CS731 : SOFTWARE TESTING PROJECT REPORT

Astha Borkataky (MT2021027)
Satya Jyoti Das (MT2021120)
Gaurav Kumar (MT2021046)

This report contains details for the project done in CS731 course - Software Testing. Our chosen domain for testing was to formulate **Control flow graphs (CFG)** over the source code.

Coverage Criteria Demonstrated :

- ➔ Edge coverage
- ➔ Prime Path coverage

Tools Used

- ➔ JUnit 4.13

The project consists of functions covering different topics, such as Arithmetic, Maths, Searching and Sorting. It is a Java based console application that is split up into 4 packages each containing amalgamation of codes, such that we can aptly demonstrate various scenarios like- loops, loops+conditionals, conditional within loops, loops within conditionals, switch cases with conditionals, nested conditionals etc, thereby attempting to make the source code rich in control flow structure.

- **Arithmetic**
  - ➔ Operations
  - ➔ AdvancedOperations
- **Maths**
  - ➔ Armstrong Number
  - ➔ HCF
  - ➔ LCM
  - ➔ LeapYear
  - ➔ NthFibo
  - ➔ PalindromeCheck
  - ➔ MaxThreeNumbers
  - ➔ TriplePythoVal
  - ➔ AreaOfShapes
- **Search**
  - ➔ LinearSearch
  - ➔ BinarySearch
- **Sorting**
  - ➔ BubbleSort
  - ➔ InsertionSort

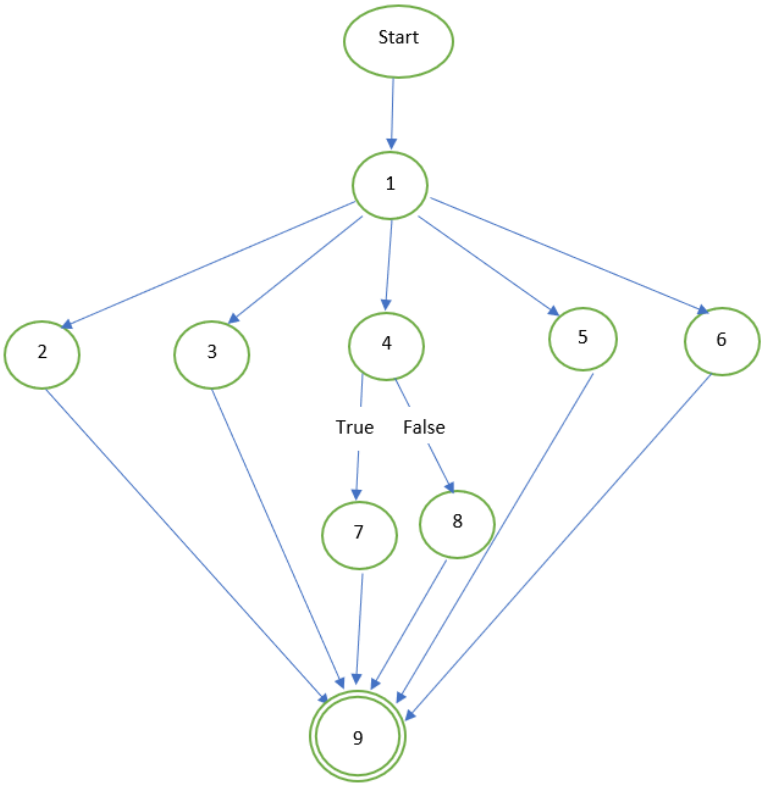
CONTROL FLOW GRAPHS FOR SOURCE CODES

A. ARITHMETIC

1. Operations.java

```java
public class Operations
{
    12 usages  kratos12310
    public String run(double num1,double num2,int c)
    {
            double number1;
            double number2;
            double result;
            String outp;
            Scanner sc = new Scanner(System.in);
            number1=num1;
            number2=num2;
            int ch=c;

            System.out.println("\nHere are your options:");
            System.out.println("\n1. Addition, 2. Subtraction, 3. Division, 4. Multiplication");
            //ch = sc.nextInt();
            switch (ch)
            {
                case 1:
                    result= number1 + number2;
                    outp=result+"";
                    break;

                case 2:
                    result= number1 - number2;
                    outp=result+"";
                    break;
```

```java
                case 3:
                    if(number2==0)
                    {
                        outp = "Divide By Zero";
                        break;
                    }
                    else
                    {
                        result = number1 / number2;
                        outp = result + "";
                        break;
                    }
                case 4:
                    result = number1 * number2;
                    outp=result+"";
                    break;
                default:
                    outp= "Invalid Input";
                    break;
            }
            System.out.println("Your answer is " + outp );
            return outp;
        }
}
```



BLOCK	Lines
1	9-21
2	23-26
3	28-31
4	33-34
5	45-48
6	49-51
7	35-38
8	39-44
9	53-54

Edge Coverage TR: [1,2],[1,3],[1,4],[1,5],[1,6],[4,7],[4,8],[2,9],[3,9],[7,9],[8,9],[5,9],[6,9]
Prime Path Coverage TR: [1,2,9],[1,3,9],[1,4,7,9],[1,4,8,9],[1,5,9],[1,6,9]

The Test path suite that does both edge coverage and prime path coverage is:
[1,2,9],[1,3,9],[1,4,7,9],[1,4,8,9],[1,5,9],[1,6,9]

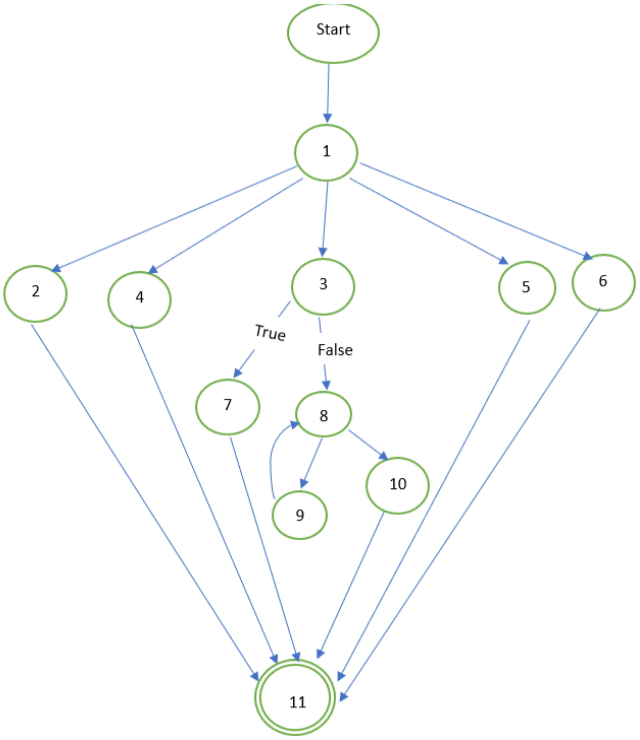
JUnit Test cases covering above mentioned Test paths:

```
19      @Test
20      public void runTest(){
21          assertEquals( expected: "3.0",a.run( num1: 1, num2: 2, c: 1), message: "Addition test successful");
22          assertNotEquals( unexpected: "6",a.run( num1: 1, num2: -2, c: 1));
23
24          assertEquals( expected: "2.0",a.run( num1: 5, num2: 3, c: 2), message: "Subtraction test successful");
25          assertNotEquals( unexpected: "6",a.run( num1: 5, num2: -3, c: 2));
26
27          assertEquals( expected: "4.0",a.run( num1: 12, num2: 3, c: 3), message: "Division test successful");
28          assertNotEquals( unexpected: "3",a.run( num1: 12, num2: -3, c: 3));
29
30          assertEquals( expected: "Divide By Zero",a.run( num1: 1, num2: 0, c: 3), message: "Divide By zero test successful");
31          assertNotEquals( unexpected: "0",a.run( num1: 1, num2: 0, c: 3));
32
33          assertEquals( expected: "15.0",a.run( num1: 3, num2: 5, c: 4), message: "Multiplication test successful");
34          assertNotEquals( unexpected: "15",a.run( num1: 3, num2: -5, c: 4));
35
36          assertEquals( expected: "Invalid Input",a.run( num1: 3, num2: 5, c: 5), message: "Invalid Choice test successful");
37          assertNotEquals( unexpected: "8",a.run( num1: 3, num2: 5, c: 5));
38      }
```

