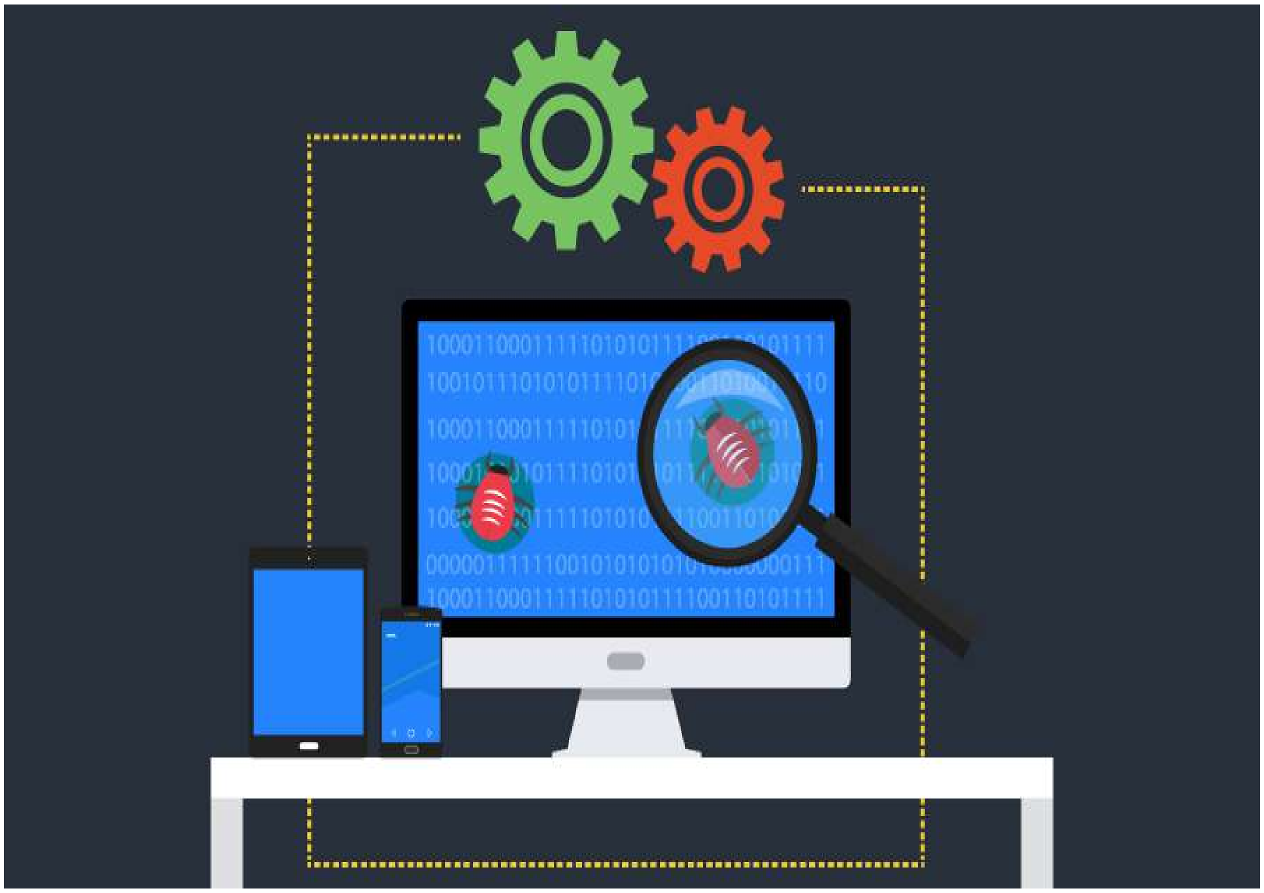


SOFTWARE TESTING PROJECT

[CONTROL FLOW GRAPH TESTING METHOD]

KUSHAL AGRAWAL
[MT2021071]

RUSHIKESH KALE
[MT2021111]

Overview

Aim of this project was to learn the control flow graph testing method.

Code Description

We have implemented various types of code for different searching algorithms, various types of sorting methods, and multiple array operations.

Tools Used

We implemented this project using Junit as a Testing tool.

We used IDE as Intellij.

We used GitHub for Version control.

About Testing Strategy

Control Flow Graph

It is the graphical representation of control flow or computation during the executions of the program.

Test Case Description

I. Edge Coverage

We have designed cases in such a way that it covers all the edges.

