Dynamic Programming Project

Ladybug & the Miraculous

- Student Name: Amna Khdair آمنة عبدالرحيم خضير خضير
- Student ID: 11819646.
- Section Number: 1.
- Instructor Name: Dr.Samer Arandi.

➢ ***Part One: Divide & Conquer:***

✓ **Define the value returned by the function f which we want to optimize?**

The function will return the ***Maximum of Energy***.

✓ **Define the parameters which f depends on?**

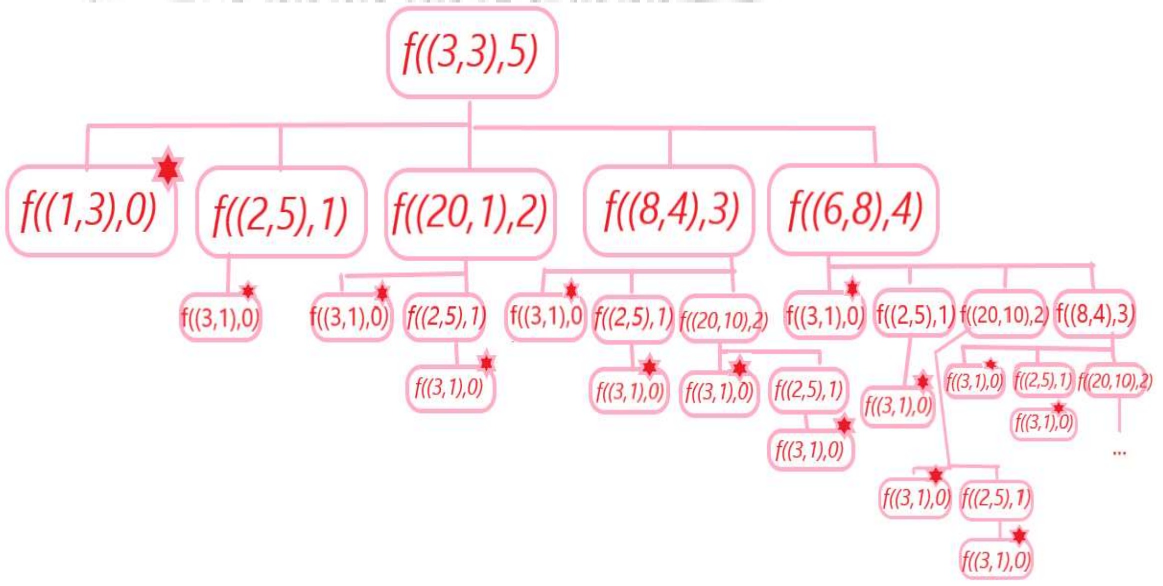
The parameter of the function that I choose is the endpoint that ladybug want to meet the Hawkmoth there, this parameter achieves all the factor to putt as a parameter because it describes:

1- The size of the problem, if we assume the first point fixed as (1, 3), so we can know the problem **f ((3, 3), 5)** large than the problem **f ((8, 4,) 4)**.

2- It's unique because it's impossible to find two-point the same, also we don't need any additional information to solve the problem, or there is no missing information.

