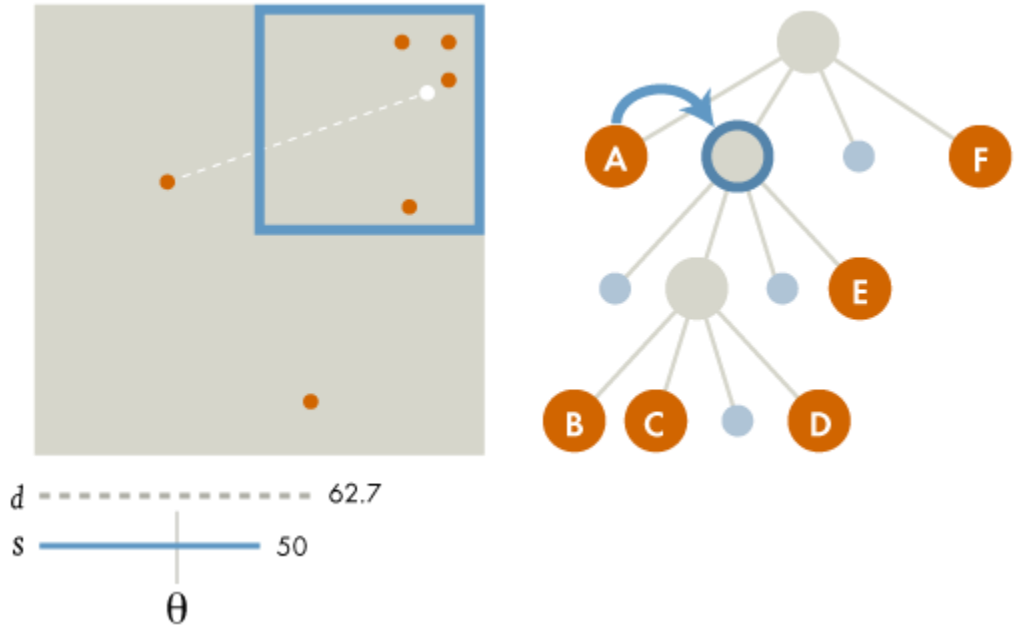
1. If the node is an external node, calculate the force exerted by the current node on b, and add it to the b's net force
2. Otherwise, calculate the quotient s/d. If s/d < θ, consider this internal node as a single body and calculate the force exerts on b. Add it to b's net force, too.
3. Otherwise, run the above procedure recursively on each of the current node's child node.

Here is an example of calculating the force acting on a body, we call it a here. This quad-tree has six bodies from a to f. which have masses 1, 2, 3, 4 ,5 and 6 kg respectively.

1. The algorithm starts from the root node. Comparing body A to the node's center-of-mass, the quotient $s/d = 100/43.1 > \theta = 0.5$, so we recursively perform the process on every child node.