2. AdvancedOperations.java

```
public String run(int cc,double a,int b,double c,double d, double e)
{
        //System.out.println("Enter your Choice: \n 1.Square Root \n 2.Factorial \n 3.Natural Log
        int ch=cc;
        Scanner sc = new Scanner(System.in);
        String temp="";
        switch (ch)
        {
                case 1:
                    System.out.println("Enter a number to find its square root: ");
                    //double a = sc.nextDouble();
                    temp=sqrt(a)+"";
                    System.out.println("Square root of " + a + " is: " + temp);
                    break;
                case 2:
                    System.out.println("Enter a number to find its factorial: ");
                   // int b = sc.nextInt();
                    double fact = 1;
                    if (b < 0)
                    {
                        System.out.println("Factorial of a negative number is not possible!");
                        temp="Neg";
                        break;
                    }
```

```
                    else
                    {
                        for (int i = 1; i <= b; i++)
                        {
                            fact = fact * i;
                        }
                        System.out.println("Factorial of " + b + " is: " + fact);
                        temp = fact + "";
                        break;
                    }
                case 3:
                    System.out.println("Enter number to find its natural log: ");
                    //double c = sc.nextDouble();
                    System.out.println("Natural log of " + c + " is: " + log(c));
                    temp =log(c)+"";
                    break;
                case 4:
                    System.out.print("Enter the number: ");
                    //double d = sc.nextDouble();
                    System.out.print("Enter power: ");
                    //double e = sc.nextDouble();
                    System.out.println("The number " + d + " raised to power " + e + " is: " + pow(d, e));
                    temp= pow(d,e)+"";
                    break;
                default:
                    System.out.println("Invalid Choice");
                    temp="Invalid";
                    break;
```



BLOCK	Lines
1	10-13
2	15-20
3	21-25
4	42-46
5	47-54
6	55-58
7	27-29
8	33
9	35
10	37-39
11	60

Edge Coverage TR:
[1,2],[1,3],[1,4],[1,5],[1,6],[3,7],[3,8],[8,9],[9,8],[8,10],[2,11],[4,11],[7,11],[10,11],[5,11],[6,11]
Prime Path Coverage TR:
[1,2,11],[1,4,11],[1,3,7,11],[1,3,8,10,11],[1,3,8,9],[8,9,8],[9,8,9],[1,5,11],[1,6,11],[9,8,10,11]

The Test path suite that does both edge coverage and prime path coverage is:
[1,2,11], [1,4,11], [1,3,7,11], [1,5,11], [1,6,11], [1,3,8,10,11], [1,3,8,9,8,9,8,10,11]

JUnit Test cases covering above mentioned Test paths

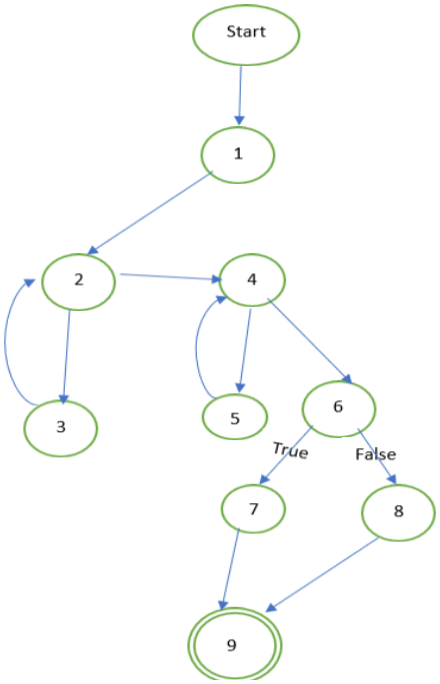
```
@Test
public void runTest(){
    assertEquals( expected: "2.0",a.run( cc: 1, a: 4, b: 0, c: 0, d: 0, e: 0), message: "Square Root test successful");
    assertNotEquals( unexpected: "2.0",a.run( cc: 1, a: -4, b: 0, c: 0, d: 0, e: 0));

    assertEquals( expected: "Neg",a.run( cc: 2, a: 0, b: -1, c: 0, d: 0, e: 0), message: "Factorial test successful");
    assertNotEquals( unexpected: "1.0",a.run( cc: 2, a: 0, b: -1, c: 0, d: 0, e: 0));

    assertEquals( expected: "1.0",a.run( cc: 2, a: 0, b: 0, c: 0, d: 0, e: 0), message: "Factorial test successful");
    assertNotEquals( unexpected: "",a.run( cc: 2, a: 0, b: 0, c: 0, d: 0, e: 0));

    assertEquals( expected: "2.0",a.run( cc: 1, a: 4, b: 0, c: 0, d: 0, e: 0), message: "Factorial test successful");
    assertNotEquals( unexpected: "2.0",a.run( cc: 1, a: -4, b: 0, c: 0, d: 0, e: 0));

    assertEquals( expected: "2.302585092994046",a.run( cc: 3, a: 0, b: 0, c: 10, d: 0, e: 0), message: "Natural Log test suc
    assertNotEquals( unexpected: "2.302585092994046",a.run( cc: 3, a: 0, b: 0, c: -10, d: 0, e: 0));

    assertEquals( expected: "8.0",a.run( cc: 4, a: 0, b: 0, c: 0, d: 2, e: 3), message: "Power test successful");
    assertNotEquals( unexpected: "8.0",a.run( cc: 1, a: 0, b: 0, c: 0, d: 2, e: -3));

    assertEquals( expected: "Invalid",a.run( cc: 5, a: 0, b: 0, c: 0, d: 0, e: 0), message: "Invalid test successful");
    assertNotEquals( unexpected: "2.0",a.run( cc: 5, a: 0, b: 0, c: 0, d: 0, e: 0));
```

B. MATHS

1. ArmstrongNum.java

```java
public boolean armstrong(int n)
{
    boolean t;
    Scanner sc= new Scanner(System.in);
    //System.out.print("Enter a num: ");
    //n=sc.nextInt();
    int temp;
    int digits=0;
    int last=0;
    int sum=0;
    temp=n;
    while(temp>0)
    {
        temp = temp/10;
        digits++;
    }
    temp = n;
    while(temp>0)
    {
        last = temp % 10;
        sum +=  (Math.pow(last, digits));
        temp = temp/10;
    }
```

```java
        if(n==sum)
        {
          //  System.out.println("It is an Armstrong Num");
            t= true;
        }
        else
        {
          //  System.out.println("It is not a Armstrong no");
            t= false;
        }
        return t;
    }
}
```



BLOCK	Lines
1	9-17
2	18
3	20-21
4	23-24
5	26-28
6	30
7	32-33
8	37-38
9	40

Edge Coverage TR: [1,2],[2,3],[3,2],[2,4],[4,5],[5,4],[4,6],[6,7],[6,8],[7,9],[8,9]
Prime Path Coverage TR:
[1,2,3],[2,3,2],[3,2,3],[1,2,4,5],[4,5,4],[5,4,5],[1,2,4,6,7,9],[1,2,4,6,8,9],[5,4,6,7,9],[5,4,6,8,9],
[3,2,4,5],[3,2,4,6,7,9],[3,2,4,6,8,9]

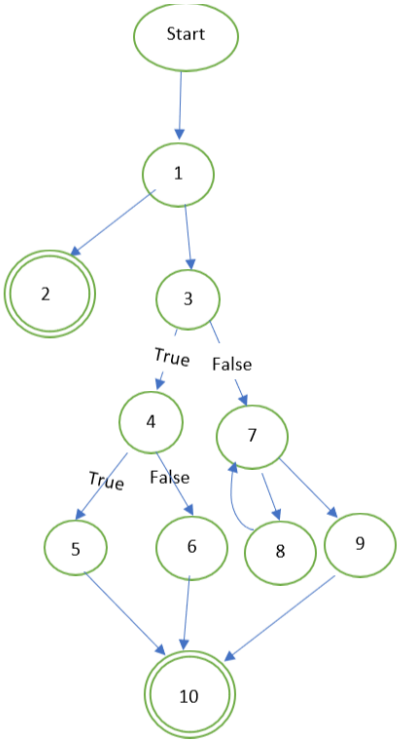
The Test path suite that does both edge coverage and prime path coverage is:
[1,2,3,2,3,2,4,5,4,5,4,6,7,9], [1,2,4,6,8,9], [1,2,4,6,7,9],[1,2,3,2,4,5,4,6,8,9]

