Sheetal Parikh
Programming Assignment 1 - Summary

1. Closest Pairs

a.) *Brute Force Closest Pair*

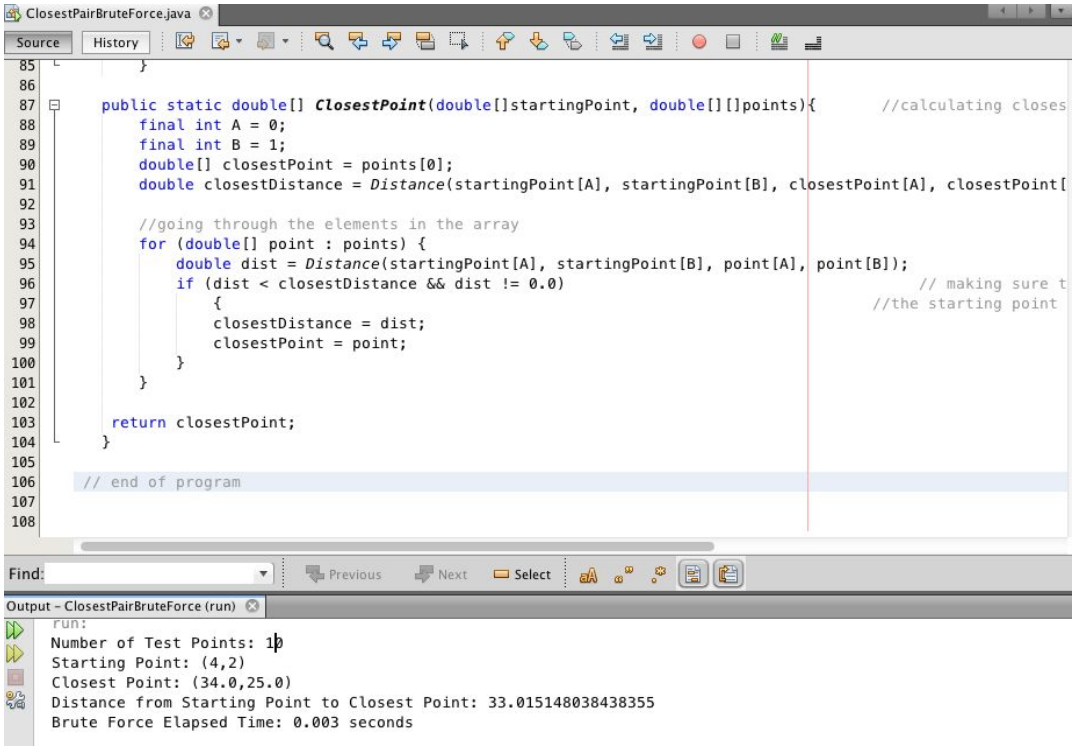
For the brute force method, we would compute all the distances between every pair of points and select the minimum distance. If we are at point n, then we have to find the distances between n and n-1 points. This applies to all n points, so the total distance comparisons between all points can be described as n(n-1). In this calculation, we also double counted distances. For example, the distance between point and 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:

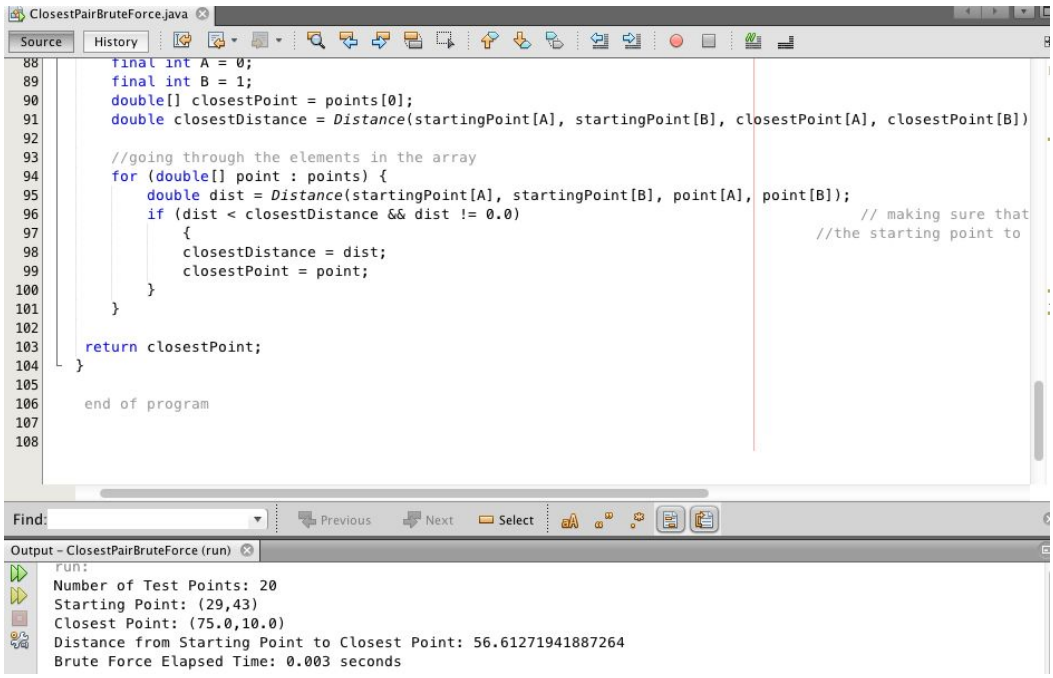
```
Closest_Pair(Points P, N)
{
    for i = 0; i < N-1; i++) {
        for (j=1+i; j < N; j++ {
            Get minimum distance(P[i], P[j]);
        }
    }
}
return Closest_Pair
```