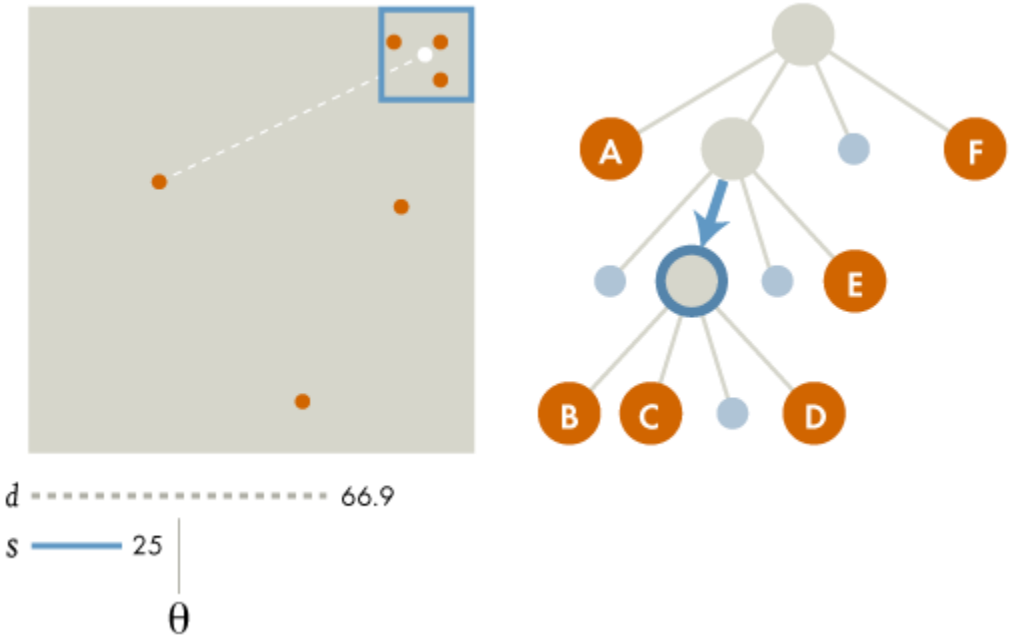


2. The first child is itself, doesn't have a force exerted on itself.
3. This is an internal node which represents the northeast quadrant of the whole area. Calculate the $s/d = 50/62.7 > \theta$, so we recursively perform the process on every child node.

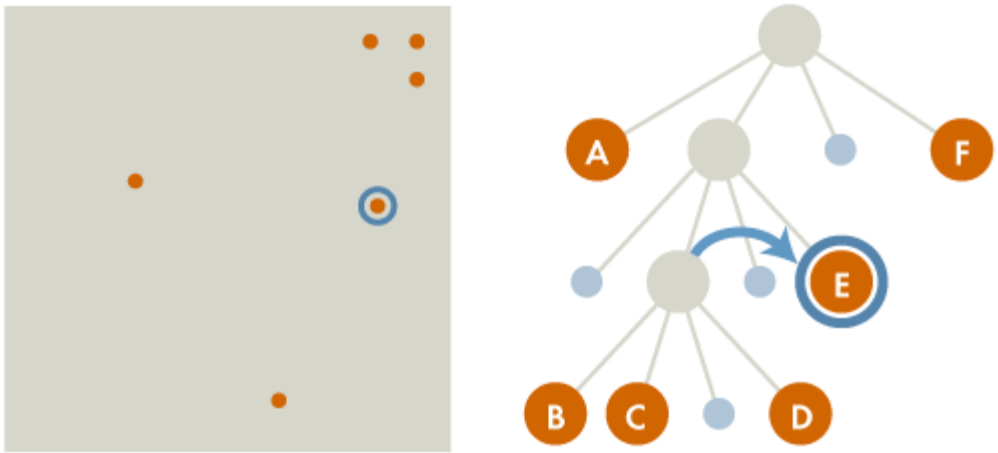


4. This is an internal node which representing the northeast quadrant of the parent node. It contains the body b, c, and d. Calculating $s/d = 25/66.9 < \theta$. So we consider the internal node as a single body. Now we should calculate the center-of-mass and the total mass of this internal node.

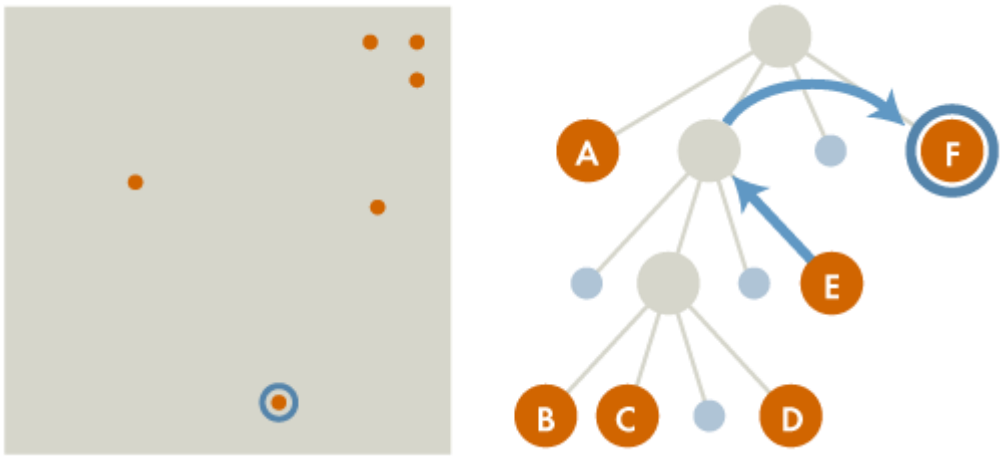
Then add the force exerted by this node on body a and add it to the net force of a. Last, Due to the parent of this node doesn't have more child node, we move to the sibling node of the parent node.



