assi2

1. 

```
class DuplicateN {
    static IntUtil u = new IntUtil();

    /* NOTHING CAN BE CHANGED IN CONSTRUCTOR */
    DuplicateN() { testBed(); }

    /*
     * Time: O(n^2)
     * Space: THETA(1)
     */

    private int bigOnSquareTimeTheta1Space(int[] a, boolean show) {
        int k = 0;
        //WRITE CODE HERE
        for (int i = 0; i < a.length; i++) {
            int ct = 0;
            for (int j = 0; j < a.length; j++) {
                if (a[j] == i) {
                    ct++;
                }
            }
            if (ct > 1) {
                k++;
                if (show) {
                    System.out.print((i) + " ");
                }
            }
        }
        if (show) {
            System.out.println() ;
        }
        return k;
    }
```

2.

```java
private int thetaNTimeThetaNSpace(int[] a, boolean show) {
    int k = 0;
    //WRITE CODE HERE
    int n = a.length;
    int myhash[] = new int[n];
    for (int i = 0; i < n; i++)
        myhash[i] = 0;
    for (int i = 0; i < n; i++) {
        myhash[a[i]]++;
    }
    for (int i = 0; i < n; i++)
        if (myhash[i] > 1) {
            k++;
            if (show) {
                System.out.print((i) + " ") ;
            }
        }
    if (show) {
        System.out.println() ;
    }
    return k;
}

int myabs(int x) { return x > 0 ? x : -x; }
```

ass3

```
class AmicablePair {
   private int max;
   //YOU CAN HAVE ANY NUMBER OF PRIVATE VARIABLES HERE
//    static IntUtil u = new IntUtil();

   /*
    * Constructor
    */
   AmicablePair(int n) {
      max = n;
      long start = System.currentTimeMillis();
      findAmicablePair(max);
      long end = System.currentTimeMillis();
      System.out.println("run time for " + n + " is " + (end - start) / 1000 + "secs");
      /* You can initialize any variables here */
      /* you can call any private functions here */
   }
```

```
    /*
     *      WRITE YOUR CODE BELOW
     *      MUST SOLVE FROM FUNDAMENTALS
     *      CANNOT USE SOME WEIRD FORMULAS FROM INTERNET
     */
    private void findAmicablePair(int n) {
        int j, last, count = 0;
        int[] sumOfFactorArr = new int[n + 1];
        for (int i = 1; i <= (n / 2); i++) {
            j = i * 2;
            while (j <= n) {
                sumOfFactorArr[j] += i;
                j += i;
            }
        }
        for (int first = 2; first <= n; first++) {
            last = sumOfFactorArr[first];
            if (last <= n && last > first) {
                if (sumOfFactorArr[last] == first) {
                    System.out.println(count + ": " + first + " and " + last + " ");
                    count++;
                }
            }
        }
    }


    /*
     * NOTHING CAN BE CHANGED BELOW
     */
    private static void test() {
        int n = 100000000;
        AmicablePair a = new AmicablePair(n);
    }

    public static void main(String[] args) {
        System.out.println("AmicablePair.java starts");
        test();
        System.out.println("If you can do in less than 20 secs, you will get FULL points");
        System.out.println("Attach AmicablePair.java, output of your program as a pdf file");
        System.out.println("Attach a word file, explaining why your method is fast");
        System.out.println("AmicablePair.java ends");
    }
}
```

assi4

1. prime

```
public class Prime {
static private int max;
static private int[] p; //array that has prime number
static private int pkount;
static private long steps;
static private long gsteps; //Number of steps guessed through formula
static private boolean display = false;
static IntUtil u = new IntUtil();

Prime(int n) {
    max = n;
    p = new int[n + 1];
    pkount = 0;
    steps = 0;
    gsteps = 0;
}

private boolean isPrimebBruteForce(int n) {
    gsteps = (long)(n / (Math.log(n) - 1));
    for (int i = 2; i < n; i++) {
        steps++;
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}
public void bruteForce() {
    //WRITE CODE
    for (int i = 2; i <= max; i++) {
                if (isPrimebBruteForce(i)) {
                    p[pkount++] = i;
        }
    }
    //This must be last line
    pLn("----------bruteForce method---------- ");
}
private boolean isPrimeUptoSquareRoot(int n) {
    gsteps = (long)(n / (Math.log(n) - 1));
    for (int i = 2; i * i < n; i++) {
        steps++;
        if (n % i == 0) {
            return false;
```

```
        }
    }
    return true;
}
public void uptoSquareRoot() {
    //WRITE CODE
    for (int i = 2; i <= max; i++) {
        if (isPrimeUptoSquareRoot(i)) {
            p[pkount++] = i;
        }
    }
    //This must be last line
    pLn("----------uptoSquareRoot method---------- ");
}

public void uptoPrimeNumbers() {
    //WRITE CODE
    gsteps = (long)(max / (Math.log(max) - 1));
    int pkount =0;
    p[pkount++] = 2;
    for (int i = 3; i <= max; i++) {
        boolean divisible = false;
        for (int k = 0; p[k] * p[k] <= i; k++) {
            steps++;
            if (p[k] % i == 0) {
                divisible = true;
            }
        }
        if (divisible == false) {
            p[pkount++] = i;
        }

    }
    //This must be last line
    pLn("----------uptoPrimeNumbers method---------- ");
}

public void SieveOfEratosthenes() {
    //WRITE CODE
    gsteps = (long)(max / (Math.log(max) - 1));
    boolean [] a = new boolean[max + 1];
    for (int i = 0; i <= max; i++) {
        a[i] = true;
    }
    a[0] = false;
    a[1] = false;
```

```
        for (int i = 2; i * i <= max; i++) {
            steps++;
            if (a[i]) {
                for (int j = 2; i * j <= max; j++) {
                    a[j * i] = false;
                    pkount++;
                }
            }
        }

        //This must be last line
        pLn("----------SieveOfEratosthenes method---------- ");
    }
```