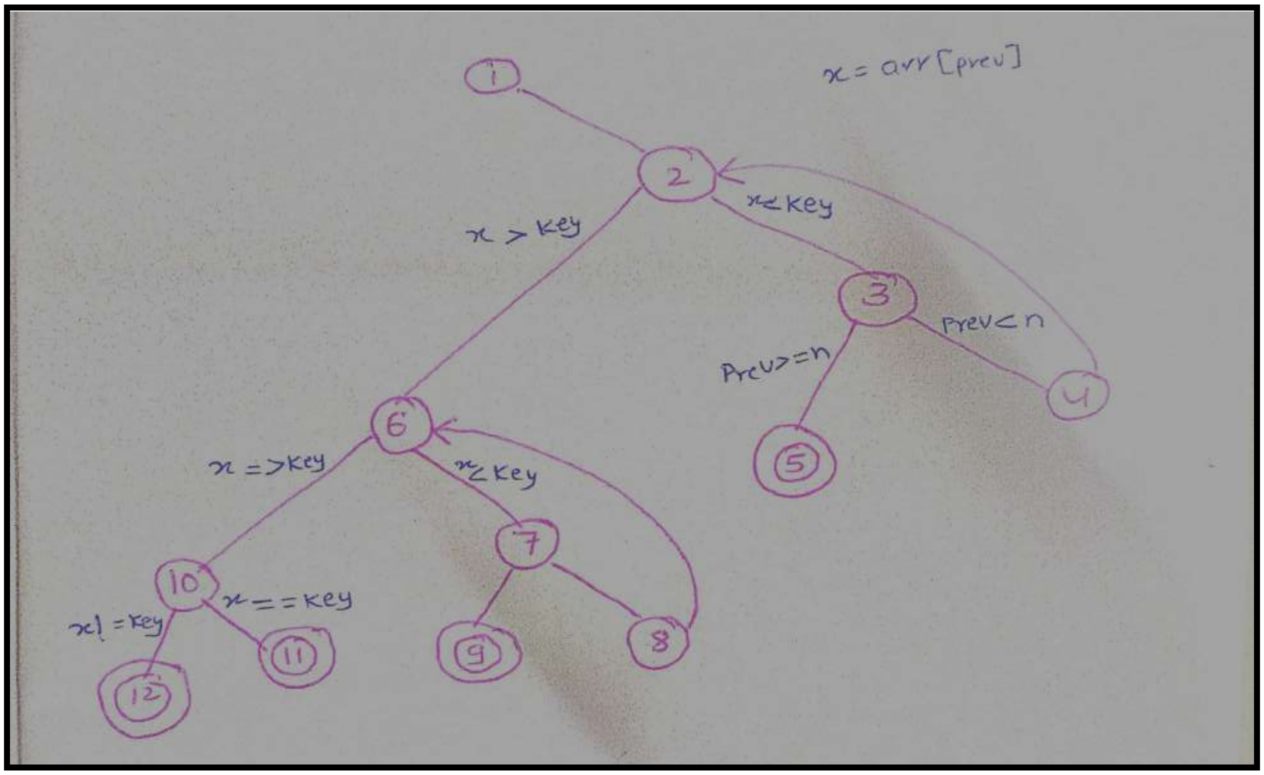
II. Prime Path Coverage

We have designed test cases in such a way that it covers all the

Prime paths.