JUnit Test cases covering above mentioned Test paths:

```
@Test
public void ArmstrongNumTest(){
    assertEquals( expected: true,a.armstrong( n: 153), message: "Armstrong Num test successful");
    assertNotEquals( unexpected: true,a.armstrong( n: -153));

    assertEquals( expected: false,a.armstrong( n: 154), message: "Armstrong Num test successful");
    assertNotEquals( unexpected: true,a.armstrong( n: 154));

    assertEquals( expected: false,a.armstrong( n: -1), message: "Armstrong Num test successful");
    assertNotEquals( unexpected: true,a.armstrong( n: -1));

    assertEquals( expected: true,a.armstrong( n: 0), message: "Armstrong Num test successful");
    assertNotEquals( unexpected: false,a.armstrong( n: 0));
```

2. HCF.java

```java
public int hcf(int n1,int n2)
{
    int ans = 0;
    int a1=n1, a2=n2;
    Scanner sc = new Scanner(System.in);
    System.out.println("The First Num is: "+n1);
    //num1 = sc.nextInt();
    System.out.println("The Second Num is: "+n2);
    //num2 = sc.nextInt();
    if (a1 < 0 || a2 < 0)
    {
        System.out.println("Negative Number Entered");
        return -1;
    }
    else
    {
        if (a1 == 0 || a2 == 0)
        {
            if (a1 < a2)
            {
                ans = a2 - a1;
            }
            else
            {
                ans = a1 - a2;
            }
        }
        else
        {
            while (a1 % a2 != 0)
            {
                int remainder = a1 % a2;
                a1 = a2;
                a2 = remainder;
            }
            ans = a2;
        }
        //System.out.println("Result  : " + ans);
        return ans;
    }
```



BLOCK	Lines
1	10-17
2	19-20
3	24
4	26
5	28
6	32
7	37
8	39-41
9	43
10	45-46

Edge Coverage TR: [1,2],[1,3],[3,4],[3,7],[4,5],[4,6],[6,10],[5,10],[7,8],[8,7],[7,9],[9,10]
Prime Path Coverage TR:
[1,2],[1,3,4,5,10],[1,3,4,6,10],[1,3,7,9,10],[1,3,7,8],[7,8,7],[8,7,8],[8,7,9,10]

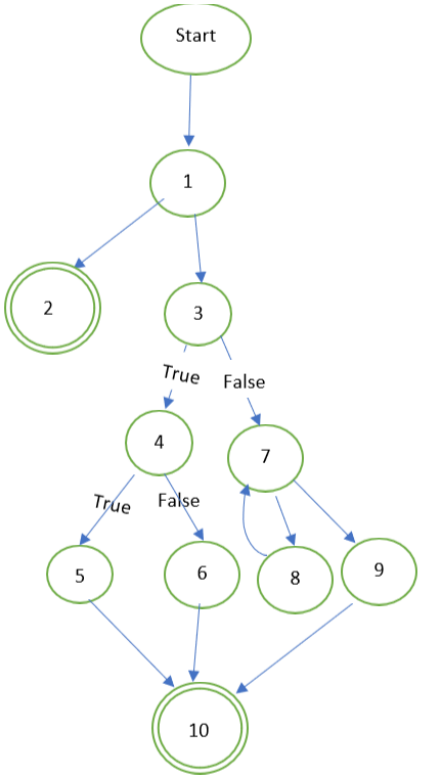
The Test path suite that does both edge coverage and prime path coverage is:
[1,2], [1,3,4,5,10], [1,3,4,6,10], [1,3,7,9,10], [1,3,7,8,7,8,7,9,10]

JUnit Test cases covering above mentioned Test paths:

```
21      @Test
22      public void HCFTest(){
23          assertEquals( expected: -1,a.hcf(-1,2), message: "HCF test successful");
24          assertNotEquals( unexpected: 1,a.hcf(-1,1));
25
26          assertEquals( expected: 1,a.hcf(0,1), message: "HCF test successful");
27          assertNotEquals( unexpected: 0,a.hcf(0,1));
28
29          assertEquals( expected: 1,a.hcf(1,0), message: "HCF test successful");
30          assertNotEquals( unexpected: 0,a.hcf(1,0));
31
32          assertEquals( expected: 4,a.hcf(4,4), message: "HCF test successful");
33          assertNotEquals( unexpected: 2,a.hcf(4,4));
34
35          assertEquals( expected: 1,a.hcf(4,3), message: "HCF test successful");
36          assertNotEquals( unexpected: 12,a.hcf(4,3));
```

3 LCM.java

```java
public int lcm(int n1,int n2)
{
    int ans = 0, a1=n1, a2=n2;
    Scanner sc = new Scanner(System.in);
    System.out.println("The First Num is: "+a1);
    //num1 = sc.nextInt();
    System.out.println("The Second Num is: "+a2);
    //num2 = sc.nextInt();
    int t1=a1,t2=a2;
    if (a1 < 0 || a2 < 0)
    {
        System.out.println("Number Entered is Negative");
        return -1;
    }
    else
    {
        if (a1 == 0 || a2 == 0)
        {
            if (a1 < a2)
            {
                ans = a2 - a1;
            }
            else
            {
                ans = a1 - a2;
            }
        }
        else
        {
            while (a1 % a2 != 0)
            {
                int rem = a1 % a2;
                a1 = a2;
                a2 = rem;
            }
            ans = a2;
        }
        ans=(t1/ans)*t2;
        //System.out.println("Result  : " + ans);
        return ans;
    }
}
```



BLOCK	Lines
1	10-17
2	19-20
3	24
4	26
5	28
6	32
7	37
8	39-41
9	43
10	45-49

Edge Coverage TR: [1,2],[1,3],[3,4],[3,7],[4,5],[4,6],[6,10],[5,10],[7,8],[8,7],[7,9],[9,10]
Prime Path Coverage TR:
[1,2],[1,3,4,5,10],[1,3,4,6,10],[1,3,7,9,10],[1,3,7,8],[7,8,7],[8,7,8],[8,7,9,10

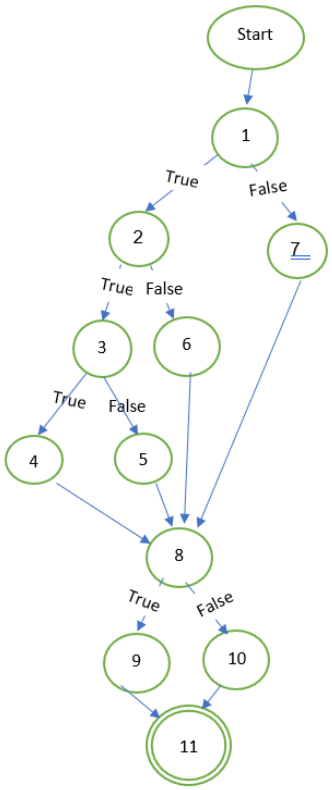
The Test path suite that does both edge coverage and prime path coverage is:
[1,2], [1,3,4,5,10], [1,3,4,6,10], [1,3,7,9,10], [1,3,7,8,7,8,7,9,10]

JUnit Test cases covering above mentioned Test paths:

```
20      @Test
21      public void LCMTest(){
22          assertEquals( expected: -1,a.lcm(-1,2), message: "LCM test successful");
23          assertNotEquals( unexpected: 1,a.lcm(-1,1));
24
25          assertEquals( expected: 0,a.lcm(0,1), message: "LCM test successful");
26          assertNotEquals( unexpected: 1,a.lcm(0,1));
27
28          assertEquals( expected: 0,a.lcm(1,0), message: "LCM test successful");
29          assertNotEquals( unexpected: 1,a.lcm(1,0));
30
31          assertEquals( expected: 4,a.lcm(4,4), message: "LCM test successful");
32          assertNotEquals( unexpected: 2,a.lcm(4,4));
33
34          assertEquals( expected: 12,a.lcm(4,3), message: "LCM test successful");
35          assertNotEquals( unexpected: 1,a.lcm(4,3));
```

4 LeapYear.java

```java
public class LeapYear
{
    8 usages  kratos12310
    public boolean checkLeap(int y)
    {
        int yr=y;
        boolean leap_year = false;
        Scanner sc=new Scanner(System.in);
        System.out.println("Year Entered is: "+yr);
        //year=sc.nextInt();
        if (yr % 4 == 0)
        {
            if (yr % 100 == 0)
            {
                if (yr % 400 == 0)
                {
                    leap_year = true;
                }
                else
                {
                    leap_year = false;
                }
            }
            else
            {
                leap_year = true;
            }
        }
        else
        {
            leap_year = false;
        }
        System.out.println("Result: ");
        if (leap_year)
        {
            System.out.println(yr + " :  Leap-year");
        }
        else
        {
            System.out.println(yr + " : Non Leap-year");
        }
        return leap_year;
```