Screenshots:

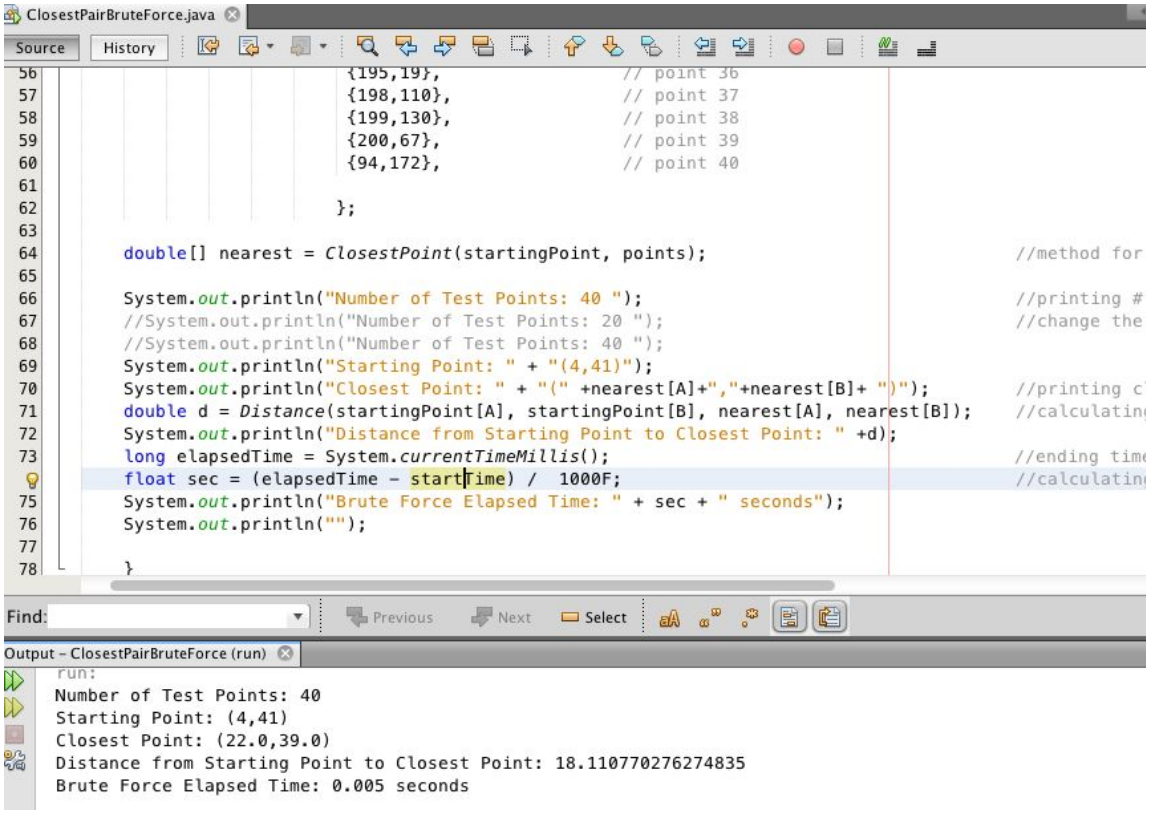
10 Points



20 points



40 Points:



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. Px, P - sorted by x coordinate
      ii. Py, 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. +/- 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(nlogn) time overall since the sorting time will take O(nlogn)
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(nlogn)