1. Jump Search

Control Flow Graph for Jump Search code

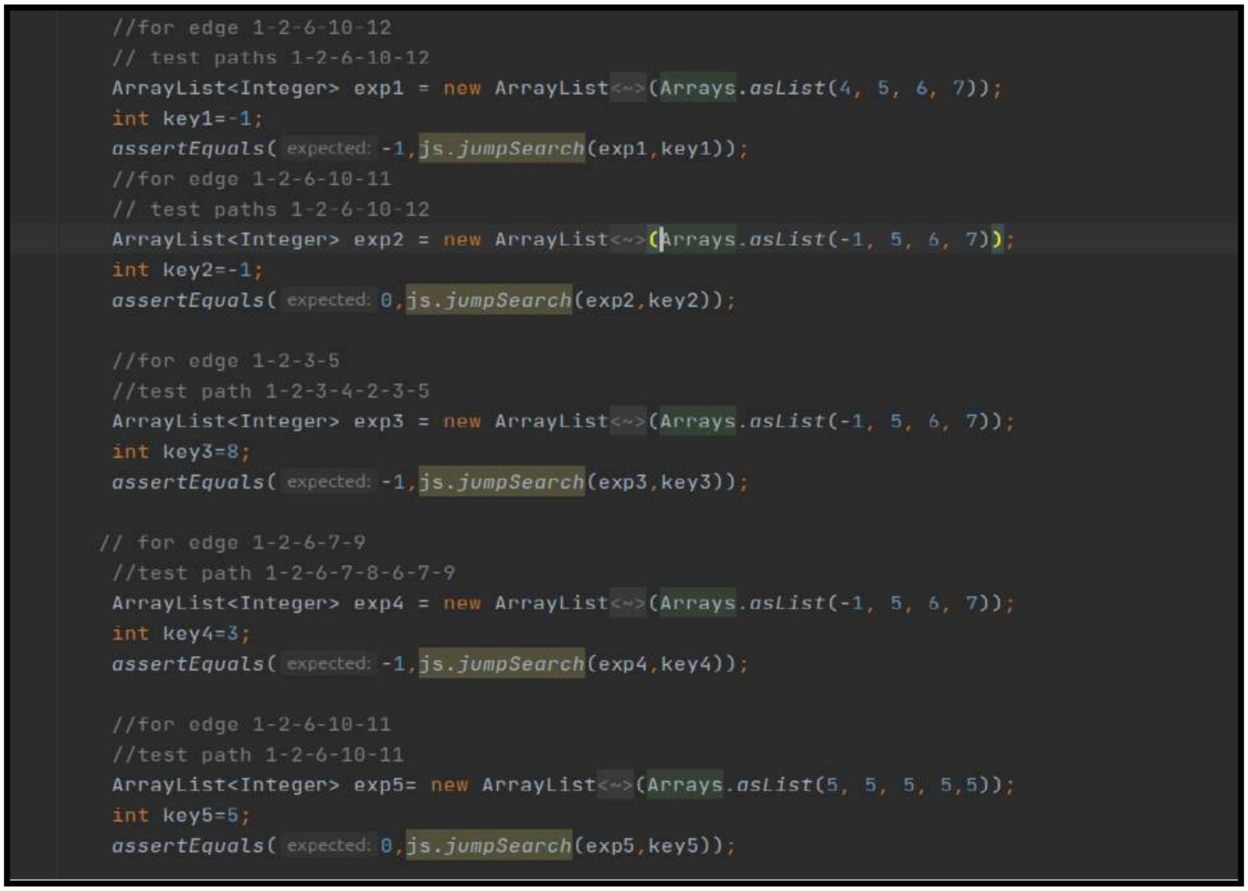


BLOCK NUMBER VS LINE NUMBER OF CODE

NODE	LINE NO
1	9-17
2	18
3	20-21
4	24
5	23
6	29
7	30

8	37
9	36
10	40
11	41
12	23

Test Cases for Jump Search code

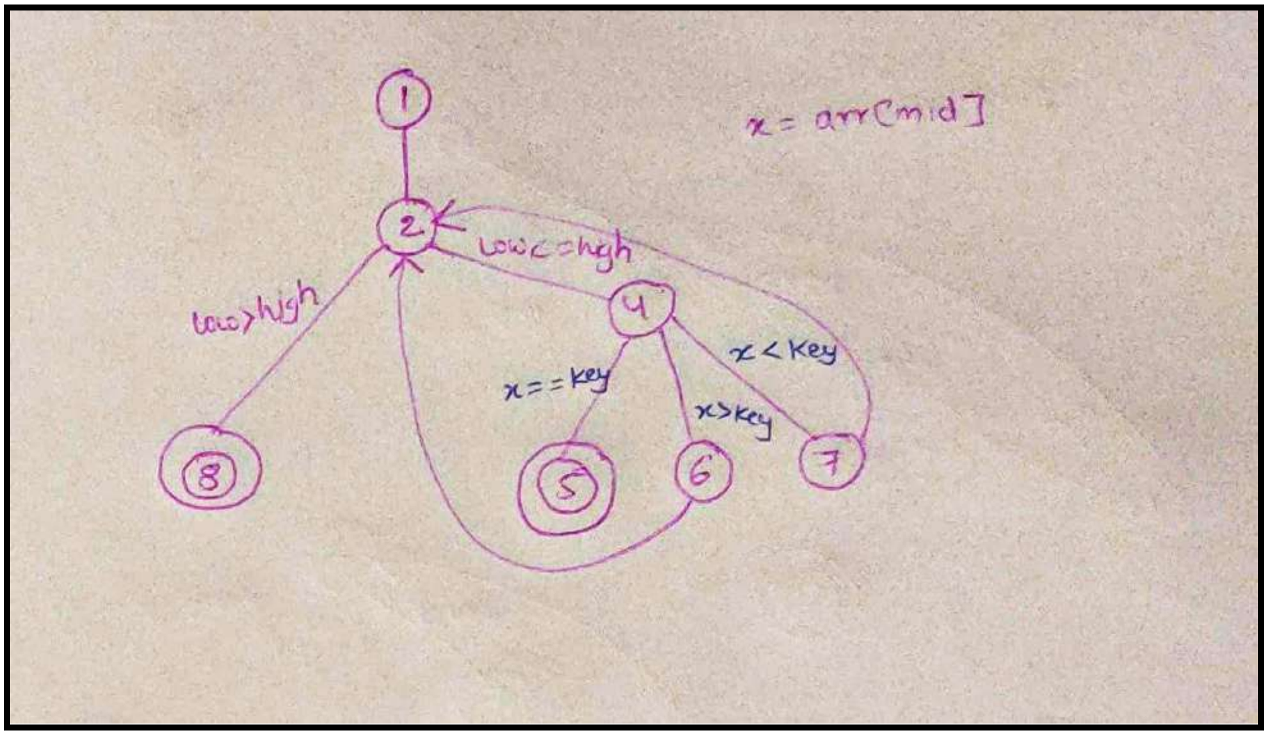


```
    //for edge 1-2-6-10-12
    // test paths 1-2-6-10-12
    ArrayList<Integer> exp1 = new ArrayList<~>(Arrays.asList(4, 5, 6, 7));
    int key1=-1;
    assertEquals( expected: -1,js.jumpSearch(exp1,key1));
    //for edge 1-2-6-10-11
    // test paths 1-2-6-10-12
    ArrayList<Integer> exp2 = new ArrayList<~>(Arrays.asList(-1, 5, 6, 7));
    int key2=-1;
    assertEquals( expected: 0,js.jumpSearch(exp2,key2));

    //for edge 1-2-3-5
    //test path 1-2-3-4-2-3-5
    ArrayList<Integer> exp3 = new ArrayList<~>(Arrays.asList(-1, 5, 6, 7));
    int key3=8;
    assertEquals( expected: -1,js.jumpSearch(exp3,key3));

   // for edge 1-2-6-7-9
    //test path 1-2-6-7-8-6-7-9
    ArrayList<Integer> exp4 = new ArrayList<~>(Arrays.asList(-1, 5, 6, 7));
    int key4=3;
    assertEquals( expected: -1,js.jumpSearch(exp4,key4));

    //for edge 1-2-6-10-11
    //test path 1-2-6-10-11
    ArrayList<Integer> exp5= new ArrayList<~>(Arrays.asList(5, 5, 5, 5,5));
    int key5=5;
    assertEquals( expected: 0,js.jumpSearch(exp5,key5));
```

These 5 test paths shown in the picture cover all edges and prime paths.

