

1. Closest Pairs

a.) *Brute Force Closest Pair*

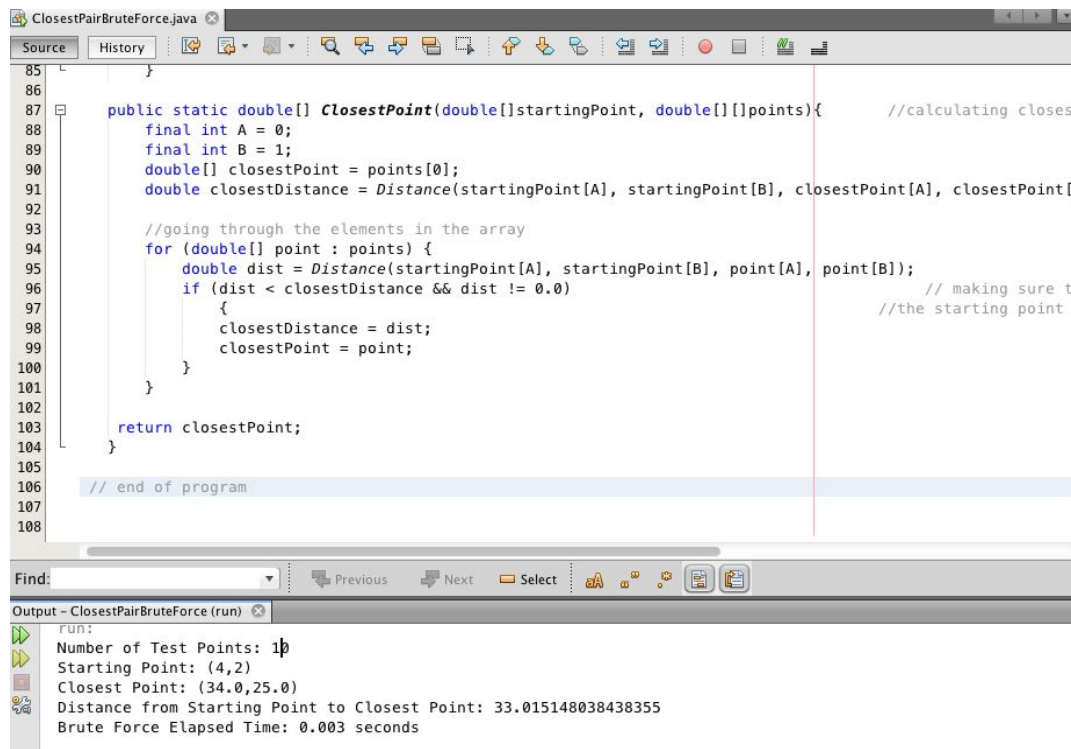
For the brute force method, we would compute all the distances between every pair of points and select the minimum distance. If we are at point n , then we have to find the distances between n and $n-1$ points. This applies to all n points, so the total distance comparisons between all points can be described as $n(n-1)$. In this calculation, we also double counted distances. For example, the distance between point A and B is the same as the distance between point B and A. Therefore, the actual number of comparisons would be $n(n-1)/2$. If we simplify the equation we get $O(n^2)$.

Overall we are will be implementing the following:

```
Closest_Pair(Points P, N)
{
    for i = 0; i < N-1; i++) {
        for (j=1+i; j < N; j++) {
            Get minimum distance(P[i], P[j]);
        }
    }
}
return Closest_Pair
```

Screenshots:

10 Points



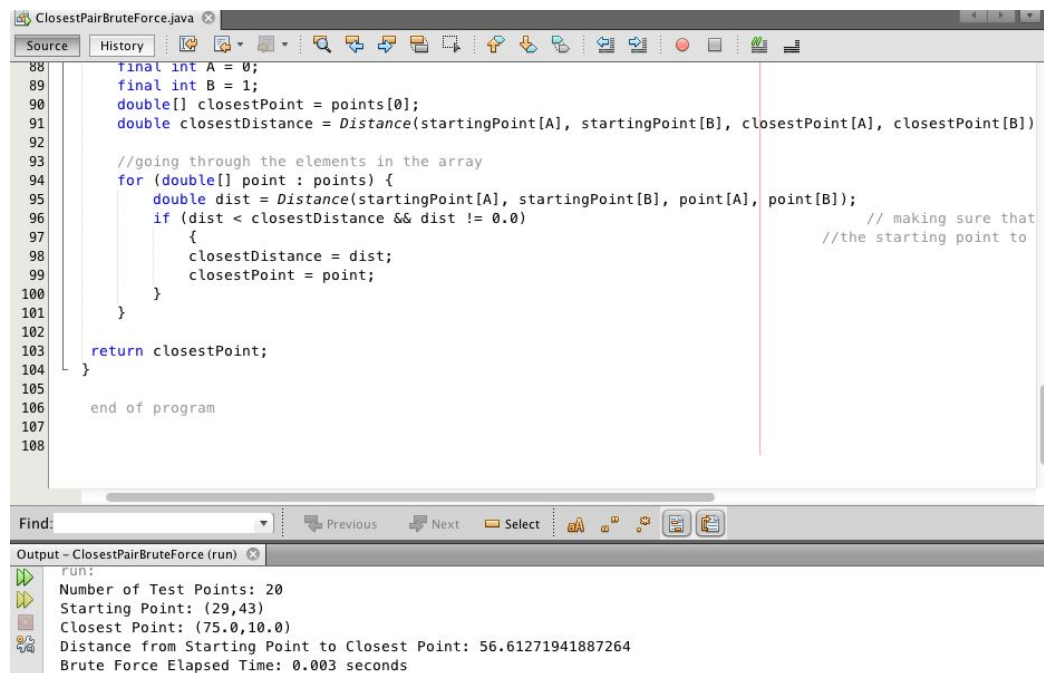
The screenshot shows an IDE window titled 'ClosestPairBruteForce.java'. The source code is as follows:

```
85
86
87 public static double[] ClosestPoint(double[] startingPoint, double[][] points){ //calculating closes
88     final int A = 0;
89     final int B = 1;
90     double[] closestPoint = points[0];
91     double closestDistance = Distance(startingPoint[A], startingPoint[B], closestPoint[A], closestPoint[B]);
92
93     //going through the elements in the array
94     for (double[] point : points) {
95         double dist = Distance(startingPoint[A], startingPoint[B], point[A], point[B]);
96         if (dist < closestDistance && dist != 0.0) // making sure t
97             { //the starting point
98                 closestDistance = dist;
99                 closestPoint = point;
100             }
101     }
102     return closestPoint;
103 }
104
105 // end of program
106
107
108
```