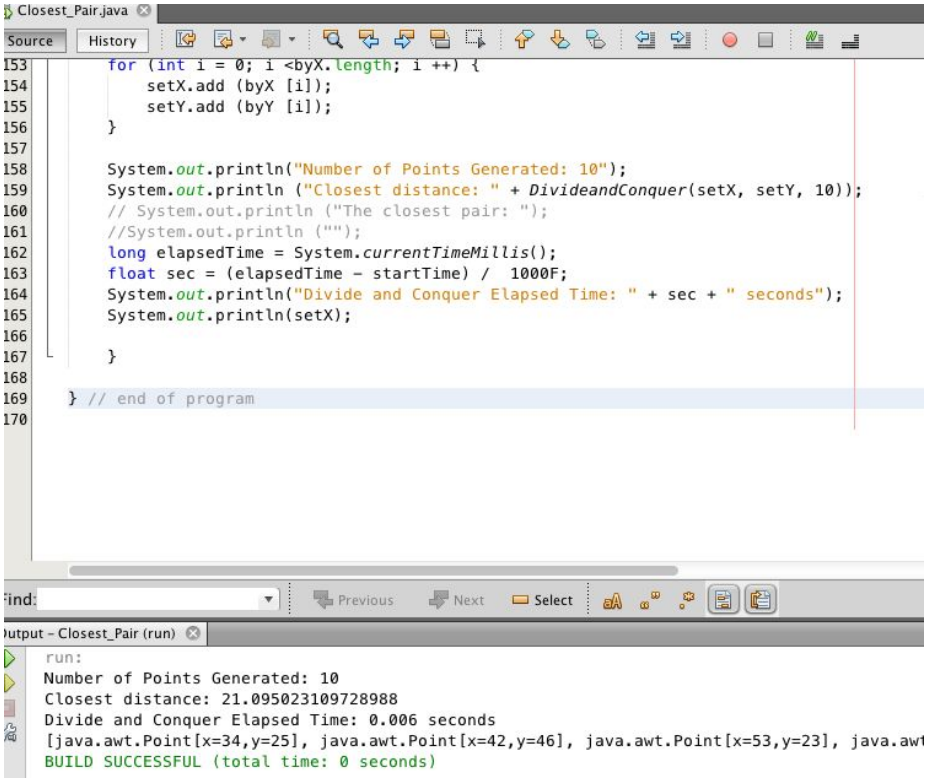
Overall Strategy:

```
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
```

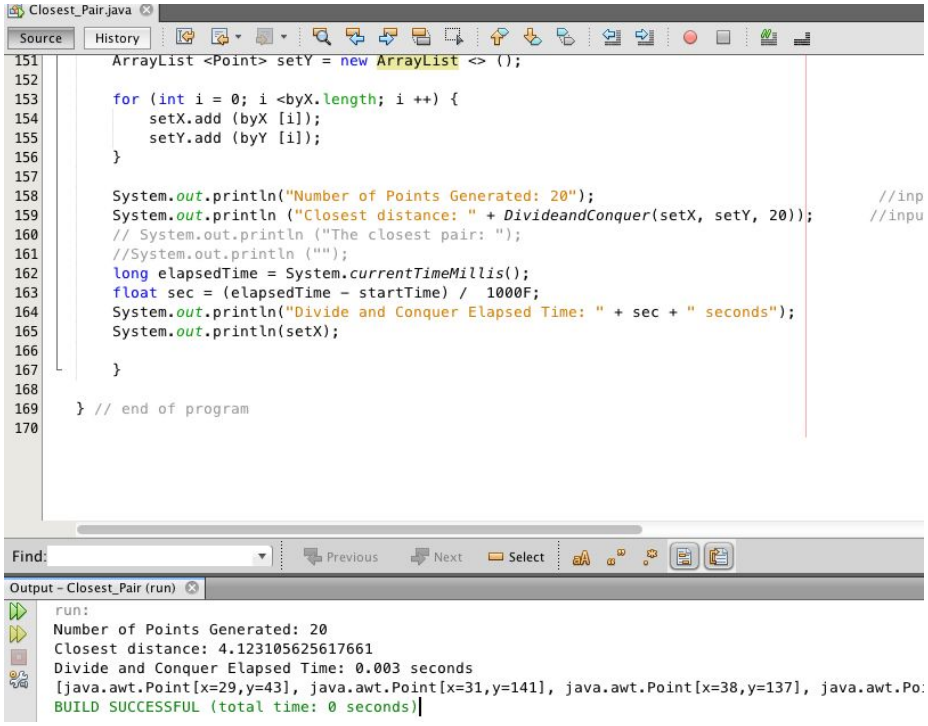
Return dLeft, dRight, or dStrip depending on which is the smallest

Screen Shots:

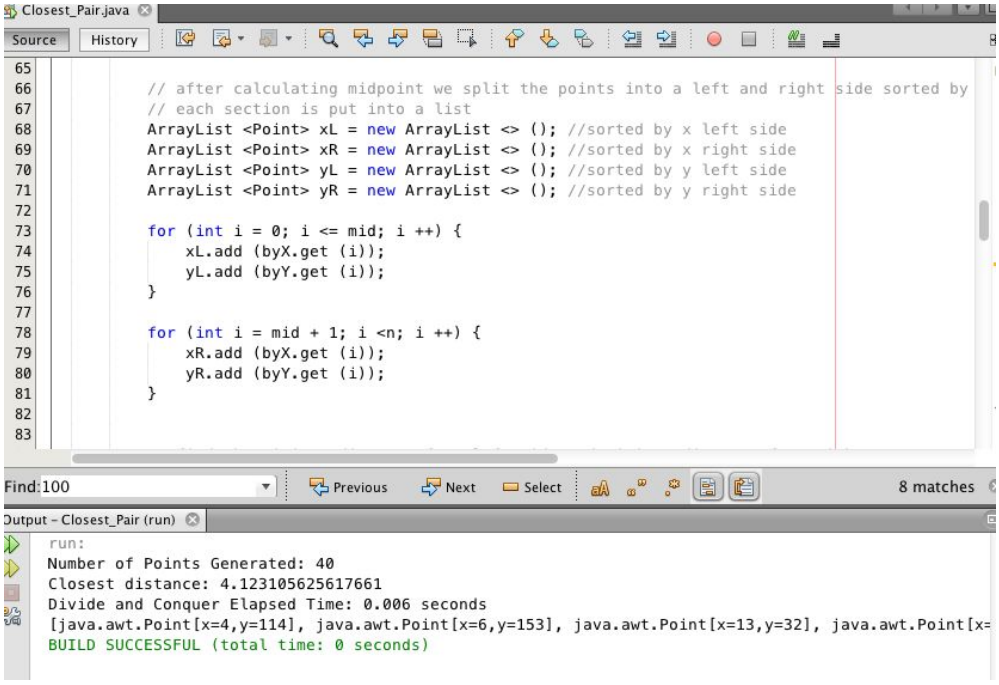
10 points



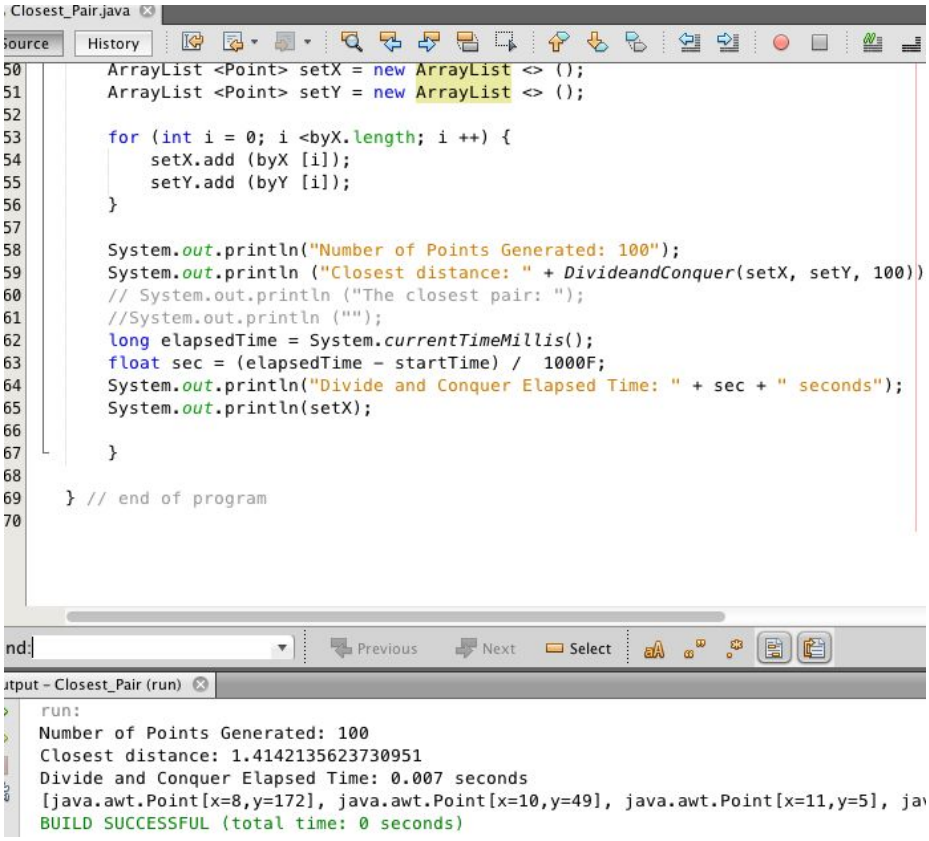
20 Points



40 Points



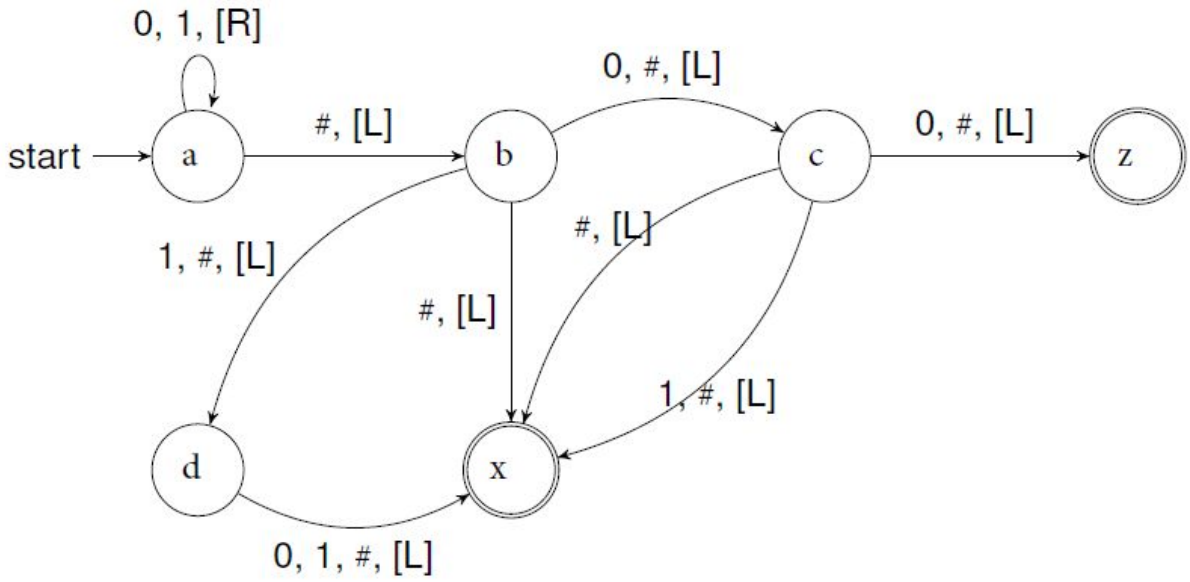
100 Points



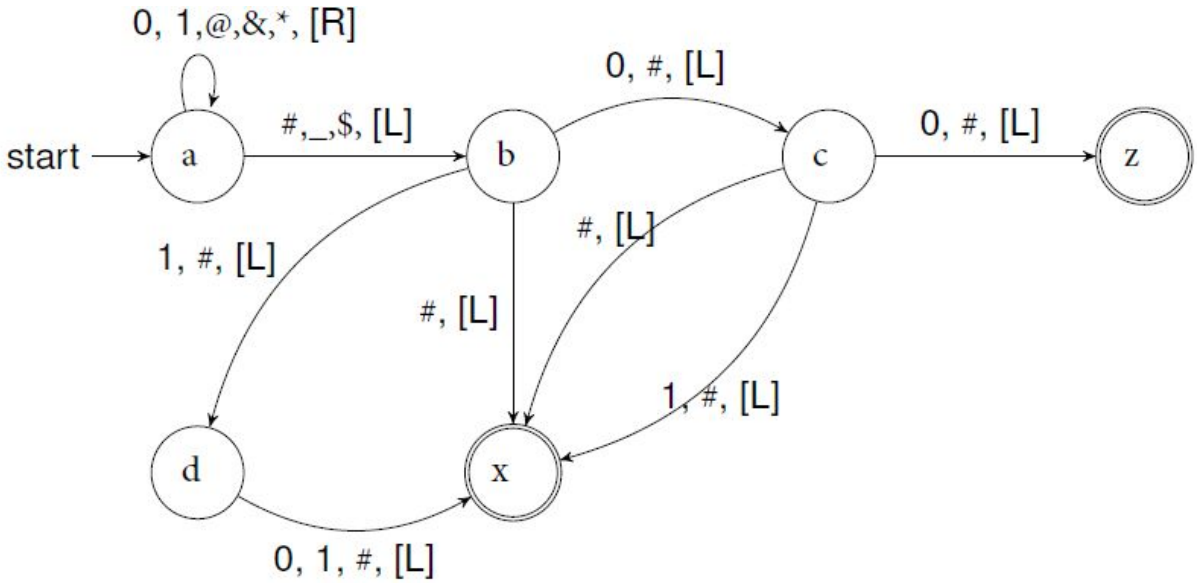
Conclusion:
Based on the time complexity of the Brute Force algorithm being linear, I would expect that with the increase in the number of points, the runtime using the brute force strategy would also increase. Whereas, based on the time complexity of the Divide and Conquer algorithm being nlogn, the 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 toTuring Machine to implement binary operations:

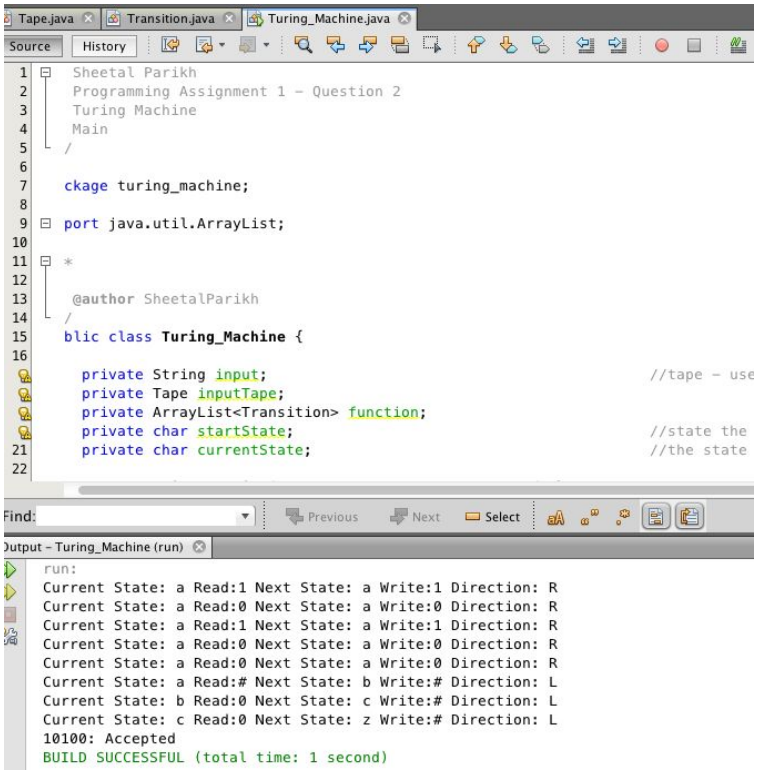


- Primary changes for addition and multiplication:
  State a:
  - @ → @, [R]
  - & → 0, [R]
  - * → 0 , [R]
  - _ → _, [L]
  - $ → #, [L]

- 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]

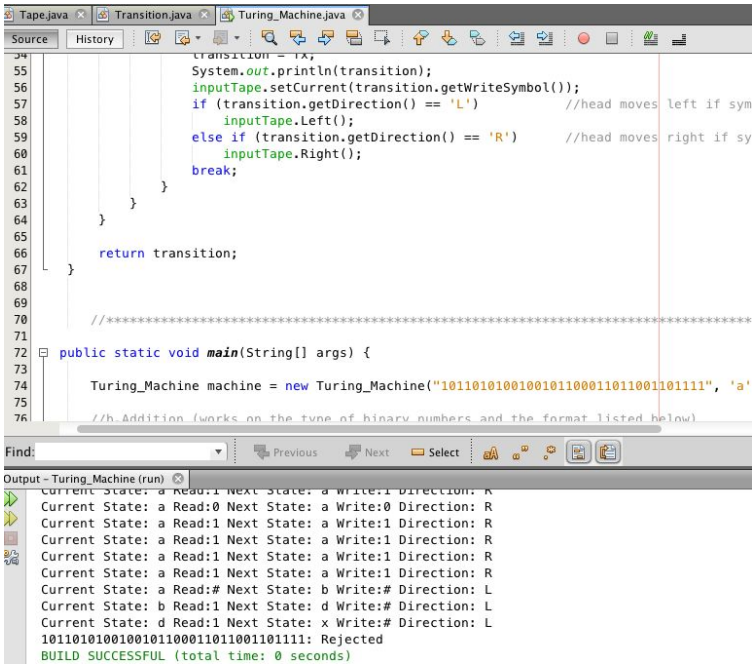
ScreenShots

Small input



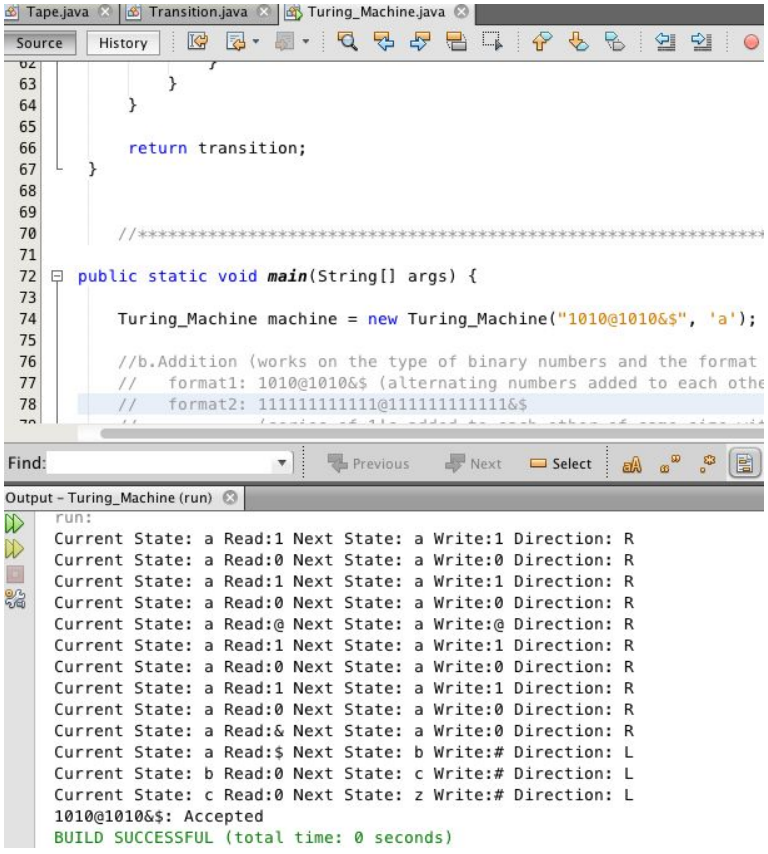
```
Tape.java | Transition.java | Turing_Machine.java
Source | History

 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
      private String input;                                    //tape - use
      private Tape inputTape;
      private ArrayList<Transition> function;
      private char startState;                                  //state the
21    private char currentState;                               //the state
22

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: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)
```