✓ **Draw the recursion tree for f?**

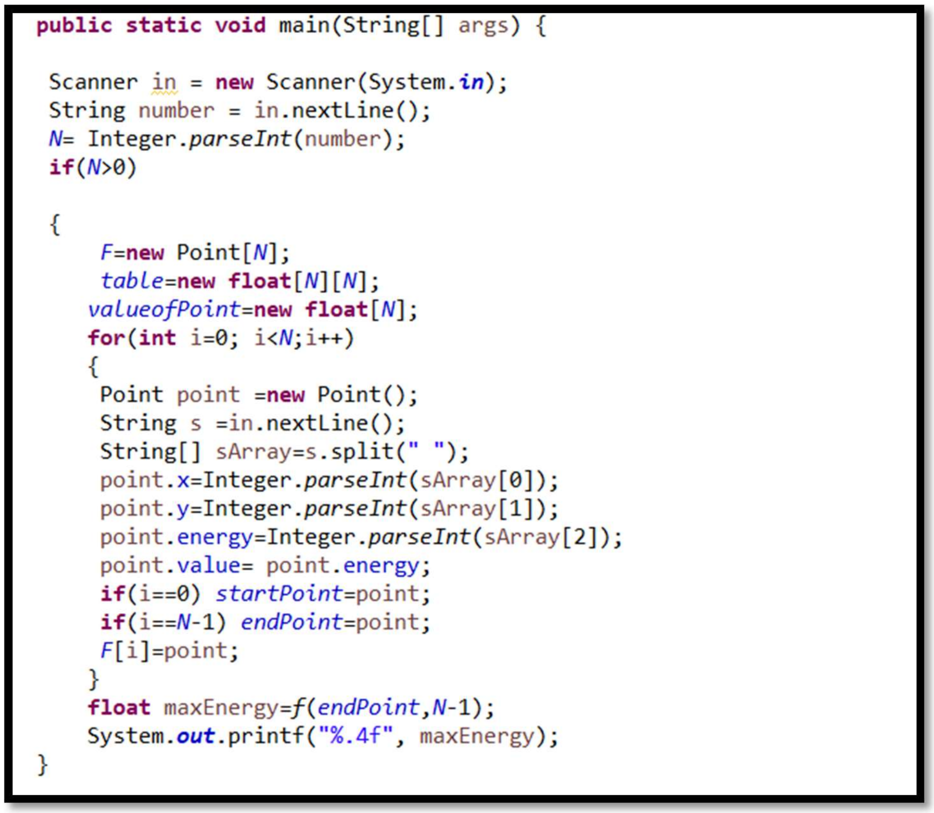
For example if we have these point (1,3),(2,5),(20,10),(8,4),(6,8),(3,3) ,So the tree will be like that:



✓ Write the recursive (divide and conquer) code to solve the question?

-Short brief about my code:

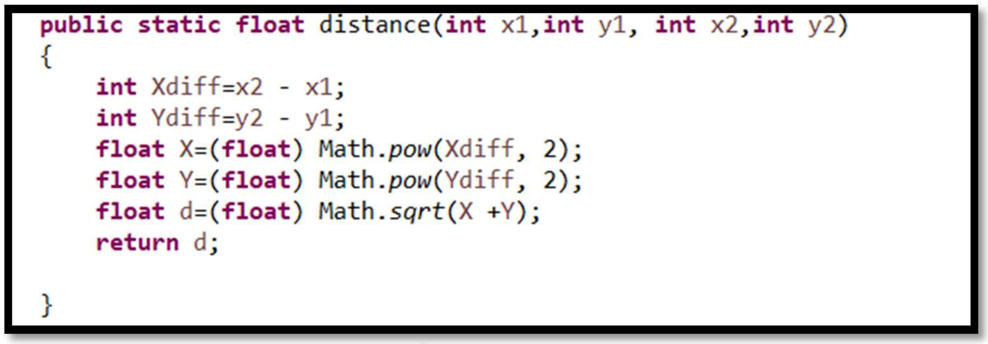
Main function: The first thing I read from the user is the number of miracles, the coordinates of each point with its energy, and then call the recursive function, to print the maximum number of energy her collected as shown in Fig.1:



```
public static void main(String[] args) {

 Scanner in = new Scanner(System.in);
 String number = in.nextLine();
 N= Integer.parseInt(number);
 if(N>0)

 {
     F=new Point[N];
     table=new float[N][N];
    valueofPoint=new float[N];
    for(int i=0; i<N;i++)
    {
     Point point =new Point();
     String s =in.nextLine();
     String[] sArray=s.split(" ");
     point.x=Integer.parseInt(sArray[0]);
     point.y=Integer.parseInt(sArray[1]);
     point.energy=Integer.parseInt(sArray[2]);
     point.value= point.energy;
     if(i==0) startPoint=point;
     if(i==N-1) endPoint=point;
     F[i]=point;
    }
    float maxEnergy=f(endPoint,N-1);
    System.out.printf("%.4f", maxEnergy);
}
```

Fig.1: Main function.

