Consider that numAtoms is n,

```
public double calculateEnergy(double[][] coords, int numAtoms) {

    double energySum = 0.0;
    double r0 = 1.2;

    for (int i=0; i < numAtoms-1; i++) {
        for (int j=0; j < numAtoms; j++) {
            double distance = Math.sqrt( (coords[i][0] - coords[j][0]) * (coords[i][0] - coords[j][0]) +
                                         (coords[i][1] - coords[j][1]) * (coords[i][1] - coords[j][1]) +
                                         (coords[i][2] - coords[j][2]) * (coords[i][2] - coords[j][2]) );

            double partialTerm1cubed= (r0/distance) * (r0/distance)*(r0/distance);
            double term2 = partialTerm1cubed * partialTerm1cubed ;
            double term1 = term2 * term2;

            energySum = energySum + (term1 - 2.0 * term2);
        }
    }
    return energySum;

}
```

Every time this inner loop runs, it takes 26 ticks for all of the assignments and operators.

This loop will run 'n' times.

This loop will run 'n'-1' times.

Therefore, given numAtoms, the program shall execute $(n \times (n-1))$ times or (n^2-n) times while costing 26 ticks for each time it's ran, or $26(n^2-n)$ ticks of the clock.

In terms of big-O signature of this algorithm, this algorithm has a time complexity of $O(n^2)$.