2. Binary Search

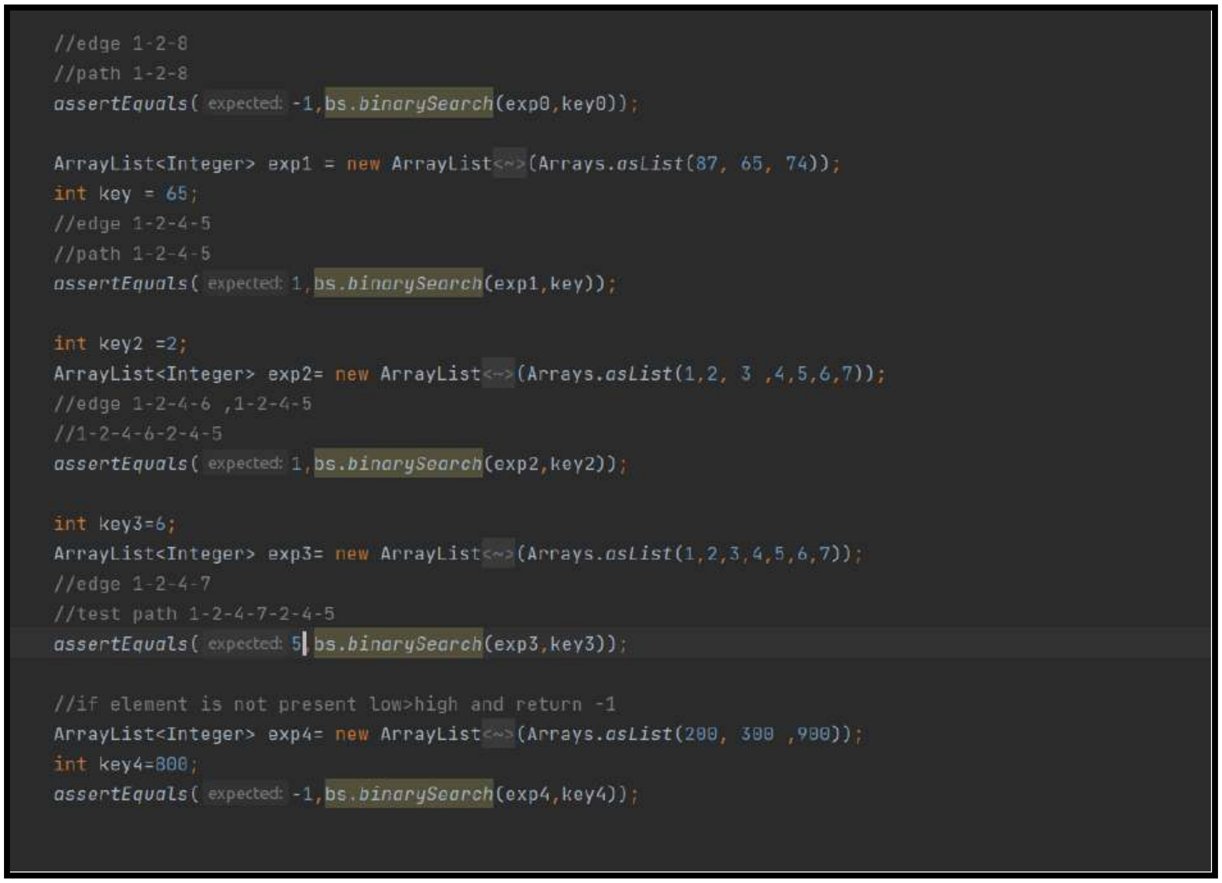
Control Flow Graph for Binary Search



BLOCK NUMBER VS LINE NUMBER OF CODE

Node	Line No.
1	7-8
2	9
4	11
5	12-16
6	17-19
7	20-21
8	26

Test Cases for Binary Search code



```
//edge 1-2-8
//path 1-2-8
assertEquals( expected: -1,bs.binarySearch(exp0,key0));

ArrayList<Integer> exp1 = new ArrayList<~>(Arrays.asList(87, 65, 74));
int key = 65;
//edge 1-2-4-5
//path 1-2-4-5
assertEquals( expected: 1,bs.binarySearch(exp1,key));

int key2 =2;
ArrayList<Integer> exp2= new ArrayList<~>(Arrays.asList(1,2, 3 ,4,5,6,7));
//edge 1-2-4-6 ,1-2-4-5
//1-2-4-6-2-4-5
assertEquals( expected: 1,bs.binarySearch(exp2,key2));

int key3=6;
ArrayList<Integer> exp3= new ArrayList<~>(Arrays.asList(1,2,3,4,5,6,7));
//edge 1-2-4-7
//test path 1-2-4-7-2-4-5
assertEquals( expected: 5,bs.binarySearch(exp3,key3));

//if element is not present low>high and return -1
ArrayList<Integer> exp4= new ArrayList<~>(Arrays.asList(200, 300 ,900));
int key4=800;
assertEquals( expected: -1,bs.binarySearch(exp4,key4));
```

These 4 test paths cover all edges and prime paths.