5. The sibling node is an external node which contains the body e, so we just calculate the pairwise force between a and e and add it to the net force of body a.



6. Since all the node has been examined at this level, we continually examine the node in the above level. Then we come to the external node which contains the body f, so we just calculate the pairwise force between a and e and add it to the net force of body a.



4. Experimental Configurations and Details

Suppose that all the bodies are in the same flat. (If in space, we have to use oct-tree, not quad-tree.). We create an animation for this project: Draw each particle at its current position and repeat this process at each time step. By showing this sequence frame by frame in a small time step, we can create the illusion of movement.

We have several test data, the number of bodies in the simulation range from 3 to 30000. The test file is a text file which the first row is the number of bodies in the file. The second row is a real number R which represents the radius of the universe. The following are N rows and each row contains eight values. The first two values are the x- and y-coordinates of the initial position; the second two values are the x- and y-components of the initial velocity; the third value is the mass; The last three value is the RGB color value of the corresponding body. The test file input format is as follow:

5

2.5E11

0.000E00 0.0 0.0 0.000E00 4.97250E41 255 255 0

5.790E10 0.0 0.0 2.395E10 8.25500E34 0 255 0

1.082E11 0.0 0.0 1.750E10 1.21725E36 255 0 255

1.496E11 0.0 0.0 1.490E10 1.49350E36 0 50 255
2.279E11 0.0 0.0 1.205E10 1.60475E35 255 0 0

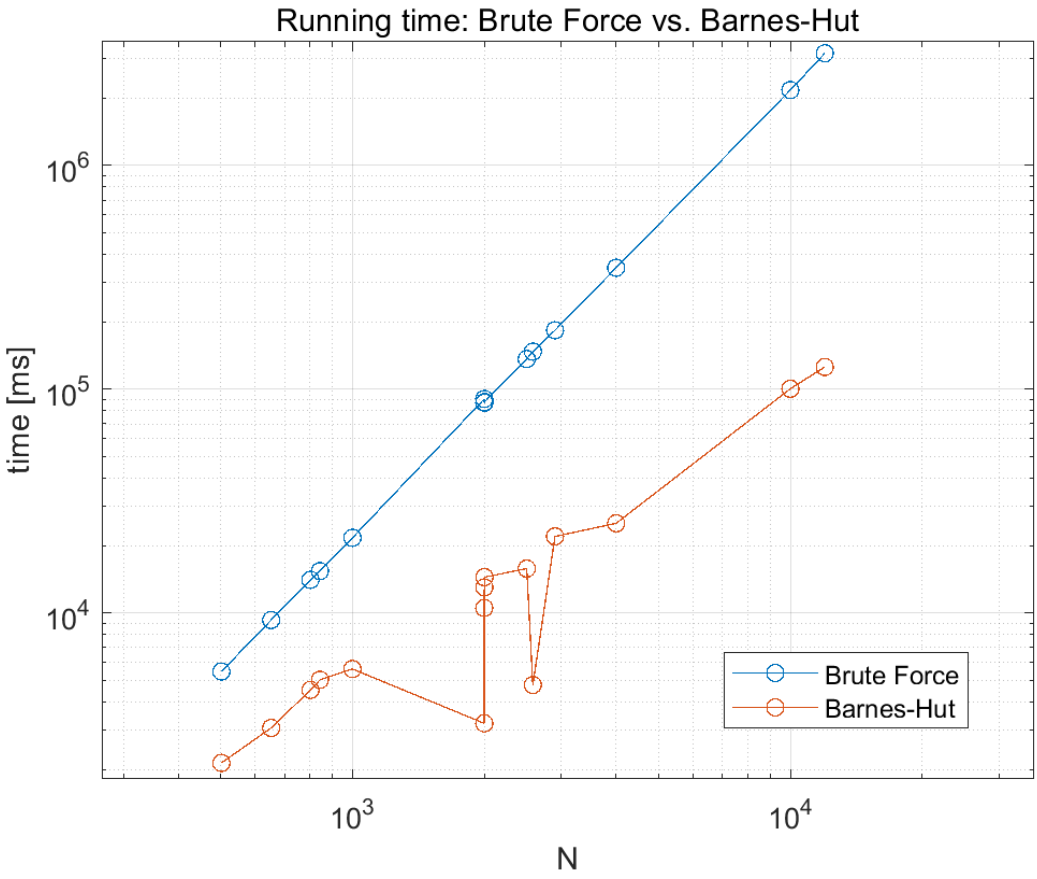
This is a file simulate the movement of a universe with 5 planets.