BLOCK	Lines
1	9-14
2	16
3	18
4	20
5	24
6	29
7	34
8	36-37
9	39
10	43
11	45

Edge Coverage TR:
[1,2],[1,7],[2,3],[2,6],[3,5],[3,4],[4,8],[5,8],[6,8],[8,10],[7,8],[8,9],[10,11],[9,11]
Prime Path Coverage TR: [1,2,3,4,8,9,11], [1,2,3,4,8,10,11],[1,2,3,5,8,9,11],[1,2,3,5,8,10,11], [1,2,6,8,9,11],[1,2,6,8,10,11],[1,7,8,9,11],[1,7,8,10,11]

The Test path suite that covers edge coverage & prime path coverage is: [1,2,3,4,8,9,11], [1,2,3,4,8,10,11],[1,2,3,5,8,9,11],[1,2,3,5,8,10,11],[1,2,6,8,9,11],[1,2,6,8,10,11],[1,7,8,9,11],[1,7, 8,10,11]

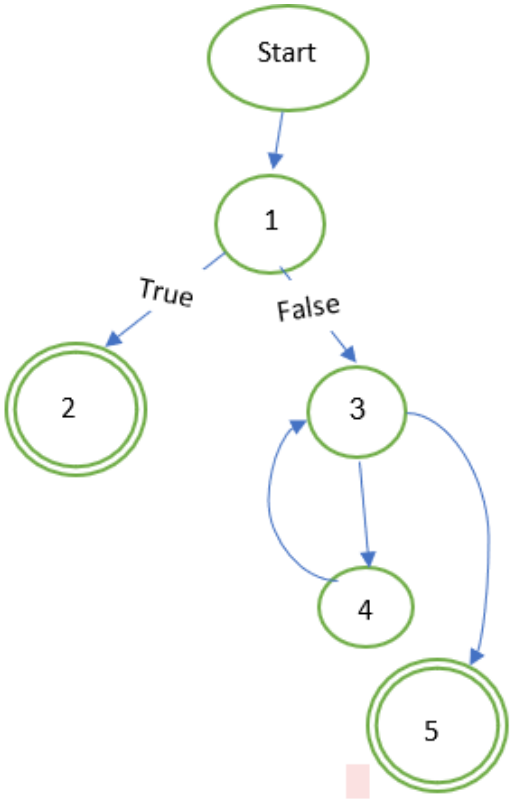
JUnit Test cases covering above mentioned Test paths except the infeasible prime paths:

```
22      @Test
23      public void LeapYearTest() {
24          assertEquals( expected: true, a.checkLeap( y: 1600),  message: "Leap Year test successful");
25          assertNotEquals( unexpected: false, a.checkLeap( y: 1600));
26
27          assertEquals( expected: false,a.checkLeap( y: 1700), message: "Leap Year test successful");
28          assertNotEquals( unexpected: true,a.checkLeap( y: 1700));
29
30          assertEquals( expected: true,a.checkLeap( y: 1988), message: "Leap Year test successful");
31          assertNotEquals( unexpected: false,a.checkLeap( y: 1988));
32
33          assertEquals( expected: false,a.checkLeap( y: 2021), message: "Leap Year test successful");
34          assertNotEquals( unexpected: true,a.checkLeap( y: 2021));
```

5. NthFibo.java

```
8       public int fib(int nn)
9       {
10          int n=nn;
11          System.out.println("Enter a Num to find the Nth fibo Num ");
12          Scanner sc=new Scanner(System.in);
13          // n=sc.nextInt();
14          int a = 0;
15          int b = 1;
16          int c;
17          if (n == 0)
18          {
19              System.out.println("The Nth fibo number is 0");
20              return a;
21          }
22          int i=2;
23          while(i<=n)
24          {
25              c = a + b;
26              a = b;
27              b = c;
28              i++;
29          }
30          System.out.println("The Nth fibo number is "+b);
31          return b;
```

BLOCK	Lines
1	10-17
2	19-20
3	22-23
4	25-28
5	30-31



Edge Coverage TR: [1,2],[1,3],[3,4],[4,3],[3,5]
Prime Path Coverage TR: [1,2],[1,3,5],[1,3,4],[3,4,3],[4,3,4],[4,3,5]

The Test path suite that covers edge coverage & prime path coverage is:
[1,2], [1,3,5], [1,3,4,3,4,3,5]

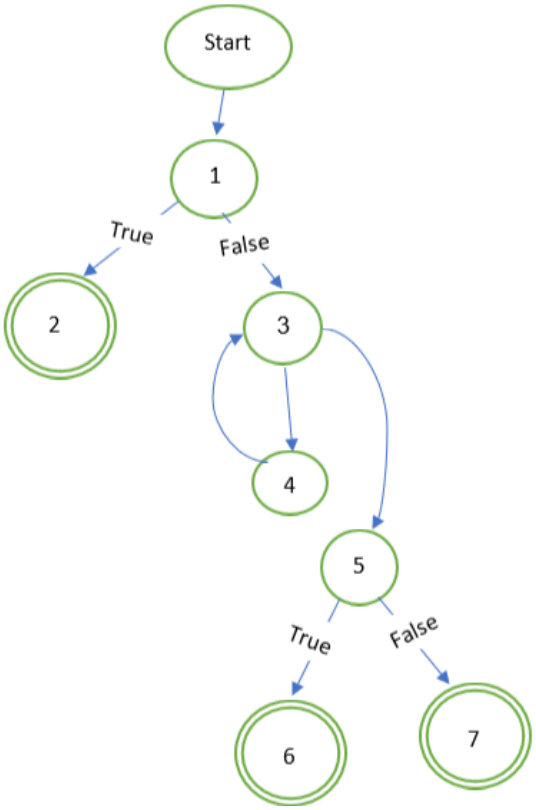
JUnit Test cases covering above mentioned Test paths:

```
20      @Test
21      public void FiboTest() {
22          assertEquals( expected: 0, a.fib( nn: 0),  message: "Fibo test successful");
23          assertNotEquals( unexpected: 1, a.fib( nn: 0));
24
25          assertEquals( expected: 1, a.fib( nn: 1),  message: "Fibo test successful");
26          assertNotEquals( unexpected: 0, a.fib( nn: 1));
27
28          assertEquals( expected: 2, a.fib( nn: 3),  message: "Fibo test successful");
29          assertNotEquals( unexpected: 1, a.fib( nn: 3));
```

6. PalindromeCheck.java

```
7       public int palin(int nn)
8       {
9           int r;
10          int sum = 0;
11          int temp;
12          int n=nn ;
13          Scanner sc=new Scanner(System.in);
14          // System.out.println("Enter The Num to check");
15          //n=sc.nextInt();
16          temp = n;
17          if(n<0)
18          {
19              System.out.println("Wrong Input");
20              return -2;
21          }
22          while (n > 0)
23          {
24              r = n % 10;  //getting remainder
25              sum = (sum * 10) + r;
26              n = n / 10;
27          }
```

```
28          if (temp == sum)
29          {
30              System.out.println("palindrome number ");
31              return 1;
32          }
33          else
34          {
35              System.out.println("not palindrome");
36              return -1;
37          }
```



BLOCK	Lines
1	9-17
2	19-20
3	22
4	24-26
5	28
6	30-31
7	35-36

Edge Coverage TR: [1,2],[1,3],[3,4],[4,3],[3,5],[5,6],[5,7]
Prime Path Coverage TR: [1,2],[1,3,4],[3,4,3],[4,3,4],[1,3,5,7],[1,3,5,6],[4,3,5,6][4,3,5,7]

The Test path suite that covers edge coverage & prime path coverage is:
[1,2], [1,3,5,6], [1,3,5,7], [1,3,4,3,5,7], [1,3,4,3,4,3,5,6]