1) 1-2-8

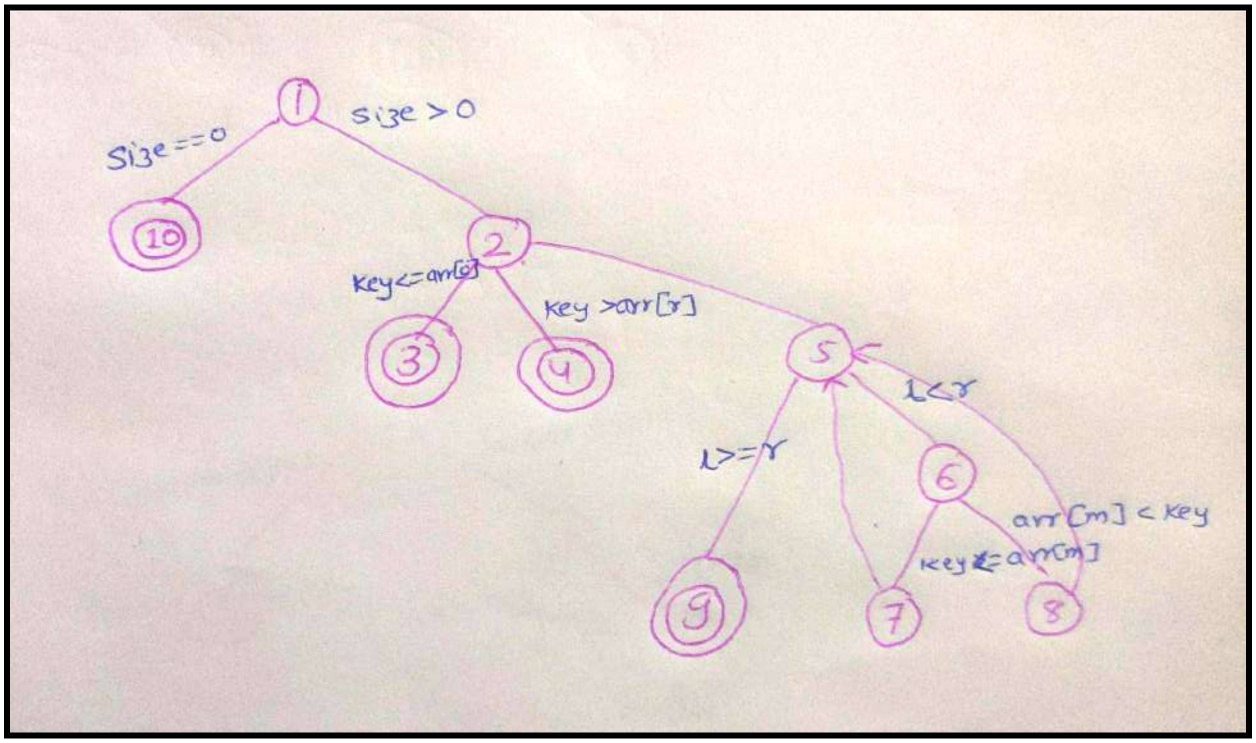
2) 1-2-4-5

3) 1-2-4-6-2-4-5

4) 1-2-4-7-2-4-5

3. Lower Bound of Element

Control Flow Graph for Lower Bound

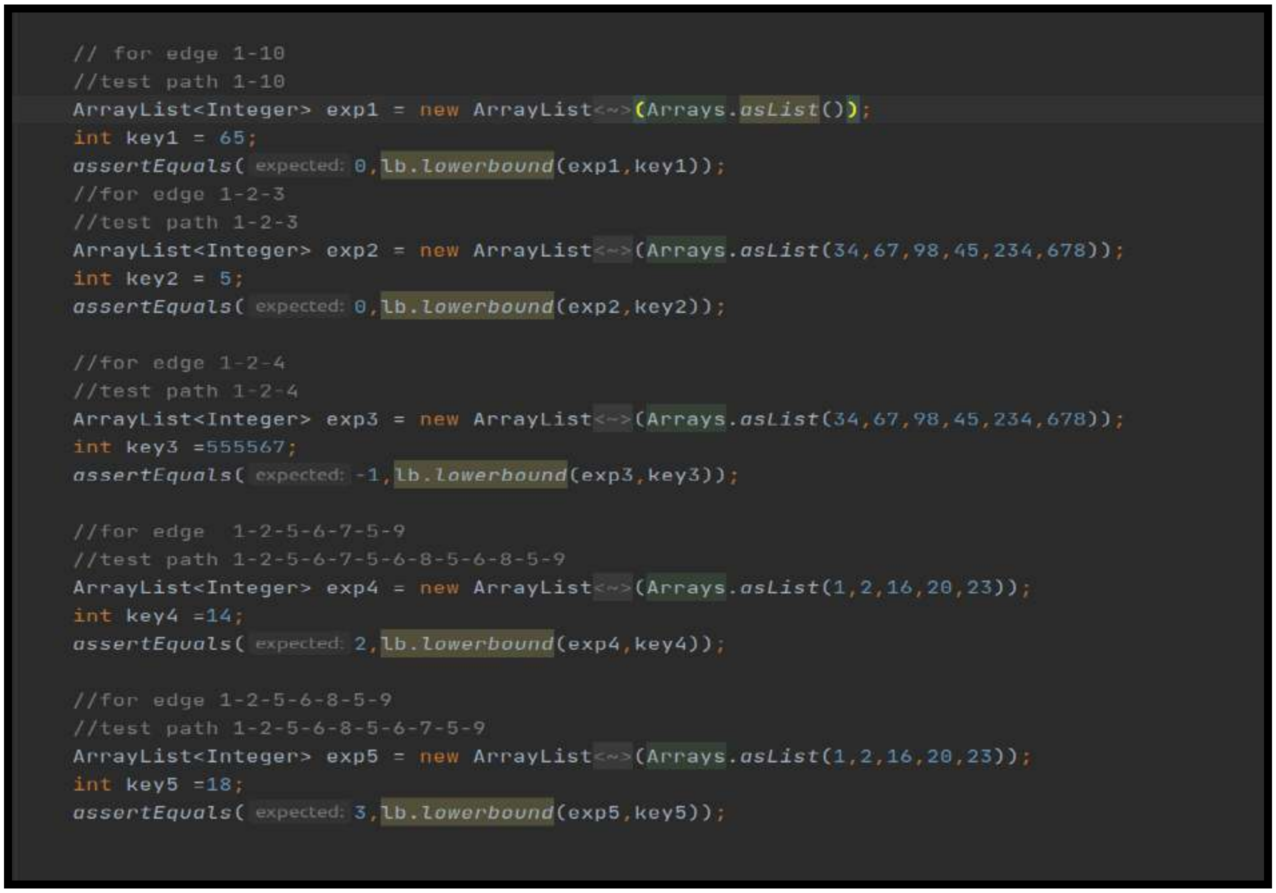


BLOCK NUMBER VS LINE NUMBER OF CODE

NODE	LINE NUMBER
1	9
2	12-13
3	14-15
4	17-18
5	20
6	21-22
7	23-24

NODE	LINE NUMBER
1	9
2	12-13
8	25-26
9	29
10	10

Test Cases for Lower Bound of Element



```
// for edge 1-10
//test path 1-10
ArrayList<Integer> exp1 = new ArrayList<~>(Arrays.asList());
int key1 = 65;
assertEquals( expected: 0,lb.lowerbound(exp1,key1));
//for edge 1-2-3
//test path 1-2-3
ArrayList<Integer> exp2 = new ArrayList<~>(Arrays.asList(34,67,98,45,234,678));
int key2 = 5;
assertEquals( expected: 0,lb.lowerbound(exp2,key2));

//for edge 1-2-4
//test path 1-2-4
ArrayList<Integer> exp3 = new ArrayList<~>(Arrays.asList(34,67,98,45,234,678));
int key3 =555567;
assertEquals( expected: -1,lb.lowerbound(exp3,key3));

//for edge  1-2-5-6-7-5-9
//test path 1-2-5-6-7-5-6-8-5-6-8-5-9
ArrayList<Integer> exp4 = new ArrayList<~>(Arrays.asList(1,2,16,20,23));
int key4 =14;
assertEquals( expected: 2,lb.lowerbound(exp4,key4));

//for edge 1-2-5-6-8-5-9
//test path 1-2-5-6-8-5-6-7-5-9
ArrayList<Integer> exp5 = new ArrayList<~>(Arrays.asList(1,2,16,20,23));
int key5 =18;
assertEquals( expected: 3,lb.lowerbound(exp5,key5));
```

These 5 test paths cover all edges and prime paths.