5. Result and Analysis

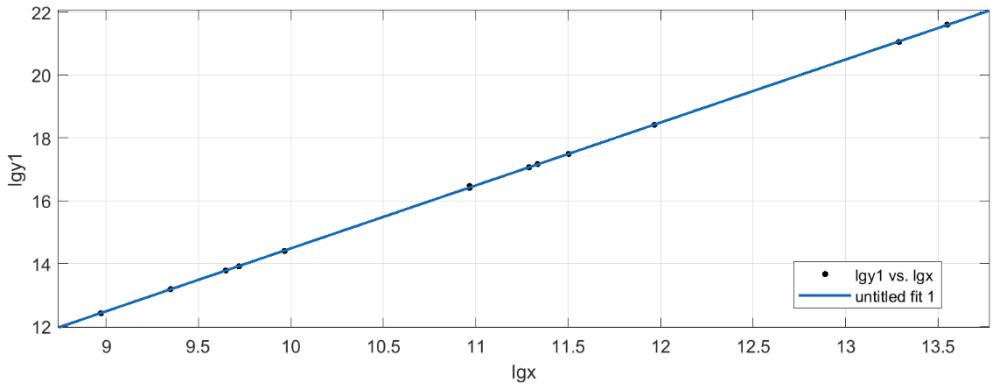
The time complexity of brute force algorithm is O(N^2), since for each body, you have to go through all the other bodies and calculate the pair-wise force between each other. For the optimized algorithm ---- Barnes-Hut algorithm's time complexity is O(NlogN). After you constructing a quad-tree by inserted all the node into it, you will get the quad-tree with average height O($log_2 N^{1/3}$) = O(logN), and the time required to construct the tree is of O(NlogN), because for each node , you might go from the root node to the leaf node. The final step of constructing the tree is to upgrade the subdivided nodes with the total mass and center-of-mass. By propagating information down the tree from the nodes towards the root, this step may also be accomplished in a time of O(NlogN). So the time complexity of the Barnes-Hut algorithm is O(NlogN).

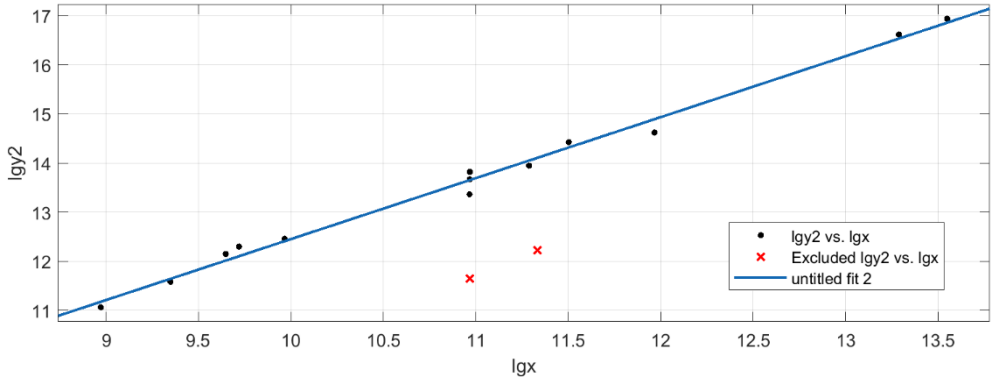
For the space complexity, it is difficult to determine a general size of the quad-tree. In the best case, each node will contain within itself a body, which is precisely a full 4-tree. The space complexity here is O(N). In the worst case, two bodies could be infinitely close enough, leading to a quad-tree with infinite height. Also, this situation is physically not allowed. So in this paper, we will not talk about space complexity.