The output window shows the following results:

```
run:
Number of Test Points: 10
Starting Point: (4,2)
Closest Point: (34.0,25.0)
Distance from Starting Point to Closest Point: 33.015148038438355
Brute Force Elapsed Time: 0.003 seconds
```

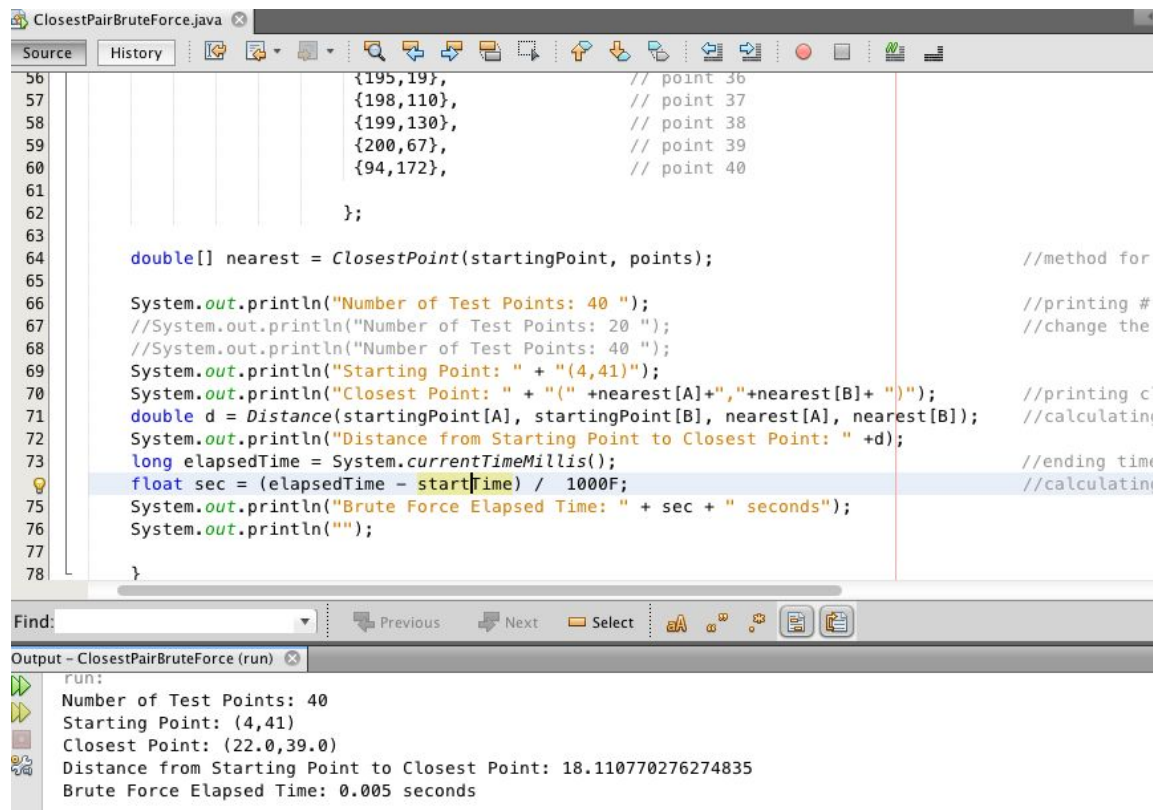
20 points



The screenshot shows the same IDE window with the source code for 20 test points. The code is identical to the previous screenshot, but the output window shows different results:

```
run:
Number of Test Points: 20
Starting Point: (29,43)
Closest Point: (75.0,10.0)
Distance from Starting Point to Closest Point: 56.61271941887264
Brute Force Elapsed Time: 0.003 seconds
```

40 Points:



```
ClosestPairBruteForce.java
Source History
56 {195,19}, // point 36
57 {198,110}, // point 37
58 {199,130}, // point 38
59 {200,67}, // point 39
60 {94,172}, // point 40
61
62 };
63
64 double[] nearest = ClosestPoint(startingPoint, points); //method for
65
66 System.out.println("Number of Test Points: 40 "); //printing #
67 //System.out.println("Number of Test Points: 20 "); //change the
68 //System.out.println("Number of Test Points: 40 ");
69 System.out.println("Starting Point: " + "(4,41)");
70 System.out.println("Closest Point: " + "(" + nearest[A] + "," + nearest[B] + ")"); //printing c
71 double d = Distance(startingPoint[A], startingPoint[B], nearest[A], nearest[B]); //calculatin
72 System.out.println("Distance from Starting Point to Closest Point: " + d);
73 long elapsedTime = System.currentTimeMillis(); //ending time
74 float sec = (elapsedTime - startTime) / 1000F; //calculatin
75 System.out.println("Brute Force Elapsed Time: " + sec + " seconds");
76 System.out.println("");
77
78 }
```

Find: Previous Next Select

Output - ClosestPairBruteForce (run)

```
run:
Number of Test Points: 40
Starting Point: (4,41)
Closest Point: (22.0,39.0)
Distance from Starting Point to Closest Point: 18.110770276274835
Brute Force Elapsed Time: 0.005 seconds
```

c.) Divide and Conquer Closest Pair

A more efficient way to solve the problem is using the divide & conquer method. In this method, generally a more difficult problem is broken down into many smaller problems. The smaller problems have the same solution as the original large problem. We recursively small the problem and combine all smaller problems at the end.