2. truthtable

```
class TruthTable {
    private int n;
    private boolean display;
    static IntUtil u = new IntUtil();

    /*
     * ---------WRITE CODE HERE ------------------
     */

    TruthTable(int i, boolean display) {
        n = i;
        this.display = display;
       System.out.println("For " + n + " inputs, table size is " + this.print2("", n));
    }


    public int print2(String j, int n) {
        if (n == 0) {
            if (display) {
                System.out.println(j);
            }
            return 1;
        } else {
            return print2(j + "0", n - 1) + print2(j + "1", n - 1);
        }
    }
```

assi5

1.length

```
private static int length(int[] s, int x) {
                    /*
                                YOU CANNOT USE ANY static variable in this function
                            YOU can use as many local variables inside the function
                            Cannot use any loop statements like  for, while, do, while, go to
                            Can use if.
                            ONLY AFTER THE execution of this routine array s MUST have the
same contents as you got it.
                     YOU cannot call any subroutine inside this function except length itself (NOT
length_easy)
                    */
//        int i = s[x];
//        if (i >= 0) {
//            s[x] = - i - 1;
//            int l = length(s, i);
//            s[x] = i;
//            return 1 + l;
//        } else {
//            return -1;
//        }
    /*

     */
        if (s[x] == x) {
            return 0;
        }
        int p = s[x];
        s[x] = s[p];
        int l = 1 + length(s, x);
        s[x] = p;
        return l;
    }
```

2.

graphbuilder

```
private void buildGraph() {
        //WRITE GRAPH
        //make sure data is right
        //get a unique number from 0 to NUMV-1 for each ride
        GraphTest.GraphType t = g.getType();
        for (String[] a : e) {
```

```
        if ((t == GraphTest.GraphType.UNDIRECTED) || (t ==
GraphTest.GraphType.DIRECTED)) {
            g.u.myassert(a.length == 2);
        } else if ((t == GraphTest.GraphType.WEIGHTED_UNDIRECTED) || (t ==
GraphTest.GraphType.WEIGHTED_DIRECTED)){
            g.u.myassert(a.length == 3);
        } else {
            g.u.myassert(false);
        }
        for (int j = 0; j < 2; j++) {
            g.insertOrFind(a[j], false);
        }
      }
      //Build the graph now
      double w = 0;
      for (String [] a : e) {
        int p1 = g.insertOrFind(a[0], true);
        int p2 = g.insertOrFind(a[1], true);
        if ((a.length == 3)) {
            w = Double.parseDouble(a[2]);
        }
        //p1 has a fanout p2
        g.createEdge(p1, p2, w, true);//fanout
        //p2 has a fanout of p1
        g.createEdge(p2, p1, w, false);
        if ((t == GraphTest.GraphType.UNDIRECTED) || (t ==
GraphTest.GraphType.WEIGHTED_UNDIRECTED)) {
            //p2 has a fanout p1
            g.createEdge(p2, p1, w, true);//fanout
            //p1 has a fanin of p2
            g.createEdge(p1, p2, w, false);//fanin
        }
      }
   }

graphdot

class GraphDot{
        private Graph g ;
        private String fname;
        //You can have any number of private variables
        private String[][] e ;

        GraphDot(Graph g, String s) {
                this.g = g ;
                this.fname = s ;
```

```
                writeDot() ;
        }

        private void writeDot(){
                //WRITE CODE
      try{
        System.out.println(fname);
        FileWriter o = new FileWriter(fname);
        GraphTest.GraphType t = g.getType();
        o.write("## Qing ####\n");
        o.write("## dot -Tpdf " + fname + " -o " + fname + ".pdf\n");
        o.write("digraph g {\n");
                        if (t == GraphTest.GraphType.UNDIRECTED || t ==
GraphTest.GraphType.WEIGHTED_UNDIRECTED) {
                                o.write("edge [dir=none, color=red]\n");
                        } else {
                           o.write("edge [color=red]\n");
        }

        int n = g.getnumV();
                        g.u.myassert(n == g.getnumV());
                        //Time complexity: O(V + E)
        for (int i = 0; i < g.getnumV(); i++) {
           String p1 = g.getNodeRealName(i);
           int nf = g.numFanout(i);
           for (int j = 0; j < nf; j++) {
              int k = g.getNodeFanout(i,j);
              String p2 = g.getNodeRealName(k);
              double w = g.getNodeFanoutEdgeWeight(i,j);
              if ((t == GraphTest.GraphType.WEIGHTED_UNDIRECTED) || (t ==
GraphTest.GraphType.WEIGHTED_DIRECTED)) {
                 if ((t == GraphTest.GraphType.WEIGHTED_DIRECTED) || (i < k)) {
                    System.out.println("   " + p1 + " -> " + p2 + " [label = " + w + "] \n");
                    o.write("   " + p1 + " -> " + p2 + " [label = " + w + "] \n");
                 }
              } else {
                 if ((t == GraphTest.GraphType.DIRECTED) || (i < k)) {
                    System.out.println("   " + p1 + " -> " + p2 + "\n");

                    o.write("   " + p1 + " -> " + p2 + "\n");
                 }
              }
           }
        }
        o.close();
      } catch (IOException e1) {
```

```
            e1.printStackTrace();//TODO
        }
    }
```