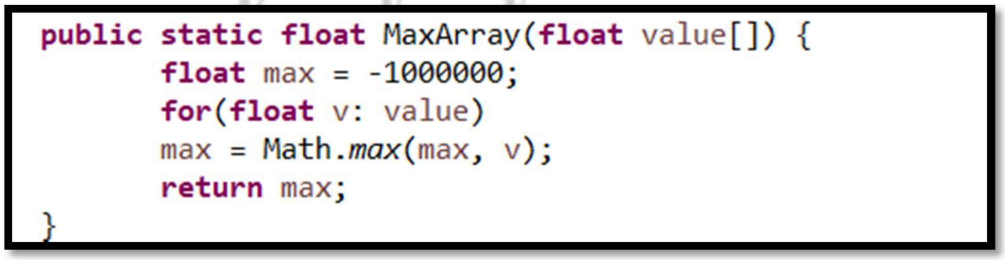
Distance function: this function to calculate the distance between two points from this equation: $\sqrt{(X2 - X1)^2 - (Y2 - Y1)^2}$, so the code as shown in Fig.2:



```
public static float distance(int x1,int y1, int x2,int y2)
{
    int Xdiff=x2 - x1;
    int Ydiff=y2 - y1;
    float X=(float) Math.pow(Xdiff, 2);
    float Y=(float) Math.pow(Ydiff, 2);
    float d=(float) Math.sqrt(X +Y);
    return d;

}
```

Fig.2: Distance between two points.

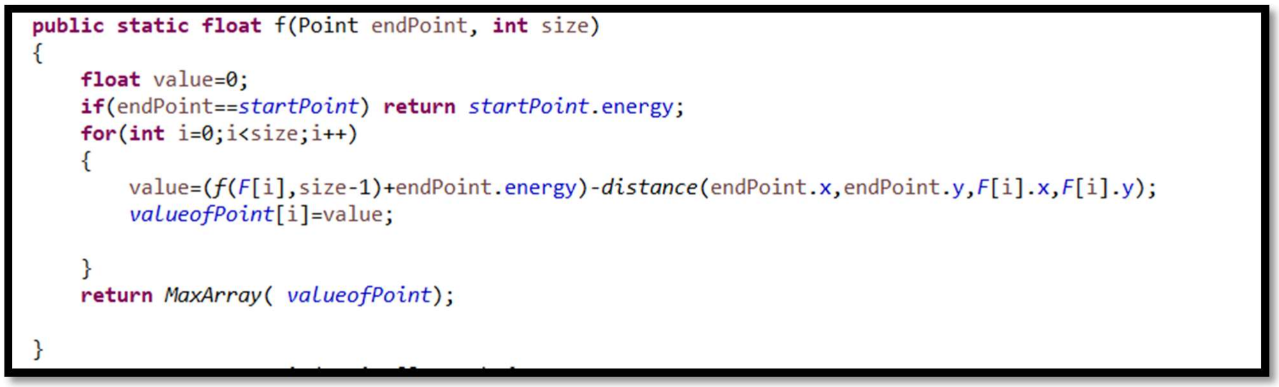
Max array function: to find the maximum value in whole array as shown in Fig.3:



```
public static float MaxArray(float value[]) {
        float max = -1000000;
        for(float v: value)
        max = Math.max(max, v);
        return max;
}
```

Fig.3: find maximum value in array.

Recursive function: to find the maximum energy that can ladybug collect, so I find all possibilities in each point by adding the energy then subtract the distance between them, then get the max value, as shown in Fig.4:

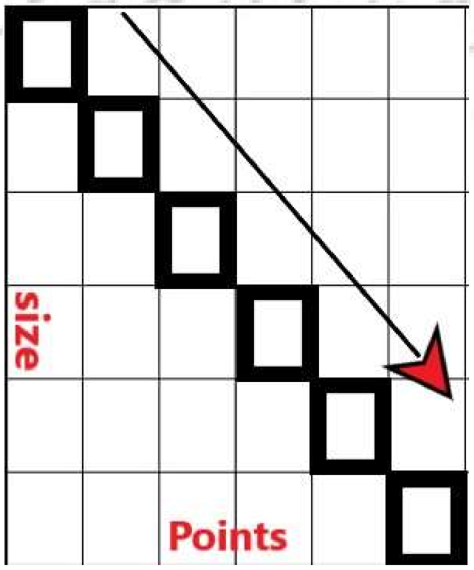


```
public static float f(Point endPoint, int size)
{
    float value=0;
    if(endPoint==startPoint) return startPoint.energy;
    for(int i=0;i<size;i++)
    {
        value=(f(F[i],size-1)+endPoint.energy)-distance(endPoint.x,endPoint.y,F[i].x,F[i].y);
        valueofPoint[i]=value;

    }
    return MaxArray( valueofPoint);

}
```

Fig.4: find the max energy by using recursive function.