So for the Divide and Conquer method to find the closest pair we will do the following:

- 1.) split the set of points by a vertical line into two halves
 - a. Sorting points by x and y
 - i. P_x , P - sorted by x coordinate
 - ii. P_y , P - sorted by y coordinate
 - b. After x is sorted, we go to the midpoint of x and "draw" a vertical line
 - i. We will have two sets: Left and Right
- 2.) Recursively calculate closest pair in the left and right section
- 3.) Find the closest pair across the vertical dividing line - dStrip
 - i. dLeft closest distance in left section
dRight closest distance in R
 - ii. d is the minimum of (dLeft, dRight)

- iii. $\pm d$ on both sides of the vertical line will represent the strip = dzone
 - When computing the closest pair across the vertical dividing line will only consider the points within the band
- 4.) Return the shortest distance (dStrip, dLeft, or dRight)

Computing (Px, Py) from P takes $O(n \log n)$ time overall since the sorting time will take $O(n \log n)$

All the steps of the recursive calculation will take $O(n)$ time.

Constructing (Points from Left and Right Regions) from (Px, Py) -- $O(n)$

Constructing Points in strip -- $O(n)$

Overall: $T(n) = 2T(n/2) + n$

- The $2T(n/2)$ for the computation of (Px,Py) from P since we are dividing 2 sets of points of n into $n/2$.
- The recursive algorithms represents the “n”
- This time complexity represents that of merge sort and so overall $T(n) = O(n \log n)$

Overall Strategy:

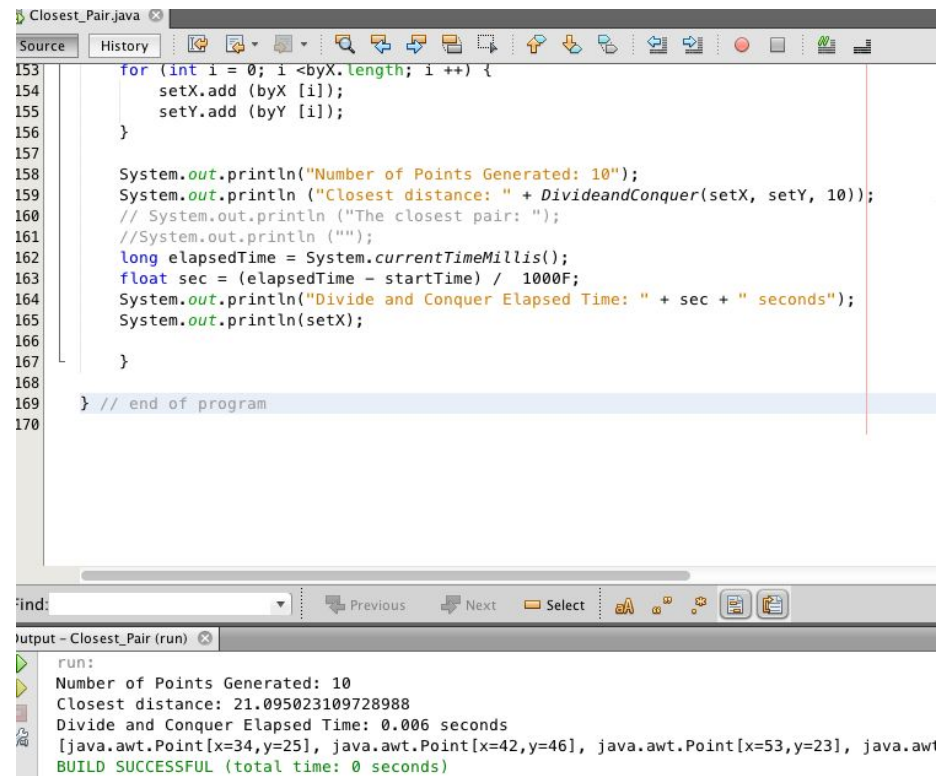
```

Closest_Pair(Px,Py)                                //assume P split into 2 sorted by x and sorted by y
  if(Total Points <= 3)                             // if you have 3 points or less than calculate using brute force
    Calculate Distance using Brute Force Method
  Calculate using Divide&Conquer
    Construct (points in left and right half)
    dLeft = Closest distance in left
    dRight = Closest distance in right
    Construct all points and find closest distance dStrip
  
```

Return dLeft, dRight, or dStrip depending on which is the smallest

Screen Shots:

10 points



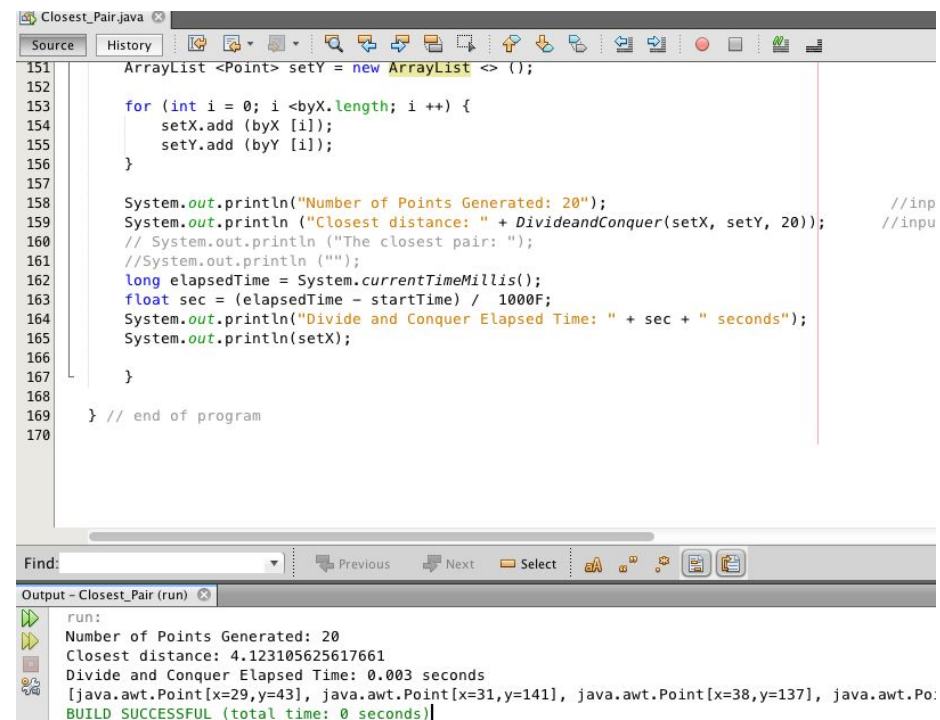
The screenshot shows an IDE window titled 'Closest_Pair.java'. The source code is as follows:

```
153     for (int i = 0; i < byX.length; i++) {
154         setX.add (byX [i]);
155         setY.add (byY [i]);
156     }
157
158     System.out.println("Number of Points Generated: 10");
159     System.out.println ("Closest distance: " + DivideandConquer(setX, setY, 10));
160     // System.out.println ("The closest pair: ");
161     //System.out.println ("");
162     long elapsedTime = System.currentTimeMillis();
163     float sec = (elapsedTime - startTime) / 1000F;
164     System.out.println("Divide and Conquer Elapsed Time: " + sec + " seconds");
165     System.out.println(setX);
166
167 }
168
169 } // end of program
170
```