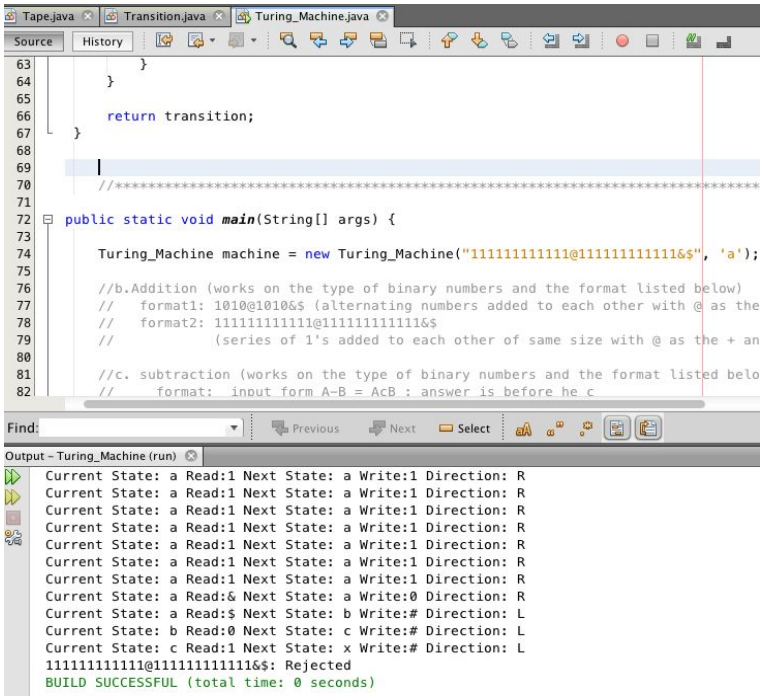
Large Input



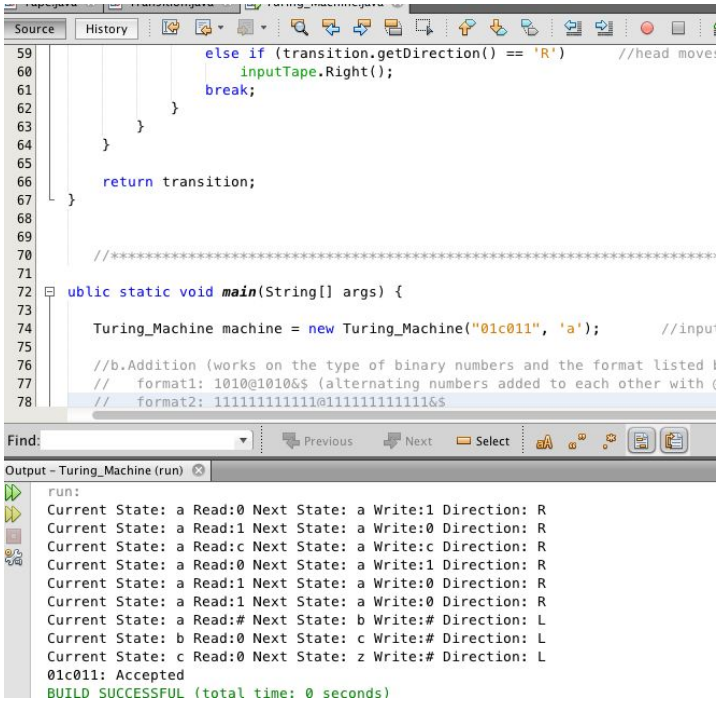
```
Tape.java | Transition.java | Turing_Machine.java
Source | History

                    System.out.println(transition);
                    inputTape.setCurrent(transition.getWriteSymbol());
                    if (transition.getDirection() == 'L')          //head moves left if sym
                        inputTape.Left();
                    else if (transition.getDirection() == 'R')     //head moves right if sy
                        inputTape.Right();
                    break;
                }
            }
        }

        return transition;
    }


    //*************************************************************************************

public static void main(String[] args) {

    Turing_Machine machine = new Turing_Machine("10110101001001011000110110011011111", 'a'

    //b Addition (works on the type of binary numbers and the format listed below)

Find:  Previous  Next  Select

Output - Turing_Machine (run)
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
10110101001001011000110110011011111: Rejected
BUILD SUCCESSFUL (total time: 0 seconds)
```

Addition


Tape.java
Transition.java
Turing_Machine.java
Source
History
}
}
return transition;
}
//**********
public static void main(String[] args) {
Turing_Machine machine = new Turing_Machine("1010@1010&$", 'a');
//b.Addition (works on the type of binary numbers and the format
// format1: 1010@1010&$ (alternating numbers added to each othe
// format2: 111111111111@111111111111&$
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: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)