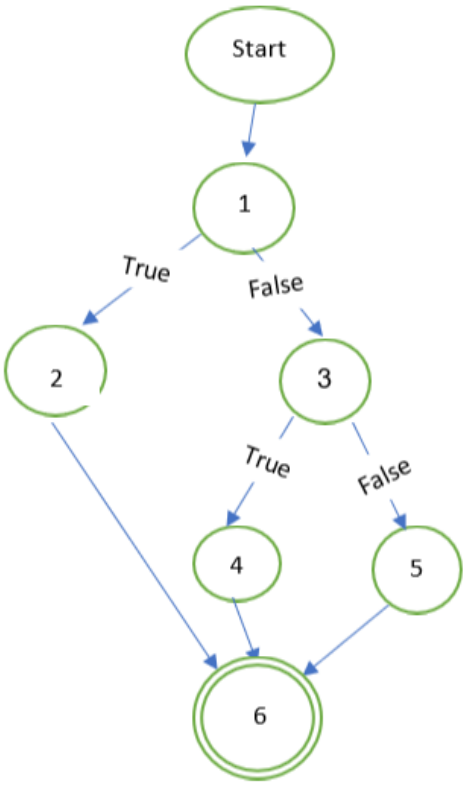
JUnit Test cases covering above mentioned Test paths excluding infeasible prime paths :

```
22      @Test
23      public void PalindromeCheckTest() {
24
25          assertEquals( expected: -2, a.palin( nn: -3),  message: "Palindrome test successful");
26          assertNotEquals( unexpected: 1, a.palin( nn: -3));
27
28          assertEquals( expected: 1, a.palin( nn: 212),  message: "Palindrome test successful");
29          assertNotEquals( unexpected: -1, a.palin( nn: 212));
30
31          assertEquals( expected: -1, a.palin( nn: 345),  message: "Palindrome test successful");
32          assertNotEquals( unexpected: 1, a.palin( nn: 345));
33
34          assertEquals( expected: 1, a.palin( nn: 0),  message: "Palindrome test successful");
35          assertNotEquals( unexpected: -1, a.palin( nn: 0));
```

7. MaxThreeNos.java

```java
public int maxNum(int n1, int n2, int n3) {
    int num1 = n1;
    int num2 = n2;
    int num3 = n3;
    int max=-1;
    if (num1 >= num2 && num1 >= num3)
    {
        System.out.println(num1 + " is the maximum number.");
        max=num1;
    }
    else if (num2 >= num1 && num2 >= num3)
    {
        System.out.println(num2 + " is the maximum number.");
        max=num2;
    }
    else
    {
        System.out.println(num3 + " is the maximum number.");
        max=num3;
    }
    return max;
}
```

BLOCK	Lines
1	6-10
2	12-13
3	15
4	17-18
5	22-23
6	25



Edge Coverage TR: [1,2],[1,3],[2,6],[3,4],[3,5],[4,6],[5,6]
Prime Path Coverage TR: [1,2],[1,3,4,6],[1,3,5,6]

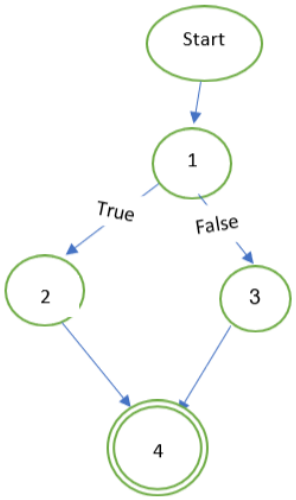
The Test path suite that covers edge coverage & prime path coverage is:
[1,2],[1,3,4,6],[1,3,5,6]

JUnit Test cases covering above mentioned Test paths is :

```
21      @Test
22      public void MaxThreeNosTest() {
23          assertEquals( expected: 30, a.maxNum(10,20,30),  message: "MaxNum test successful");
24          assertNotEquals( unexpected: 10, a.maxNum(10,20,30));
25
26          assertEquals( expected: 30, a.maxNum(10,30,20),  message: "MaxNum test successful");
27          assertNotEquals( unexpected: 10, a.maxNum(10,30,20));
28
29          assertEquals( expected: 30, a.maxNum(30,20,10),  message: "MaxNum test successful");
30          assertNotEquals( unexpected: 10, a.maxNum(30,20,10));
31      }
```

8. TriplePythoVal.java

```java
public boolean triplePythoCheck(int x,int y,int z) {

    boolean ans = true;

    int max = Math.max(x, Math.max(y, z));
    int min = Math.min(x, Math.min(y, z));
    int mid = x + y + z - max - min;

    if (min <= 0 || mid <= 0 || max <= 0)
    {
        ans =  false;
    }
    else
    {
        ans =  (min * min) + (mid * mid) == (max * max);
    }
    System.out.println("Result: "+ans);
    return ans;
```



BLOCK	Lines
1	7-13
2	15
3	19
4	21-22

Edge Coverage TR: [1,2],[1,3],[2,4],[3,4]
Prime Path Coverage TR: [1,2,4],[1,3,4]

The Test path suite that covers edge coverage & prime path coverage is:
[1,2,4], [1,3,4]

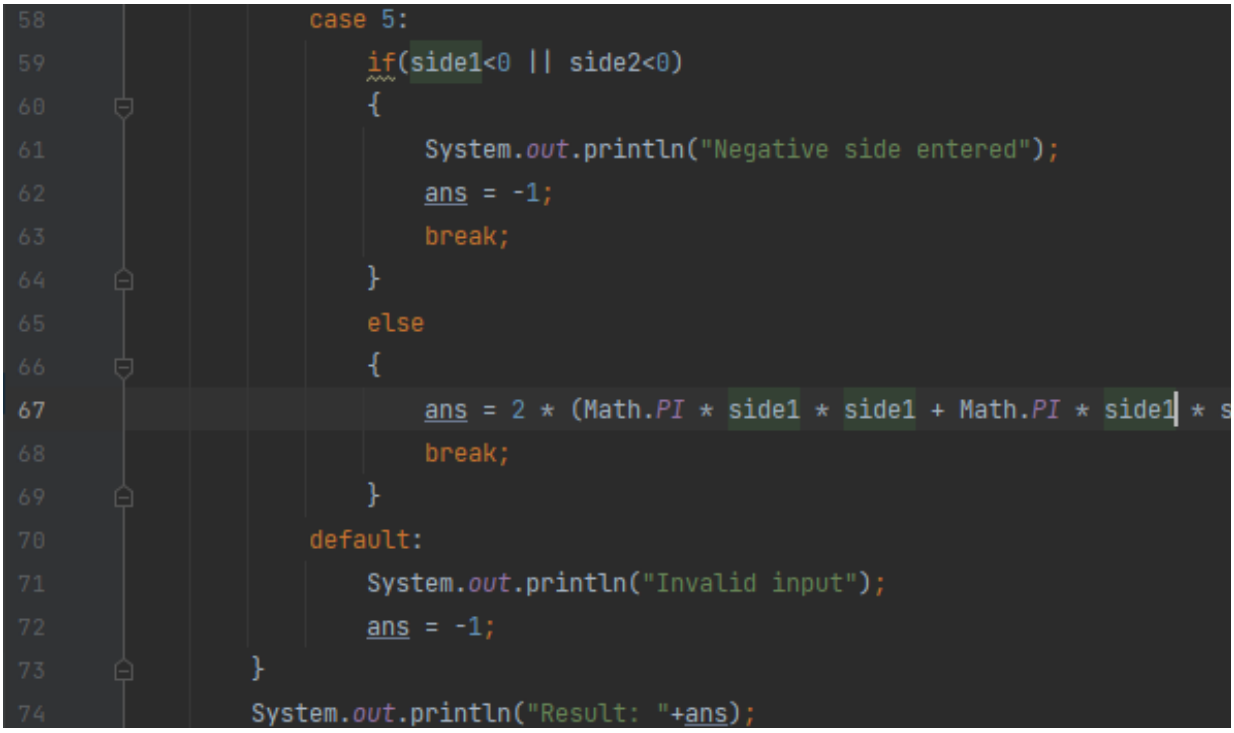
JUnit Test cases covering above mentioned Test paths is :

```java
@Test
public void TriplePythoTest() {

    assertEquals( expected: true, a.triplePythoCheck( x: 10,  y: 24,  z: 26),  message: "Pythagorean Triplet test successful");
    assertNotEquals( unexpected: true, a.triplePythoCheck( x: 10,  y: 24,  z: 27));

    assertEquals( expected: false, a.triplePythoCheck( x: 3,  y: 4,  z: 6),  message: "Pythagorean Triplet test successful");
    assertNotEquals( unexpected: false, a.triplePythoCheck( x: 3,  y: 4,  z: 5));
```