The output window shows the following results:

```
run:
Number of Points Generated: 10
Closest distance: 21.095023109728988
Divide and Conquer Elapsed Time: 0.006 seconds
[java.awt.Point[x=34,y=25], java.awt.Point[x=42,y=46], java.awt.Point[x=53,y=23], java.awt.Point[x=64,y=46]]
BUILD SUCCESSFUL (total time: 0 seconds)
```

20 Points



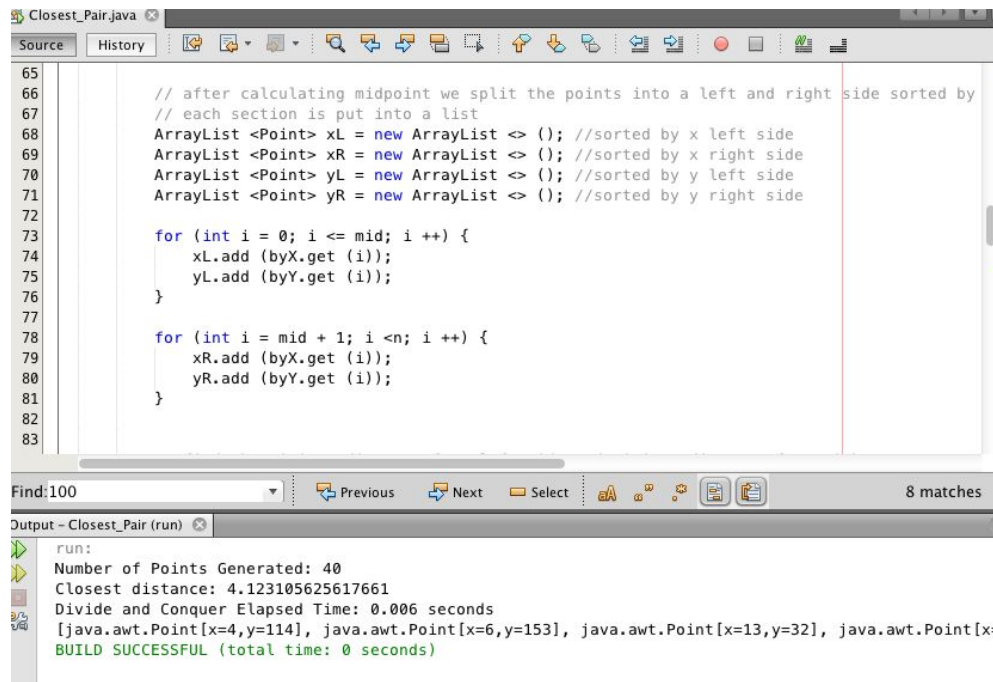
The screenshot shows the same IDE window with the source code modified to generate 20 points:

```
151     ArrayList <Point> setY = new ArrayList <> ();
152
153     for (int i = 0; i < byX.length; i++) {
154         setX.add (byX [i]);
155         setY.add (byY [i]);
156     }
157
158     System.out.println("Number of Points Generated: 20");
159     System.out.println ("Closest distance: " + DivideandConquer(setX, setY, 20));
160     // System.out.println ("The closest pair: ");
161     //System.out.println ("");
162     long elapsedTime = System.currentTimeMillis();
163     float sec = (elapsedTime - startTime) / 1000F;
164     System.out.println("Divide and Conquer Elapsed Time: " + sec + " seconds");
165     System.out.println(setX);
166
167 }
168
169 } // end of program
170
```

The output window shows the following results:

```
run:
Number of Points Generated: 20
Closest distance: 4.123105625617661
Divide and Conquer Elapsed Time: 0.003 seconds
[java.awt.Point[x=29,y=43], java.awt.Point[x=31,y=141], java.awt.Point[x=38,y=137], java.awt.Point[x=42,y=46], java.awt.Point[x=53,y=23], java.awt.Point[x=64,y=46]]
BUILD SUCCESSFUL (total time: 0 seconds)
```

40 Points



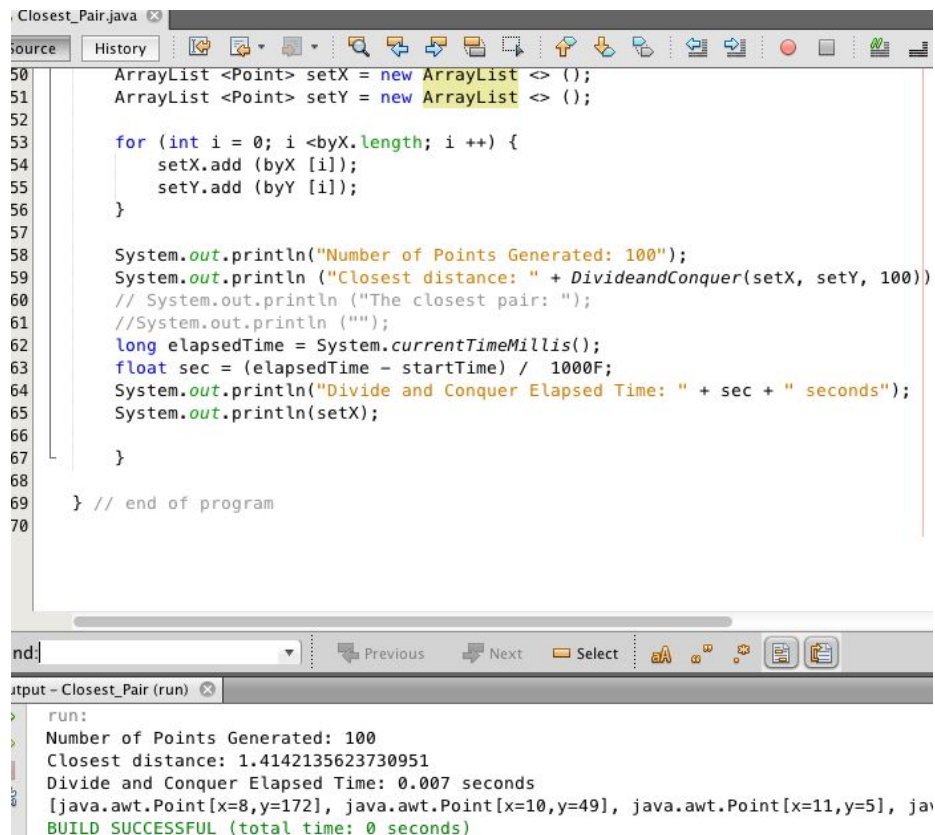
```
65
66 // after calculating midpoint we split the points into a left and right side sorted by
67 // each section is put into a list
68 ArrayList <Point> xL = new ArrayList <> (); //sorted by x left side
69 ArrayList <Point> xR = new ArrayList <> (); //sorted by x right side
70 ArrayList <Point> yL = new ArrayList <> (); //sorted by y left side
71 ArrayList <Point> yR = new ArrayList <> (); //sorted by y right side
72
73 for (int i = 0; i <= mid; i++) {
74     xL.add (byX.get (i));
75     yL.add (byY.get (i));
76 }
77
78 for (int i = mid + 1; i <n; i++) {
79     xR.add (byX.get (i));
80     yR.add (byY.get (i));
81 }
82
83
```