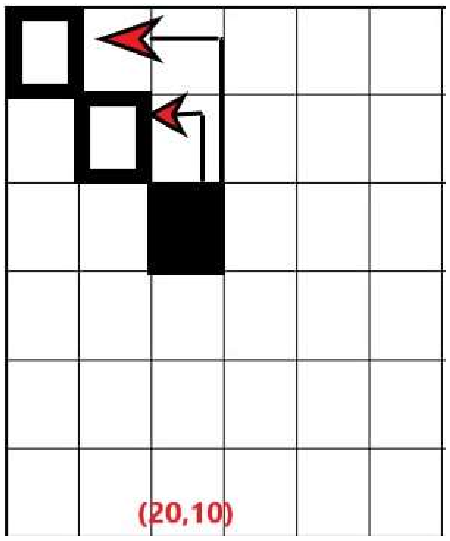
➢ Part Two: Dynamic Programming:

✓ Draw the table and determine the dependencies between the table cells and determine the direction of movement within the table?



The row of the table is a size of problem, and the columns is the points, And we move diagonally of the table. As shown in the example below:

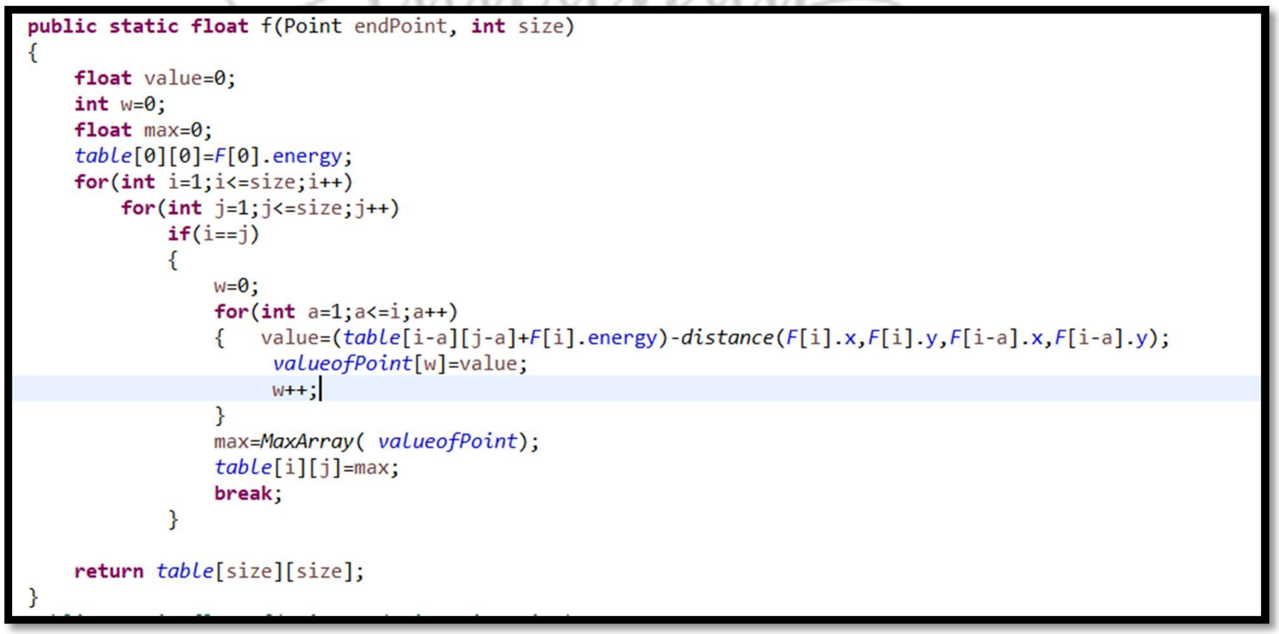
<u>EX:</u> if we want to find the problem f ((20, 10), 2) so the table will be like this:



It will compare the value of f (2, 5) and the value of f (1, 3) to take the max between them then added to her energy and subtract the distance to store the value in her.