9. AreaOfShapes.java

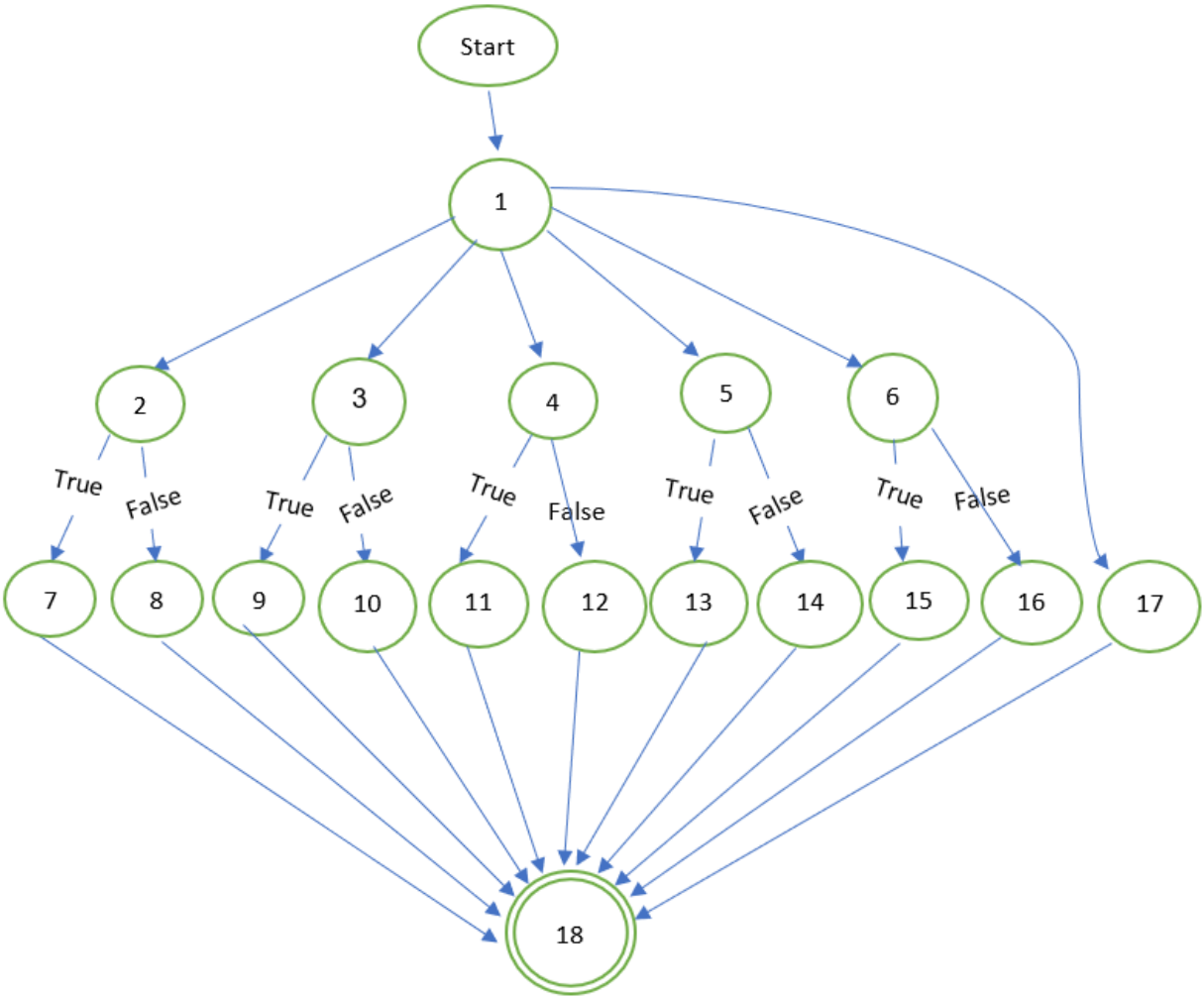
```java
public double areaCompute(int ch, double side1, double side2)
{
    double ans=0;
    switch(ch)
    {
        case 1:
            if(side1<0)
            {
                System.out.println("Negative side entered");
                ans = -1;
                break;
            }
            else
            {
                ans = 6 * side1 * side1;
                break;
            }
        case 2:
            if(side1<0)
            {
                System.out.println("Negative side entered");
                ans = -1;
                break;
            }
            else
            {
                ans = 4 * Math.PI * side1 * side1;
                break;
            }
```

```java
        case 3:
            if(side1<0 || side2<0)
            {
                System.out.println("Negative side entered");
                ans = -1;
                break;
            }
            else
            {
                ans = Math.PI * side1 * (side1 + Math.pow((side2 * side2 + side1 * side1), 0.5));
                break;
            }
        case 4:
            if(side1<0)
            {
                System.out.println("Negative side entered");
                ans = -1;
                break;
            }
            else
            {
                ans = 3 * Math.PI * side1 * side1;
                break;
            }
```



```
58              case 5:
59                  if(side1<0 || side2<0)
60                  {
61                      System.out.println("Negative side entered");
62                      ans = -1;
63                      break;
64                  }
65                  else
66                  {
67                      ans = 2 * (Math.PI * side1 * side1 + Math.PI * side1 * s
68                      break;
69                  }
70              default:
71                  System.out.println("Invalid input");
72                  ans = -1;
73          }
74          System.out.println("Result: "+ans);
```

BLOCK	Lines
1	6-7
2	9-10
3	21-22
4	34-35
5	46-47
6	58-59
7	12-14
8	18-19
9	24-26
10	30-31
11	37-39
12	43-44
13	49-51
14	55-56
15	61-63
16	67-68
17	70-72
18	74-75



Edge Coverage TR:
[1,2],[1,3],[1,4],[1,5],[1,6],[1,17],[2,7],[2,8],[3,9],[3,10],[4,11],[4,12],[5,13],[5,14],[6,15],[6,16],
[7,18],[8,18],[9,18],[10,18],[11,18],[12,18],[13,18],[14,18],[15,18],[16,18],[17,18]

Prime Path Coverage TR:
[1,2,7,18],[1,2,8,18],[1,3,9,18],[1,3,10,18],[1,4,11,18],[1,4,12,18],[1,5,13,18],[1,5,14,18],
[1,6,15,18],[1,6,16,18],[1,17,18]

The Test path suite that covers edge coverage & prime path coverage is:
[1,2,7,18], [1,2,8,18], [1,3,9,18], [1,3,10,18], [1,4,11,18], [1,4,12,18], [1,5,13,18], [1,5,14,18],
[1,6,15,18], [1,6,16,18], [1,17,18]

JUnit Test cases covering above mentioned Test paths is :

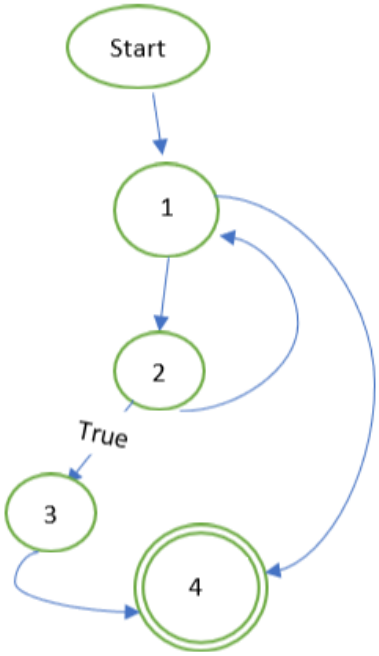
```
@Test
public void AreaTest() {
    assertEquals( expected: -1, a.areaCompute( ch: 6, side1: 20, side2: 30), message: "Area test successful");
    assertNotEquals( unexpected: 60, a.areaCompute( ch: 6, side1: 20, side2: 30));

    assertEquals( expected: 2400, a.areaCompute( ch: 1, side1: 20, side2: 20), message: "Area test successful");
    assertNotEquals( unexpected: -1, a.areaCompute( ch: 1, side1: 20, side2: 20));

    assertEquals( expected: -1, a.areaCompute( ch: 1, side1: -20, side2: -20), message: "Area test successful");
    assertNotEquals( unexpected: 400, a.areaCompute( ch: 1, side1: -20, side2: -20));

    assertEquals( expected: 5026.548245743669, a.areaCompute( ch: 2, side1: 20, side2: 20), message: "Area test successful");
    assertNotEquals( unexpected: -1, a.areaCompute( ch: 2, side1: 20, side2: 20));

    assertEquals( expected: -1, a.areaCompute( ch: 2, side1: -20, side2: 20), message: "Area test successful");
    assertNotEquals( unexpected: 5026.548245743669, a.areaCompute( ch: 2, side1: -20, side2: 20));

    assertEquals( expected: 3522.0717412637127, a.areaCompute( ch: 3, side1: 20, side2: 30), message: "Area test successful");
    assertNotEquals( unexpected: -1, a.areaCompute( ch: 3, side1: 20, side2: 30));

    assertEquals( expected: -1, a.areaCompute( ch: 3, side1: -20, side2: 30), message: "Area test successful");
    assertNotEquals( unexpected: 3522.0717412637127, a.areaCompute( ch: 3, side1: -20, side2: 30));

    assertEquals( expected: 3769.9111843077517, a.areaCompute( ch: 4, side1: 20, side2: 20), message: "Area test successful");
    assertNotEquals( unexpected: -1, a.areaCompute( ch: 4, side1: 20, side2: 20));

    assertEquals( expected: -1, a.areaCompute( ch: 4, side1: -20, side2: -20), message: "Area test successful");
    assertNotEquals( unexpected: 3769.9111843077517, a.areaCompute( ch: 4, side1: -20, side2: -20));

    assertEquals( expected: 6283.185307179587, a.areaCompute( ch: 5, side1: 20, side2: 30), message: "Area test successful");
    assertNotEquals( unexpected: -1, a.areaCompute( ch: 5, side1: 20, side2: 30));

    assertEquals( expected: -1, a.areaCompute( ch: 5, side1: -20, side2: 30), message: "Area test successful");
    assertNotEquals( unexpected: 6283.185307179587, a.areaCompute( ch: 6, side1: -20, side2: 30));
}
```