Find:100 Previous Next Select 8 matches

Output - Closest_Pair (run)

```
run:
Number of Points Generated: 40
Closest distance: 4.123105625617661
Divide and Conquer Elapsed Time: 0.006 seconds
[java.awt.Point[x=4,y=114], java.awt.Point[x=6,y=153], java.awt.Point[x=13,y=32], java.awt.Point[x=
BUILD SUCCESSFUL (total time: 0 seconds)
```

100 Points



```
50 ArrayList <Point> setX = new ArrayList <> ();
51 ArrayList <Point> setY = new ArrayList <> ();
52
53 for (int i = 0; i <byX.length; i++) {
54     setX.add (byX [i]);
55     setY.add (byY [i]);
56 }
57
58 System.out.println("Number of Points Generated: 100");
59 System.out.println ("Closest distance: " + DivideandConquer(setX, setY, 100))
60 // System.out.println ("The closest pair: ");
61 //System.out.println ("");
62 long elapsedTime = System.currentTimeMillis();
63 float sec = (elapsedTime - startTime) / 1000F;
64 System.out.println("Divide and Conquer Elapsed Time: " + sec + " seconds");
65 System.out.println(setX);
66
67 }
68
69 } // end of program
70
```

nd: Previous Next Select

Output - Closest_Pair (run)

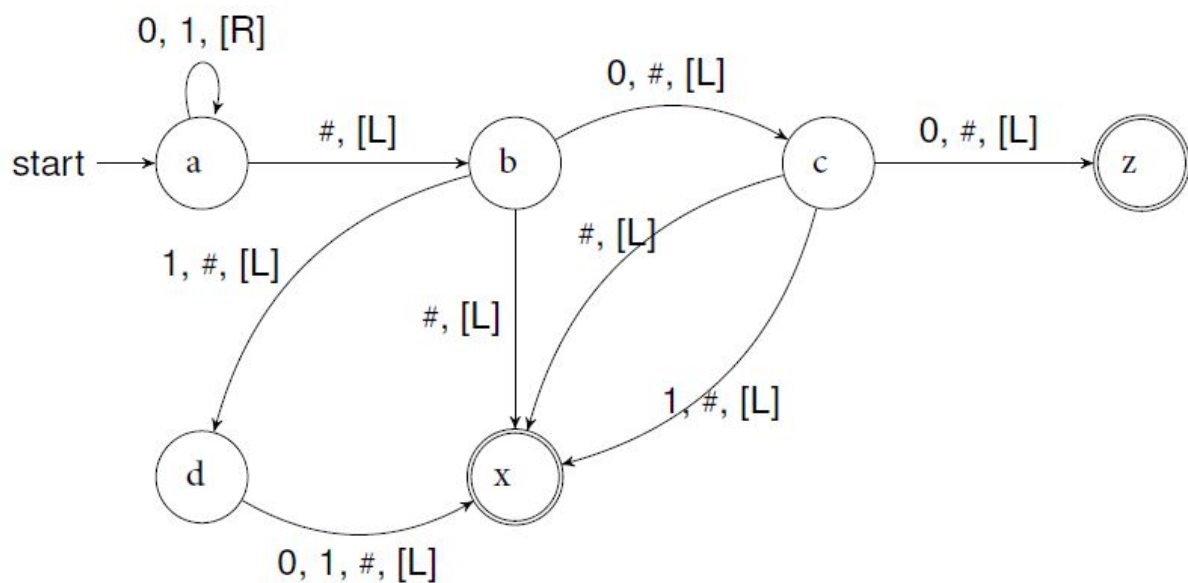
```
run:
Number of Points Generated: 100
Closest distance: 1.4142135623730951
Divide and Conquer Elapsed Time: 0.007 seconds
[java.awt.Point[x=8,y=172], java.awt.Point[x=10,y=49], java.awt.Point[x=11,y=5], ja
BUILD SUCCESSFUL (total time: 0 seconds)
```

Conclusion:

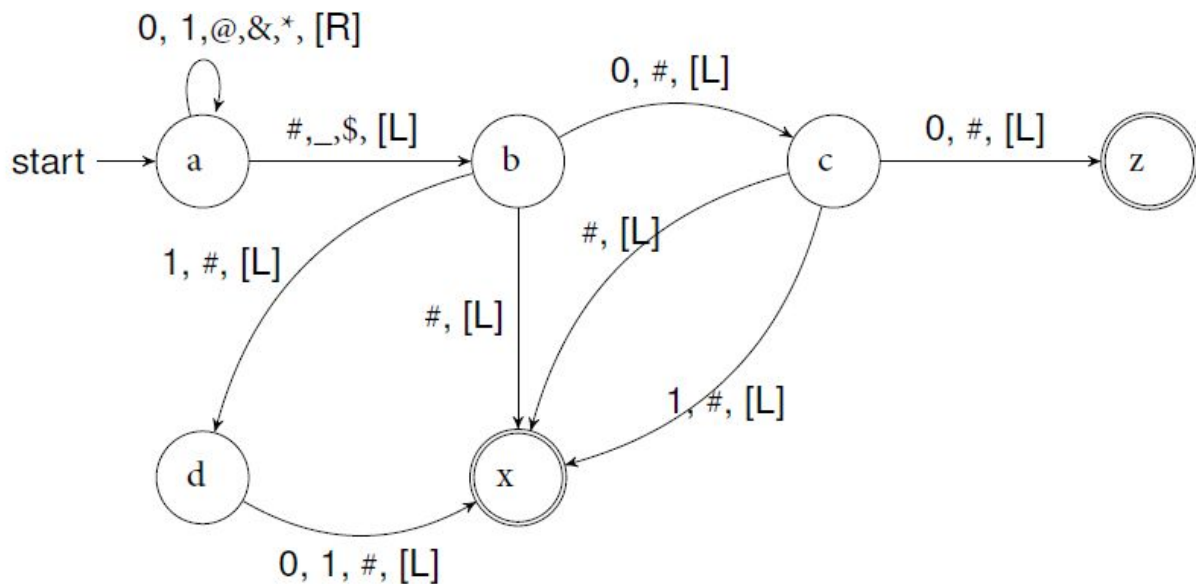
Based on the time complexity of the Brute Force algorithm being linear, I would expect that with the increase in the number of points, the runtime using the brute force strategy would also increase. Whereas, based on the time complexity of the Divide and Conquer algorithm being $n \log n$, the runtime would eventually level off or increase very slightly with the increase in points. There was a very slight difference between runtimes when running 40 points compared to 100 points. However, there was also a very small change in runtimes when increasing the number of points using the Brute Force algorithm. Overall, I believe the difference between the Brute Force algorithm and Divide and Conquer algorithm would have been more substantial if I was able to test a very large number of points but I was unable to get my divide and conquer program to work with more than 100 points.

2. Turing Machine

Original Turing Machine:



Changes to Turing Machine to implement binary operations:



- Primary changes for addition and multiplication:

State a:

- @ → @, [R]
- & → 0, [R]
- * → 0, [R]
- _ → _, [L]
- \$ → #, [L]

- Changes for subtraction: (not shown on turing machine above)

- I wasn't able to successfully perform any binary subtraction with the existing turing machine above
- For subtraction I used additional symbol c under state a in which:

- c → c, [R]

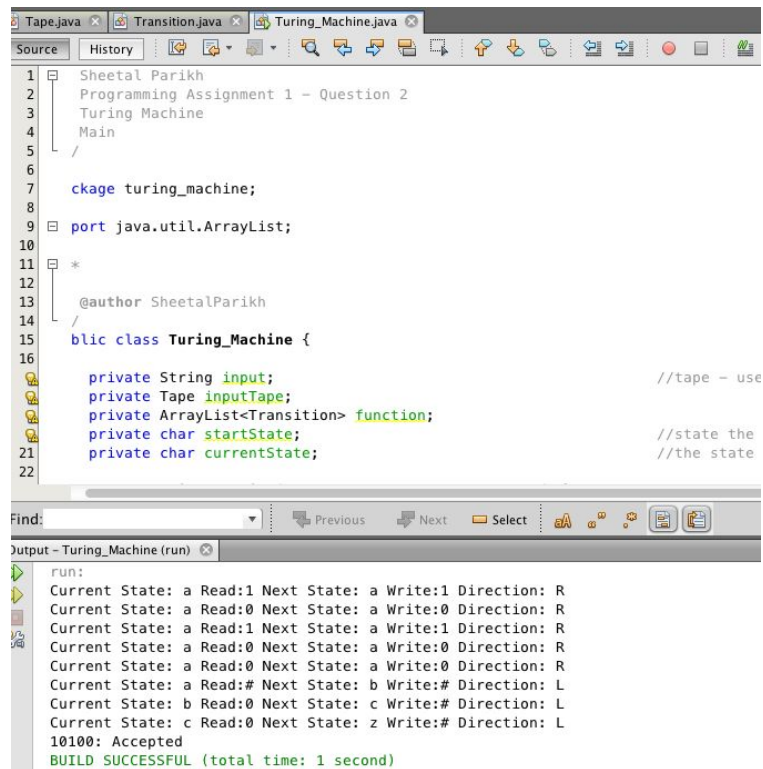
- for subtraction state a was changed to the following:

-State a:

- 1 → 0, [R]
- 0 → 1, [R]
- c → c, [R]
- @ → @, [R]
- & → 0, [R]
- * → 0, [R]
- _ → _, [L]
- \$ → #, [L]

ScreenShots

Small input



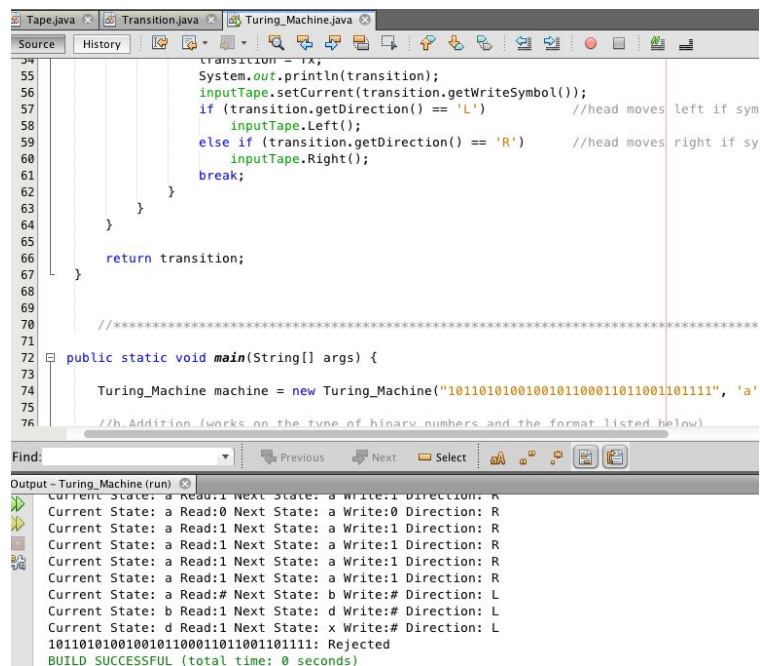
The screenshot shows an IDE with three tabs: Tape.java, Transition.java, and Turing_Machine.java. The Turing_Machine.java tab is active, displaying the following code:

```
1  Sheetal Parikh
2  Programming Assignment 1 - Question 2
3  Turing Machine
4  Main
5  /
6
7  ckage turing_machine;
8
9  port java.util.ArrayList;
10
11  *
12
13  @author SheetalParikh
14  /
15  blic class Turing_Machine {
16
17      private String input;           //tape - use
18      private Tape inputTape;
19      private ArrayList<Transition> function;
20      private char startState;       //state the
21      private char currentState;     //the state
22  }
```