✓ Write the Dynamic programming code which fills the table(s)?

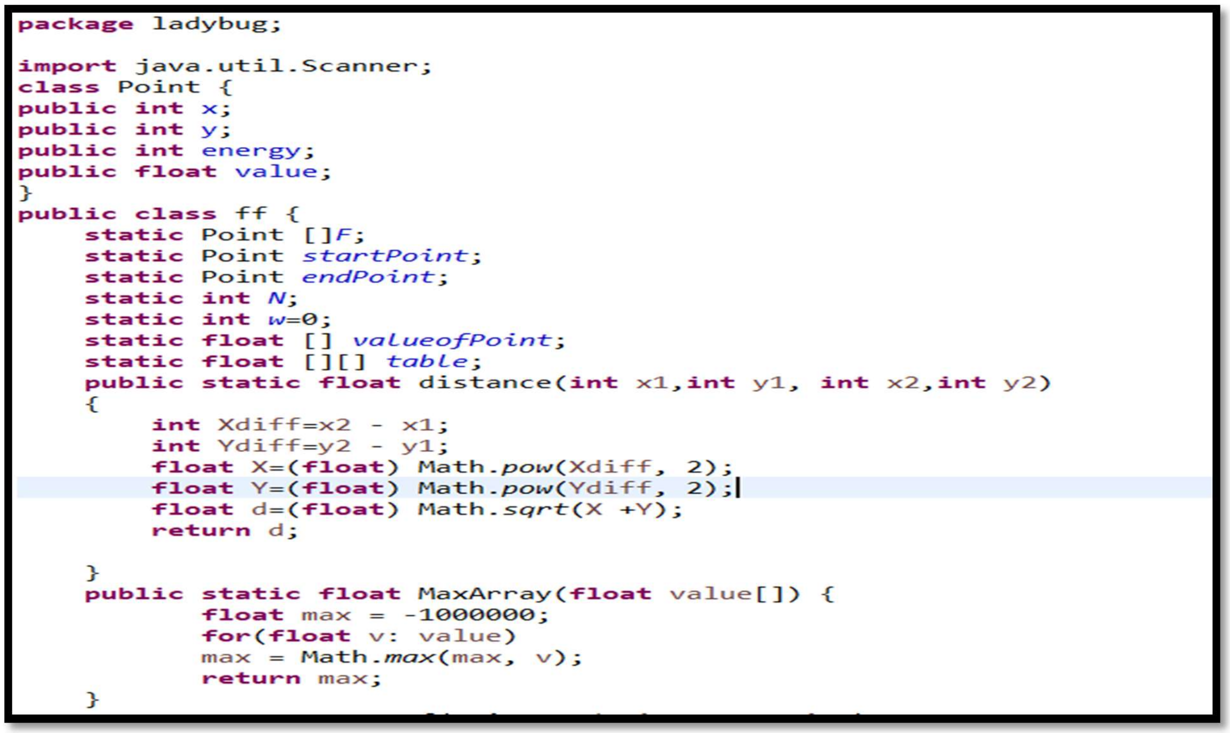
As I mentioned the table will fill diagonally as shown in Fig.5:



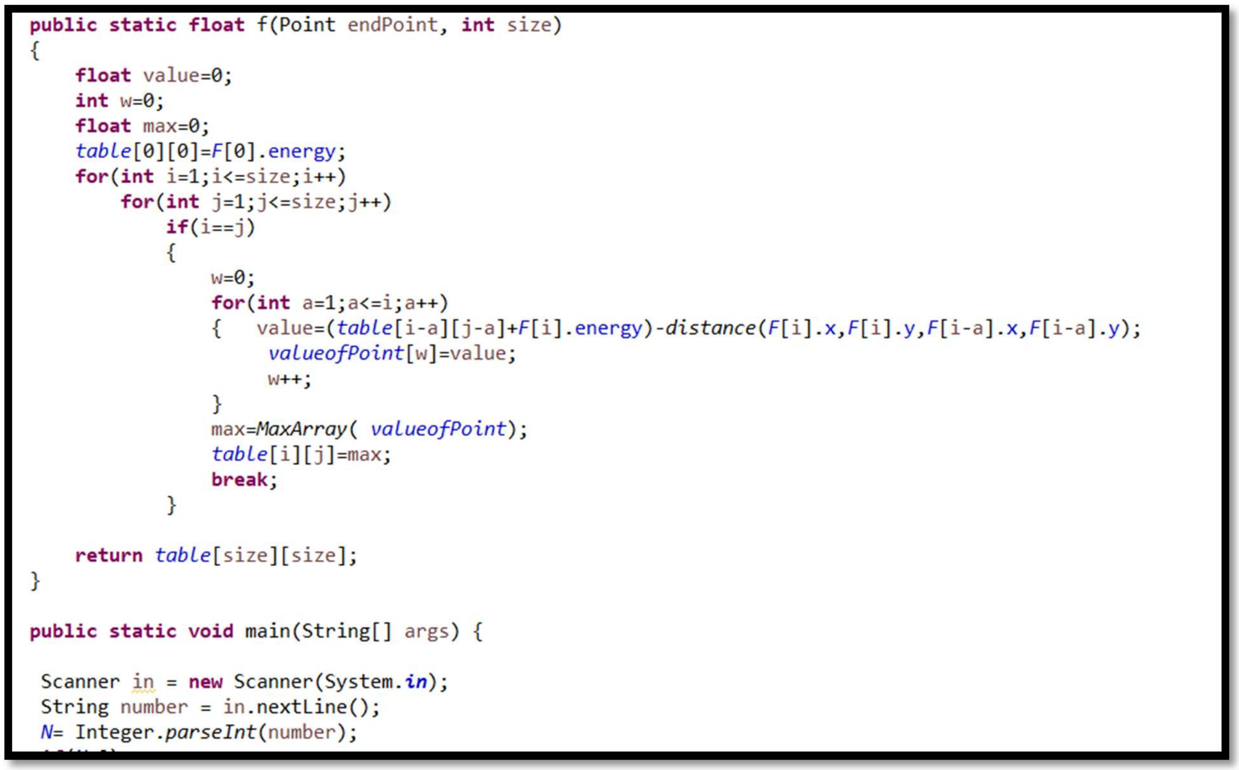
```
public static float f(Point endPoint, int size)
{
    float value=0;
    int w=0;
    float max=0;
    table[0][0]=F[0].energy;
    for(int i=1;i<=size;i++)
        for(int j=1;j<=size;j++)
            if(i==j)
            {
                w=0;
                for(int a=1;a<=i;a++)
                {   value=(table[i-a][j-a]+F[i].energy)-distance(F[i].x,F[i].y,F[i-a].x,F[i-a].y);
                     valueofPoint[w]=value;
                     w++;
                }
                max=MaxArray( valueofPoint);
                table[i][j]=max;
                break;
            }

    return table[size][size];
}
```

Fig.5: the function f that fill the table.