1) 1-10

2) 1-2-3

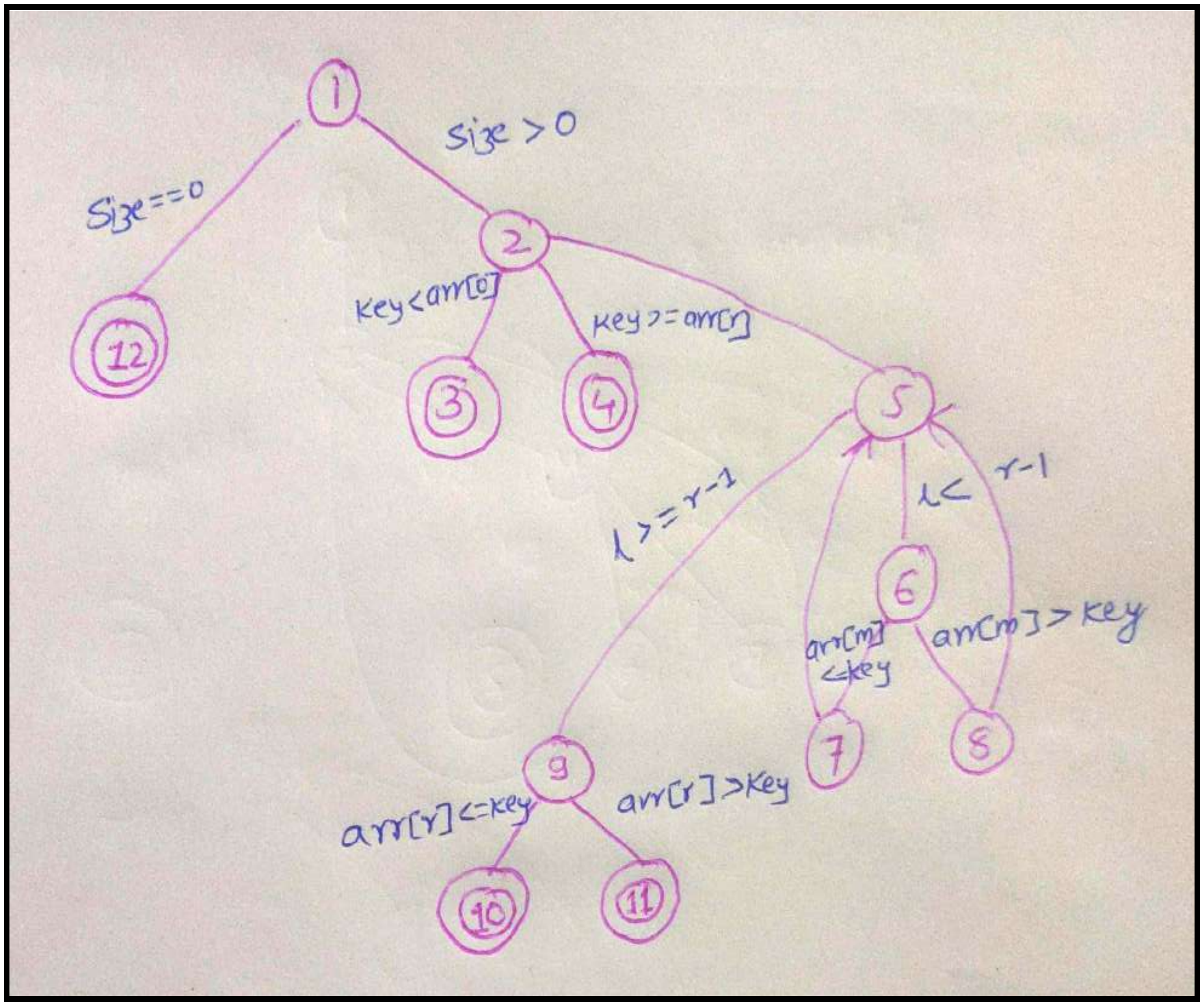
3) 1-2-4

4) 1-2-5-6-7-5-6-8-5-6-8-5-9

5) 1-2-5-6-8-7-5-9

4. Upper Bound of Element

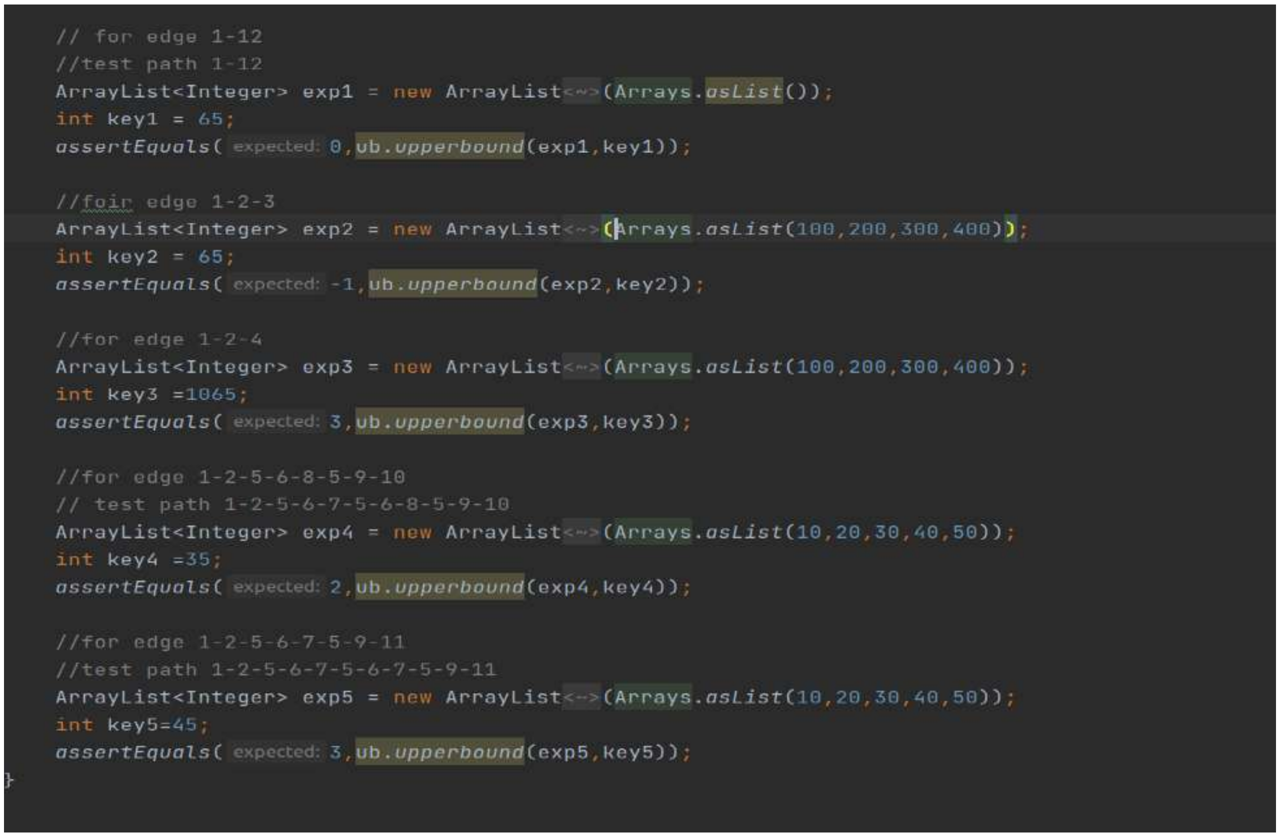
Control Flow Graph for Upper Bound



BLOCK NUMBER VS LINE NUMBER OF CODE

NODE	LINE NUMBER
1	9
2	12-13
3	14
4	17
5	20
6	21
7	22
8	25
9	28
10	29
11	31
12	10

Test Cases for Upper Bound of Element



```
    // for edge 1-12
    //test path 1-12
    ArrayList<Integer> exp1 = new ArrayList<~>(Arrays.asList());
    int key1 = 65;
    assertEquals( expected: 0,ub.upperbound(exp1,key1));

    //foir edge 1-2-3
    ArrayList<Integer> exp2 = new ArrayList<~>(Arrays.asList(100,200,300,400));
    int key2 = 65;
    assertEquals( expected: -1,ub.upperbound(exp2,key2));

    //for edge 1-2-4
    ArrayList<Integer> exp3 = new ArrayList<~>(Arrays.asList(100,200,300,400));
    int key3 =1065;
    assertEquals( expected: 3,ub.upperbound(exp3,key3));

    //for edge 1-2-5-6-8-5-9-10
    // test path 1-2-5-6-7-5-6-8-5-9-10
    ArrayList<Integer> exp4 = new ArrayList<~>(Arrays.asList(10,20,30,40,50));
    int key4 =35;
    assertEquals( expected: 2,ub.upperbound(exp4,key4));

    //for edge 1-2-5-6-7-5-9-11
    //test path 1-2-5-6-7-5-6-7-5-9-11
    ArrayList<Integer> exp5 = new ArrayList<~>(Arrays.asList(10,20,30,40,50));
    int key5=45;
    assertEquals( expected: 3,ub.upperbound(exp5,key5));
}
```

These 5 test paths cover all edges and prime paths.