The output window shows the following text:

```
run:
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:# Next State: b Write:# Direction: L
Current State: b Read:0 Next State: c Write:# Direction: L
Current State: c Read:0 Next State: z Write:# Direction: L
10100: Accepted
BUILD SUCCESSFUL (total time: 1 second)
```

Large Input



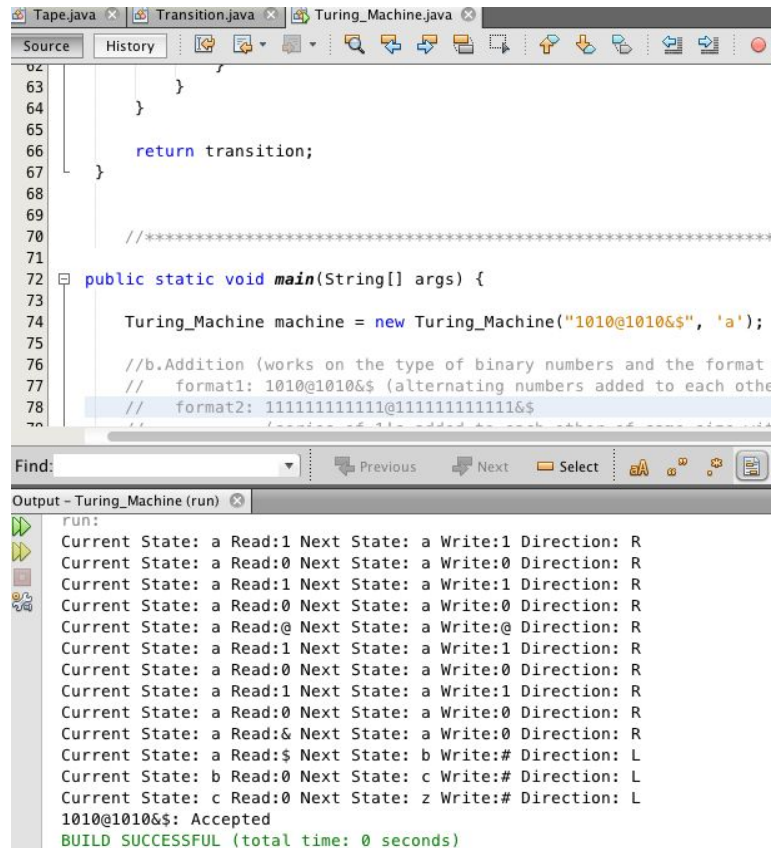
The screenshot shows the same IDE with the Turing_Machine.java tab active. The code is the same as in the previous screenshot, but the main method is now defined as follows:

```
72 public static void main(String[] args) {
73     Turing_Machine machine = new Turing_Machine("10110100100101100011011001101111", 'a')
74     //h.Addition (works on the tune of binary numbers and the format listed below)
75
76 }
```

The output window shows the following text:

```
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:# Next State: b Write:# Direction: L
Current State: b Read:1 Next State: d Write:# Direction: L
Current State: d Read:1 Next State: x Write:# Direction: L
10110100100101100011011001101111: Rejected
BUILD SUCCESSFUL (total time: 0 seconds)
```

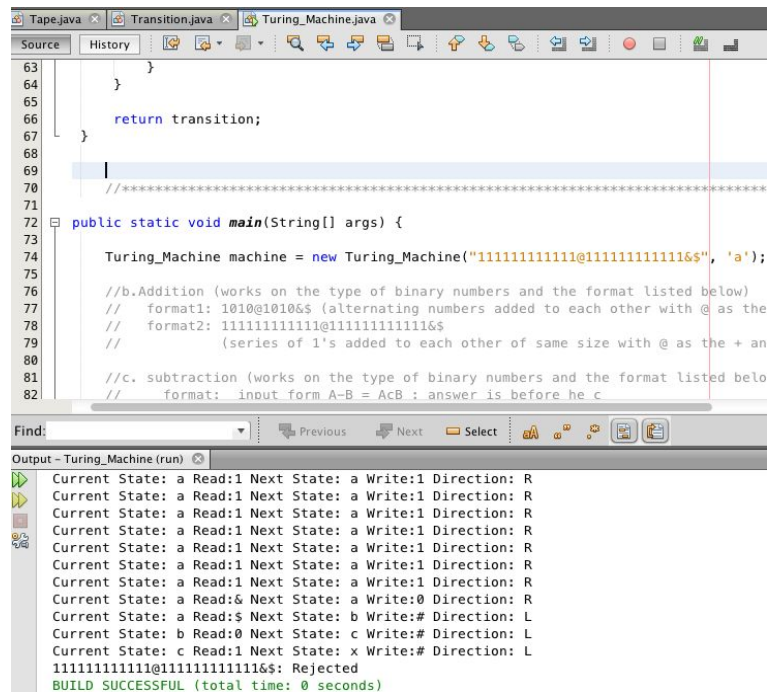
Addition



```
62 }
63 }
64 }
65 }
66 return transition;
67 }
68 }
69 }
70 //*****
71
72 public static void main(String[] args) {
73
74     Turing_Machine machine = new Turing_Machine("1010@1010&$", 'a');
75
76     //b.Addition (works on the type of binary numbers and the format
77     // format1: 1010@1010&$ (alternating numbers added to each other
78     // format2: 1111111111@1111111111&$
79     // (series of 1's added to each other of same size with @ as the + and & as the =)
80
81     //c. subtraction (works on the type of binary numbers and the format listed below)
82     // format: input form A-B = AcB : answer is before the c
83 }
```

run:

Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:@ Next State: a Write:@ Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:& Next State: a Write:0 Direction: R
Current State: a Read:\$ Next State: b Write:# Direction: L
Current State: b Read:0 Next State: c Write:# Direction: L
Current State: c Read:0 Next State: z Write:# Direction: L
1010@1010&\$: Accepted
BUILD SUCCESSFUL (total time: 0 seconds)



```
63 }
64 }
65 }
66 return transition;
67 }
68 }
69 }
70 //*****
71
72 public static void main(String[] args) {
73
74     Turing_Machine machine = new Turing_Machine("1111111111@1111111111&$", 'a');
75
76     //b.Addition (works on the type of binary numbers and the format listed below)
77     // format1: 1010@1010&$ (alternating numbers added to each other with @ as the + and & as the =)
78     // format2: 1111111111@1111111111&$
79     // (series of 1's added to each other of same size with @ as the + and & as the =)
80
81     //c. subtraction (works on the type of binary numbers and the format listed below)
82     // format: input form A-B = AcB : answer is before the c
83 }
```

run:

Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:& Next State: a Write:0 Direction: R
Current State: a Read:\$ Next State: b Write:# Direction: L
Current State: b Read:0 Next State: c Write:# Direction: L
Current State: c Read:1 Next State: x Write:# Direction: L
1111111111@1111111111&\$: Rejected
BUILD SUCCESSFUL (total time: 0 seconds)

Subtraction:

```
Source History
59
60     else if (transition.getDirection() == 'R') //head move:
61         inputTape.Right();
62         break;
63     }
64 }
65
66     return transition;
67 }
68
69
70 //*****
71
72 public static void main(String[] args) {
73
74     Turing_Machine machine = new Turing_Machine("01c011", 'a'); //input
75
76     //b.Addition (works on the type of binary numbers and the format listed l
77     // format1: 1010@1010&$ (alternating numbers added to each other with (
78     // format2: 1111111111@1111111111&$
79
80
81 Find: Previous Next Select
82
83 Output - Turing_Machine (run)
84
85 run:
86 Current State: a Read:0 Next State: a Write:1 Direction: R
87 Current State: a Read:1 Next State: a Write:0 Direction: R
88 Current State: a Read:c Next State: a Write:c Direction: R
89 Current State: a Read:0 Next State: a Write:1 Direction: R
90 Current State: a Read:1 Next State: a Write:0 Direction: R
91 Current State: a Read:1 Next State: a Write:0 Direction: R
92 Current State: a Read:# Next State: b Write:# Direction: L
93 Current State: b Read:0 Next State: c Write:# Direction: L
94 Current State: c Read:0 Next State: z Write:# Direction: L
95 01c011: Accepted
96 BUILD SUCCESSFUL (total time: 0 seconds)
```

```
Tape.java Transition.java Turing_Machine.java
Source History
55     System.out.println(transition);
56     inputTape.setCurrent(transition.getWriteSymbol());
57     if (transition.getDirection() == 'L') //
58         inputTape.Left();
59     else if (transition.getDirection() == 'R') //
60         inputTape.Right();
61     break;
62 }
63 }
64
65
66     return transition;
67 }
68
69
70 //*****
71
72 static void main(String[] args) {
73
74     Turing Machine machine = new Turing Machine("011111c000100", 'a');
75
76
77 Find: Previous Next Select
78
79 Output - Turing_Machine (run)
80
81 Current State: a Read:1 Next State: a Write:0 Direction: R
82 Current State: a Read:1 Next State: a Write:0 Direction: R
83 Current State: a Read:c Next State: a Write:c Direction: R
84 Current State: a Read:0 Next State: a Write:1 Direction: R
85 Current State: a Read:0 Next State: a Write:1 Direction: R
86 Current State: a Read:1 Next State: a Write:0 Direction: R
87 Current State: a Read:0 Next State: a Write:1 Direction: R
88 Current State: a Read:0 Next State: a Write:1 Direction: R
89 Current State: a Read:# Next State: b Write:# Direction: L
90 Current State: b Read:1 Next State: d Write:# Direction: L
91 Current State: d Read:1 Next State: x Write:# Direction: L
92 011111c000100: Rejected
```

Multiplication

```
58         inputTape.Left();
59         else if (transition.getDirection() == 'R') //head moves right
60             inputTape.Right();
61         break;
62     }
63 }
64 }
65
66     return transition;
67 }
68
69
70
71
72 public static void main(String[] args) {
73
74     Turing_Machine machine = new Turing_Machine("10000***_10000$", 'a'); //i
75
76     //b.Addition (works on the type of binary numbers and the format listed below)
77     // format1: 1010@1010$ (alternating numbers added to each other with @ as a
```

Find: Previous Next Select

Output - Turing_Machine (run)

```
run:
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:0 Next State: a Write:0 Direction: R
Current State: a Read:* Next State: a Write:0 Direction: R
Current State: a Read:* Next State: a Write:0 Direction: R
Current State: a Read:* Next State: a Write:0 Direction: R
Current State: a Read:_ Next State: b Write:_ Direction: R
Current State: b Read:1 Next State: d Write:# Direction: L
10000***_10000$: Rejected
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
56         inputTape.setCurrent(transition.getWriteSymbol());
57         if (transition.getDirection() == 'L') //head moves left
58             inputTape.Left();
59         else if (transition.getDirection() == 'R') //head moves right
60             inputTape.Right();
61         break;
62     }
63 }
64 }
65
66     return transition;
67 }
68
69
70
71
72 public static void main(String[] args) {
73
74     Turing_Machine machine = new Turing_Machine("11*_10", 'a');
75 }
```

Find: Previous Next Select

Output - Turing_Machine (run)

```
run:
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:1 Direction: R
Current State: a Read:* Next State: a Write:0 Direction: R
Current State: a Read:_ Next State: b Write:_ Direction: R
Current State: b Read:1 Next State: d Write:# Direction: L
11*_10: Rejected
BUILD SUCCESSFUL (total time: 0 seconds)
```