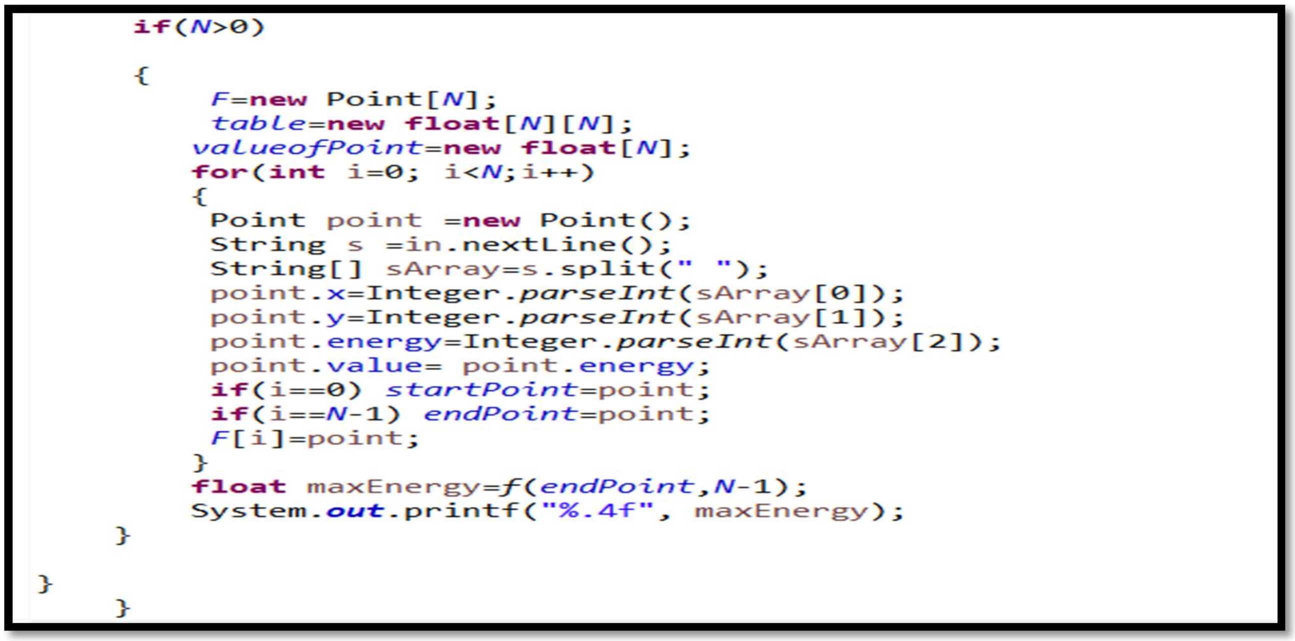
Write the code that will print the final answer?



```
package ladybug;

import java.util.Scanner;
class Point {
public int x;
public int y;
public int energy;
public float value;
}
public class ff {
    static Point []F;
    static Point startPoint;
    static Point endPoint;
    static int N;
    static int w=0;
    static float [] valueofPoint;
    static float [][] table;
    public static float distance(int x1,int y1, int x2,int y2)
    {
        int Xdiff=x2 - x1;
        int Ydiff=y2 - y1;
        float X=(float) Math.pow(Xdiff, 2);
        float Y=(float) Math.pow(Ydiff, 2);
        float d=(float) Math.sqrt(X +Y);
        return d;

    }
    public static float MaxArray(float value[]) {
          float max = -1000000;
          for(float v: value)
          max = Math.max(max, v);
          return max;
    }
```



```
public static float f(Point endPoint, int size)
{
    float value=0;
    int w=0;
    float max=0;
    table[0][0]=F[0].energy;
    for(int i=1;i<=size;i++)
        for(int j=1;j<=size;j++)
            if(i==j)
            {
                w=0;
                for(int a=1;a<=i;a++)
                {   value=(table[i-a][j-a]+F[i].energy)-distance(F[i].x,F[i].y,F[i-a].x,F[i-a].y);
                     valueofPoint[w]=value;
                     w++;
                }
                max=MaxArray( valueofPoint);
                table[i][j]=max;
                break;
            }

    return table[size][size];
}

public static void main(String[] args) {

 Scanner in = new Scanner(System.in);
 String number = in.nextLine();
 N= Integer.parseInt(number);
```



```
      if(N>0)

      {
           F=new Point[N];
           table=new float[N][N];
          valueofPoint=new float[N];
          for(int i=0; i<N;i++)
          {
           Point point =new Point();
           String s =in.nextLine();
           String[] sArray=s.split(" ");
           point.x=Integer.parseInt(sArray[0]);
           point.y=Integer.parseInt(sArray[1]);
           point.energy=Integer.parseInt(sArray[2]);
           point.value= point.energy;
           if(i==0) startPoint=point;
           if(i==N-1) endPoint=point;
           F[i]=point;
          }
          float maxEnergy=f(endPoint,N-1);
          System.out.printf("%.4f", maxEnergy);
     }

}
    }
```