The result of our experiment is shown as below:

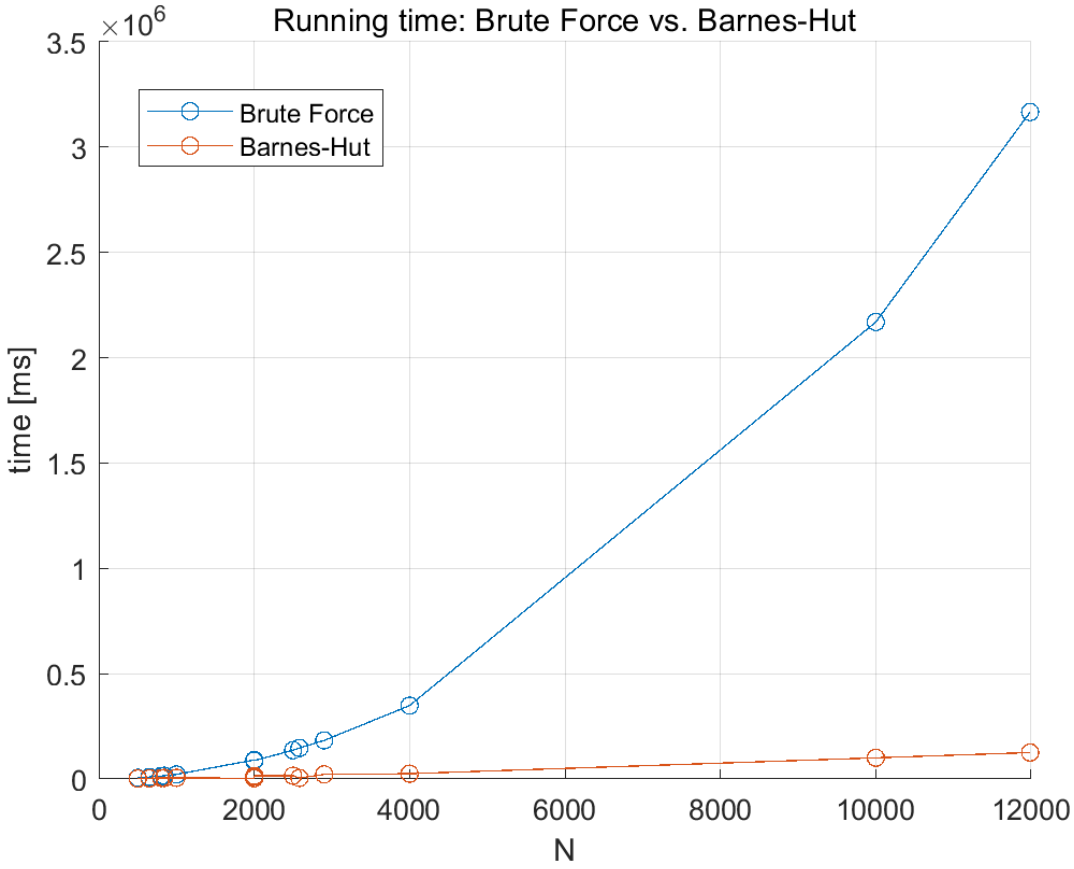


To show the result more clearly, we draw another two figures of brute-force algorithm and Barnes-Hut algorithm, respectively. Both x-axises are the log of N, while the y-axises are the log of running time (ms). The slope of the first line is 2, which indicates that the time complexity of brute-force algorithm is O(N^2). Moreover, the slope of the second line is 1.242, which indicates the time complexity of Barnes-Hut algorithm is O(NlogN), based on the knowledge we learned from the class.





Note that there are two invalid data points, we have removed them in the figure above (the two crosses). After removing these two points, we can get a new figure which shows the comparison of these two algorithms more clearly. Experimental conclusions meet our theoretical expectations.



6. Conclusion

In this project, we compare two algorithms for solving N-Body Simulation using datasets with sizes from 500 to 11K. Theoretically, the time complexity of Brute Force is $O(N^2)$ and Barnes-Hut is O(NlgN). We also analyze how those algorithms work and the data structures algorithms need. Finally, based on the result of experiments, our experiment proves that analysis of time complexity.