C. SEARCHING

1. LinearSearch.java

```
5   @       public int linearS(int val[], int num)
6           {
7
8               int res = -1;
9               for (int pos = 0; pos < val.length; pos++)
10              {
11                  if (val[pos] == num)
12                  {
13                      res = pos;
14                      break;
15                  }
16
17              }
18
19              System.out.println("Result: "+res);
20              return res;
21          }
```

BLOCK	Lines
1	8-9
2	11
3	13-14
4	19-20



Edge Coverage TR: [1,2],[2,3],[2,1],[3,4],[1,4]
Prime Path Coverage TR: [1,2,3,4],[1,2,1],[2,1,2],[2,1,4]

The Test path suite that covers edge coverage & prime path coverage is:
[1,2,1,2,1,2,3,4], [1,2,1,4]

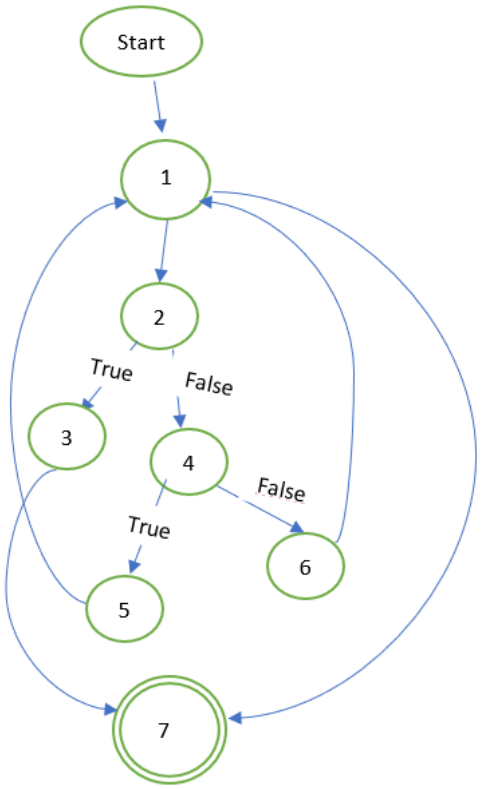
JUnit Test cases covering above mentioned Test paths is :

```
21      @Test
22      public void linearSearchTest() {
23          int arr[] = new int[]{-5,13,-4,21,16};
24
25          assertEquals( expected: 2, l.linearS(arr,  num: -4),  message: "Linear Search Test successful");
26          assertNotEquals( unexpected: -1, l.linearS(arr,  num: -4));
27
28          assertEquals( expected: -1, l.linearS(arr,  num: 12),  message: "Linear Search Test successful");
29          assertNotEquals( unexpected: 1, l.linearS(arr,  num: 12));
30          }
```

2. BinarySearch.java

```
5   @       public int binaryS (int val[], int num)
6           {
7               int left = 0;
8               int right = val.length - 1;
9               int res=-1;
10              while (left <= right)
11              {
12                  int mid = left + (right - left) / 2;
13                  if (val[mid] == num)
14                  {
15                      res =  mid;
16                      break;
17                  }
18                  if (val[mid] < num)
19                  {
20                      left = mid + 1;
21                  }
22                  else
23                  {
24                      right = mid - 1;
25                  }
26              }
27              System.out.println("Result: "+res);
28              return res;
```

BLOCK	Lines
1	7-10
2	12-13
3	15-16
4	18
5	20
6	24
7	27-28



Edge Coverage TR: [1,2],[2,3],[2,4],[3,7],[4,5],[4,6],[1,7],[5,1],[6,1]
Prime Path Coverage TR: [1,2,3,4,5,1],[1,2,4,6,1],[2,4,5,1,2],[2,4,6,1,2],[4,5,1,2,4],[4,6,1,2,4],[4,5,1,2,3,7],[4,6,1,2,3,7],[2,4,5,1,7],[2,4,6,1,7],[5,1,2,4,5],[5,1,2,4,6],[6,1,2,4,6],[6,1,2,4,5]

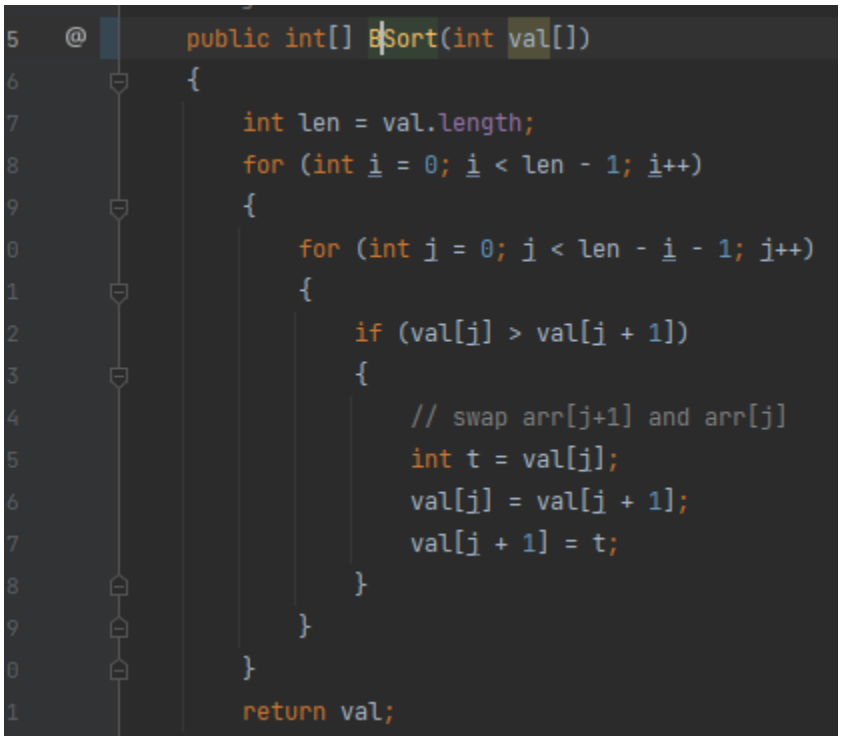
The Test path suite that covers edge coverage & prime path coverage is:
[1,2,4,5,1,2,4,6,1,2,3,7], [1,2,4,6,1,2,4,5,1,2,3,7], [1,2,4,6,1,2,4,6,1,7], [1,2,4,5,1,2,4,5,1,7]

JUnit Test cases covering above mentioned Test paths is :

```
23      @Test
24      public void binarySearchTest() {
25          int arr[] = new int[]{6, 8, 12, 18, 21, 54, 57, 59, 60, 62, 65};
26
27          assertEquals( expected: 7, b.binaryS(arr,  num: 59),  message: "Binary Search Test successful");
28          assertNotEquals( unexpected: -1, b.binaryS(arr,  num: 59));
29
30          assertEquals( expected: 4, b.binaryS(arr,  num: 21),  message: "Binary Search Test successful");
31          assertNotEquals( unexpected: -1, b.binaryS(arr,  num: 21));
32
33          assertEquals( expected: 9, b.binaryS(arr,  num: 62),  message: "Binary Search Test successful");
34          assertNotEquals( unexpected: -1, b.binaryS(arr,  num: 62));
35
36          assertEquals( expected: 2, b.binaryS(arr,  num: 12),  message: "Binary Search Test successful");
37          assertNotEquals( unexpected: -1, b.binaryS(arr,  num: 12));
```