Tape.java
Transition.java
Turing_Machine.java
Source
History
}
}
return transition;
}
//**********
public static void main(String[] args) {
Turing_Machine machine = new Turing_Machine("111111111111@111111111111&$", 'a');
//b.Addition (works on the type of binary numbers and the format listed below)
// format1: 1010@1010&$ (alternating numbers added to each other with @ as the
// format2: 111111111111@111111111111&$
// (series of 1's added to each other of same size with @ as the + an
//c. subtraction (works on the type of binary numbers and the format listed belo
// format: input form A-B = AcB : answer is before he c
Find:
Previous
Next
Select
Output - Turing_Machine (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
111111111111@111111111111&$: Rejected
BUILD SUCCESSFUL (total time: 0 seconds)

Subtraction:

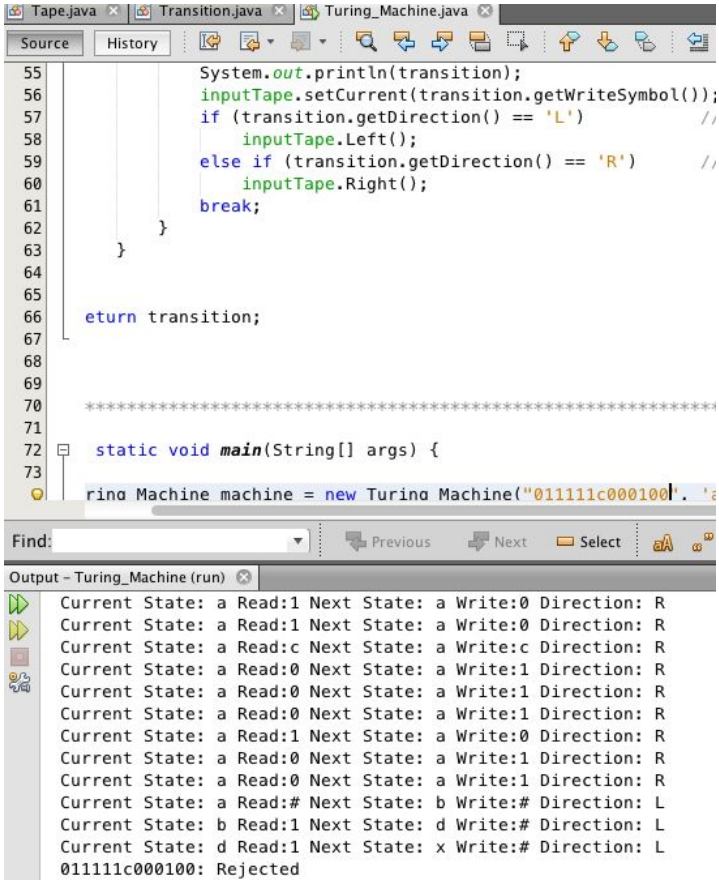


```
                else if (transition.getDirection() == 'R')      //head moves
                    inputTape.Right();
                break;
            }
        }
    }

    return transition;
}


    //*****************************************************************

ublic static void main(String[] args) {

    Turing_Machine machine = new Turing_Machine("01c011", 'a');       //input

    //b.Addition (works on the type of binary numbers and the format listed
    //    format1: 1010@1010&$ (alternating numbers added to each other with
    //    format2: 111111111111@111111111111&$
```

Find: Previous Next Select

Output - Turing_Machine (run)

```
run:
Current State: a Read:0 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:0 Direction: R
Current State: a Read:c Next State: a Write:c Direction: R
Current State: a Read:0 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:0 Direction: R
Current State: a Read:1 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
01c011: Accepted
BUILD SUCCESSFUL (total time: 0 seconds)
```



Tape.java | Transition.java | Turing_Machine.java

Source | History

```
                System.out.println(transition);
                inputTape.setCurrent(transition.getWriteSymbol());
                if (transition.getDirection() == 'L')           //
                    inputTape.Left();
                else if (transition.getDirection() == 'R')      //
                    inputTape.Right();
                break;
            }
        }


    return transition;


    //*****************************************************************

     static void main(String[] args) {

    ring Machine machine = new Turing Machine("011111c000100", 'a
```

Find: Previous Next Select

Output - Turing_Machine (run)

```
Current State: a Read:1 Next State: a Write:0 Direction: R
Current State: a Read:1 Next State: a Write:0 Direction: R
Current State: a Read:c Next State: a Write:c Direction: R
Current State: a Read:0 Next State: a Write:1 Direction: R
Current State: a Read:0 Next State: a Write:1 Direction: R
Current State: a Read:0 Next State: a Write:1 Direction: R
Current State: a Read:1 Next State: a Write:0 Direction: R
Current State: a Read:0 Next State: a Write:1 Direction: R
Current State: a Read:0 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
011111c000100: Rejected
```

Multiplication