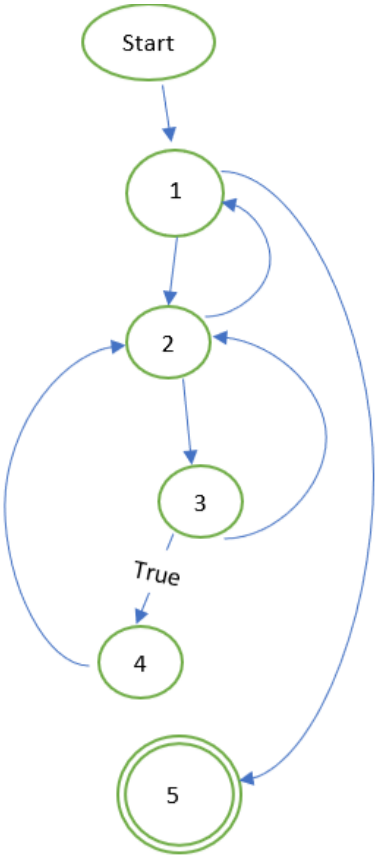
D. SORTING

1. BubbleSort.java



```
5   @       public int[] BSort(int val[])
6           {
7               int len = val.length;
8               for (int i = 0; i < len - 1; i++)
9               {
0                   for (int j = 0; j < len - i - 1; j++)
1                   {
2                       if (val[j] > val[j + 1])
3                       {
4                           // swap arr[j+1] and arr[j]
5                           int t = val[j];
6                           val[j] = val[j + 1];
7                           val[j + 1] = t;
8                       }
9                   }
0               }
1               return val;
```

BLOCK	Lines
1	7-8
2	10
3	12
4	14-17
5	21



Edge Coverage TR: [1,2],[2,3],[2,1],[3,2],[3,4],[4,2],[1,5]
Prime Path Coverage TR:
[1,2,3,4],[1,2,1],[2,1,2],[2,3,4,2],[2,3,2],[3,4,2,3],[3,2,3],[3,2,1,5],[3,4,2,1,5],[4,2,3,4]

The Test path suite that covers edge coverage & prime path coverage is:
[1,2,1,2,3,4,2,3,2,3,2,1,5], [1,2,3,4,2,3,4,2,1,5]

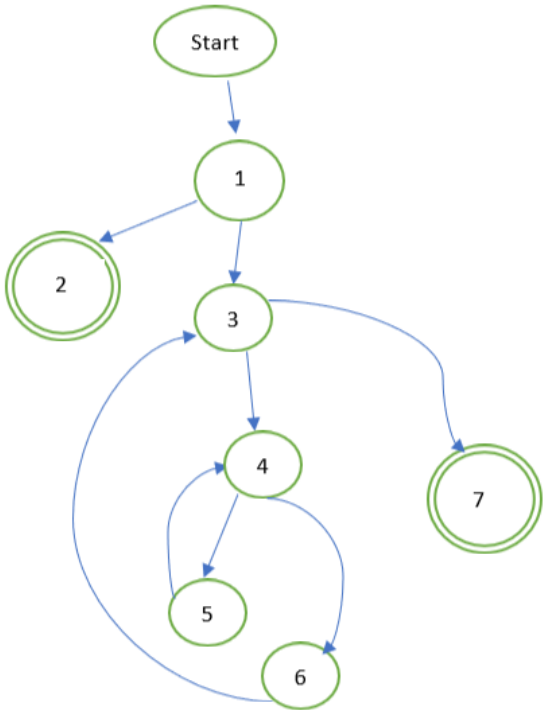
JUnit Test cases covering above mentioned Test paths is :

```
21      @Test
22      public void bubbleSortTest() {
23          int arr1[] = new int[]{6, 7, 9, 8, 10,11,15};
24          int arr2[] = new int[]{6, 7, 8, 9, 10,11,15};
25
26          int arr3[] = new int[]{7, 6, 5, 9};
27          int arr4[] = new int[]{5, 6, 7, 9};
28
29          assertArrayEquals(arr2, b.BSort(arr1), message: "Test Success");
30          assertArrayEquals(arr4, b.BSort(arr3), message: "Test Success");
31      }
```

2. InsertionSort.java

```java
public int[] ISort(int [] val)
{
    int len = val.length;
    if(len==0)
        return null;
    for (int i = 1; i < len; ++i)
    {
        int piv =val[i];
        int j = i - 1;

        while (j >= 0 && val[j] > piv)
        {
            val[j + 1] = val[j];
            j = j - 1;
        }
        val[j + 1] = piv;
    }
    return val;
}
```

BLOCK	Lines
1	6-7
2	8
3	9
4	11-14
5	16-17
6	19
7	21



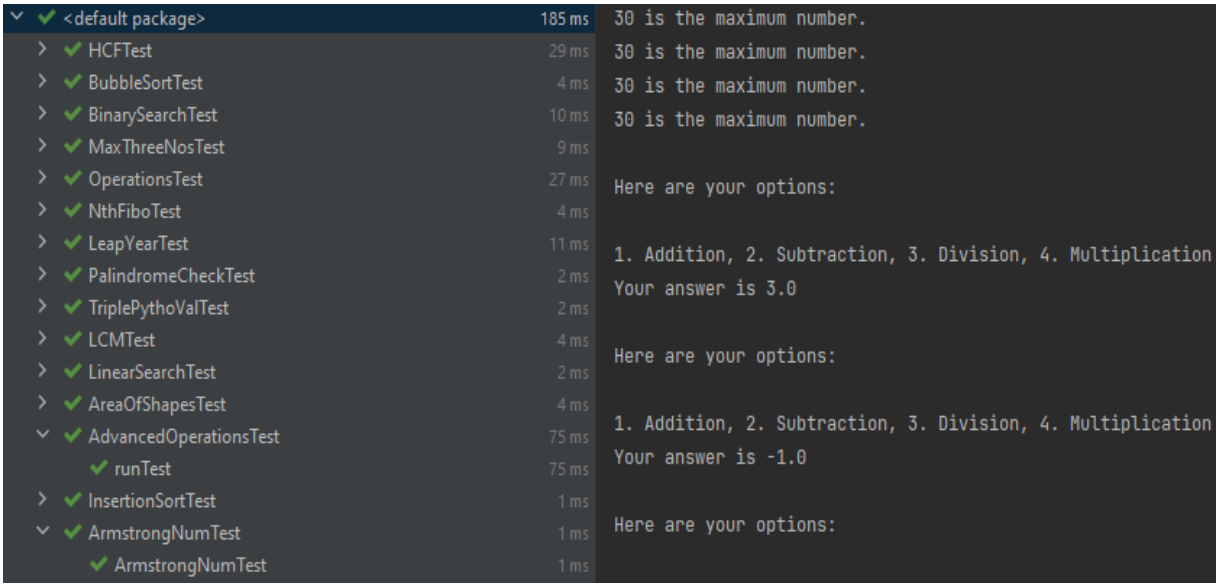
Edge Coverage TR: [1,2],[1,3],[3,7],[3,4],[4,5],[5,4],[4,6],[6,3]
Prime Path Coverage TR:
[1,2],[1,3,7],[1,3,4,5],[1,3,4,6],[4,5,4],[5,4,5],[5,4,6,3,7],[3,4,6,3],[4,6,3,4],[6,3,4,5],[6,3,4,6]

The Test path suite that covers edge coverage & prime path coverage is:
[1,2], [1,3,7], [1,3,4,5,4,6,3,4,5,4,5,4,6,3,4,5,4,6,3,7], [1,3,4,6,3,4,6,3,7]

JUnit Test cases covering above mentioned Test paths is :

```
22      @Test
23      public void insertionSortTest() {
24          int arr1[] = new int[]{9, 6, 4, 8};
25          int arr2[] = new int[]{4, 6, 8, 9};
26          int arr3[] = new int[]{1};
27
28          assertArrayEquals(arr2, i.ISort(arr1));
29
30          assertArrayEquals(arr3, i.ISort(arr3));
31
32          assertArrayEquals(arr2, i.ISort(arr2));
33
34          assertNull(i.ISort(new int[]{}));
35      }
```

ALL TEST FILES:



CONTRIBUTING TEAM MEMBERS:

Astha Borkataky (MT2021027) : Formulated all the CFGs for all the source codes & Report Preparation.
Gaurav Kumar (MT2021046) : Test Requirement and Test Path formation for each source code to fulfill the mentioned Coverage criterias.
Satya Jyoti Das (MT2021120) : Implemented the source codes and the JUnit test case files with appropriate Test Case designing that fits the Test Paths formulated for each source code.