1) 1-12

2) 1-2-3

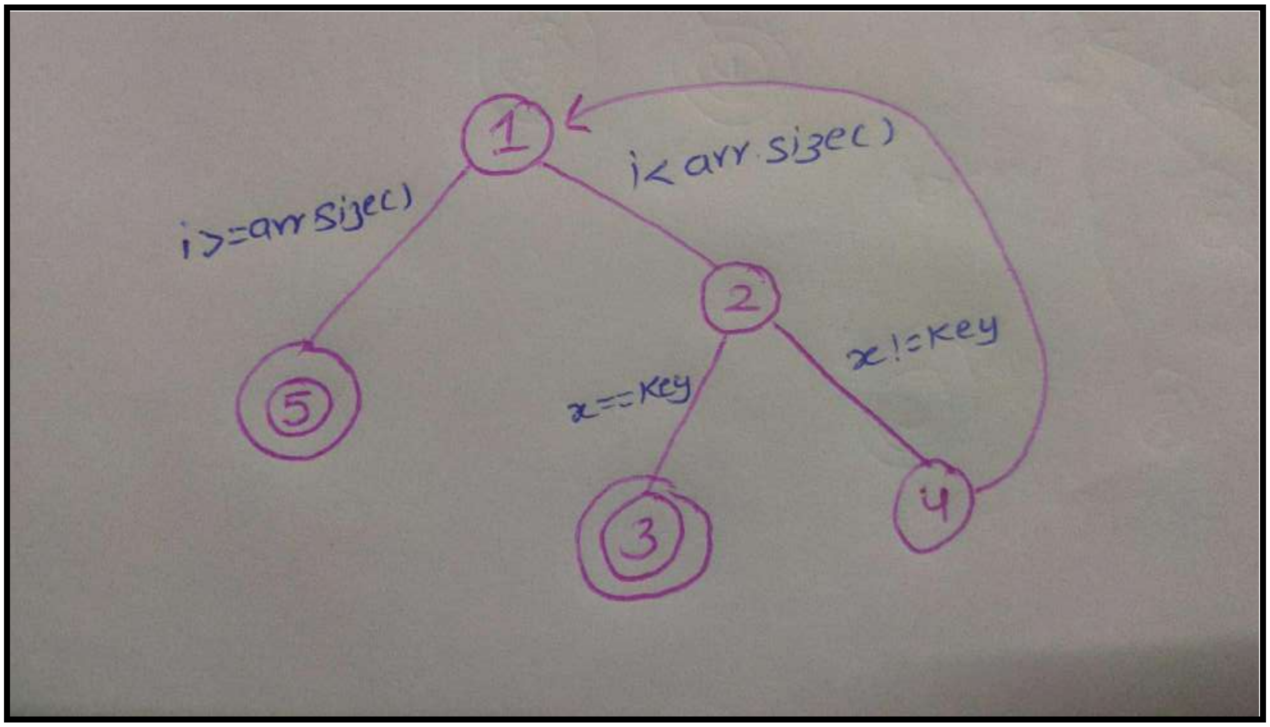
3) 1-2-4

4) 1-2-5-6-7-5-6-8-5-9-10

5) 1-2-5-6-7-5-6-7-5-9-11

5.Linear Search

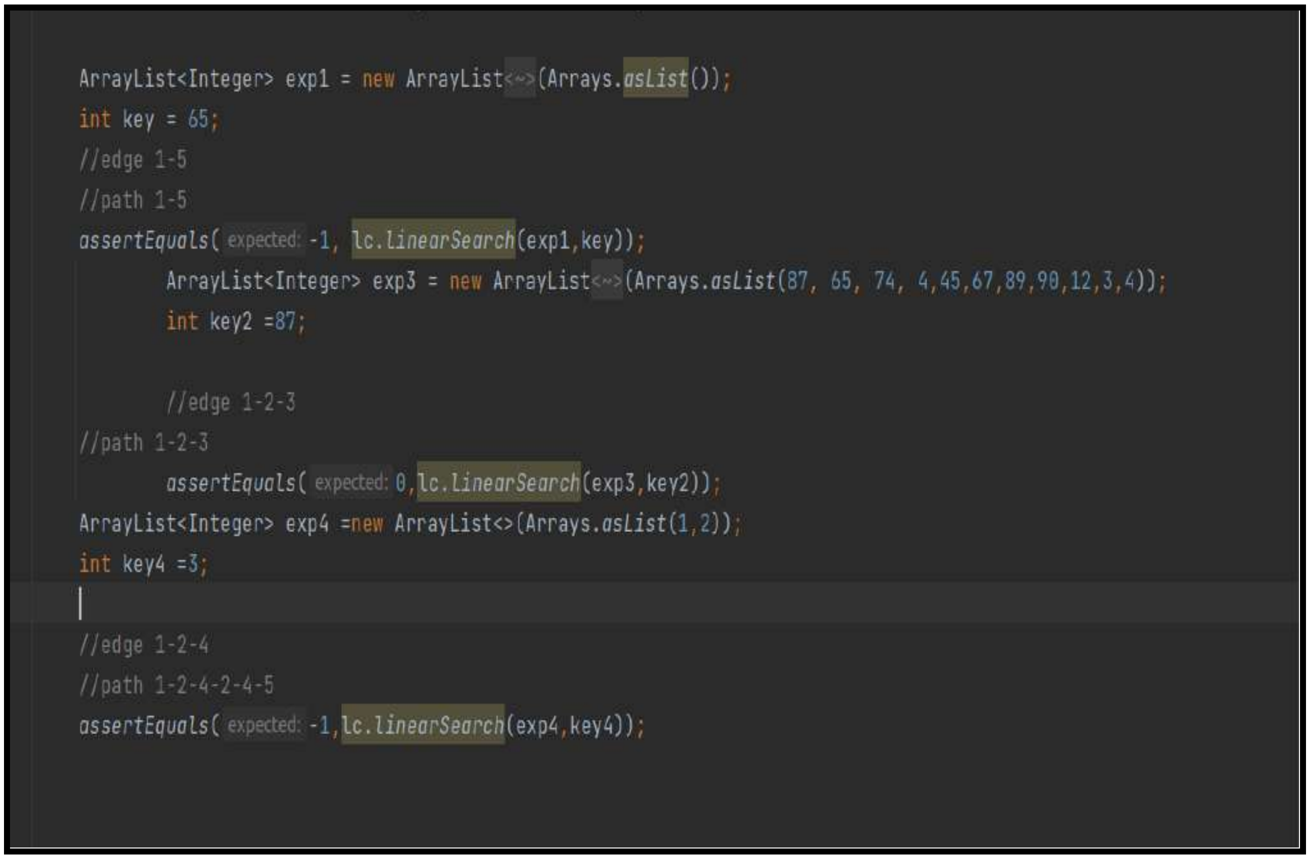
Control Flow Graph for Linear Search



BLOCK NUMBER VS LINE NUMBER OF CODE

NODE	LINE NUMBER
1	7
2	8
3	9-10
4	11
5	14

Test Cases for linear Search



```
ArrayList<Integer> exp1 = new ArrayList<~>(Arrays.asList());
int key = 65;
//edge 1-5
//path 1-5
assertEquals( expected: -1, lc.linearSearch(exp1,key));
        ArrayList<Integer> exp3 = new ArrayList<~>(Arrays.asList(87, 65, 74, 4,45,67,89,90,12,3,4));
        int key2 =87;

        //edge 1-2-3
//path 1-2-3
        assertEquals( expected: 0,lc.LinearSearch(exp3,key2));
ArrayList<Integer> exp4 =new ArrayList<>(Arrays.asList(1,2));
int key4 =3;

//edge 1-2-4
//path 1-2-4-2-4-5
assertEquals( expected: -1,lc.linearSearch(exp4,key4));
```

These 3 test paths cover all edges and prime paths.

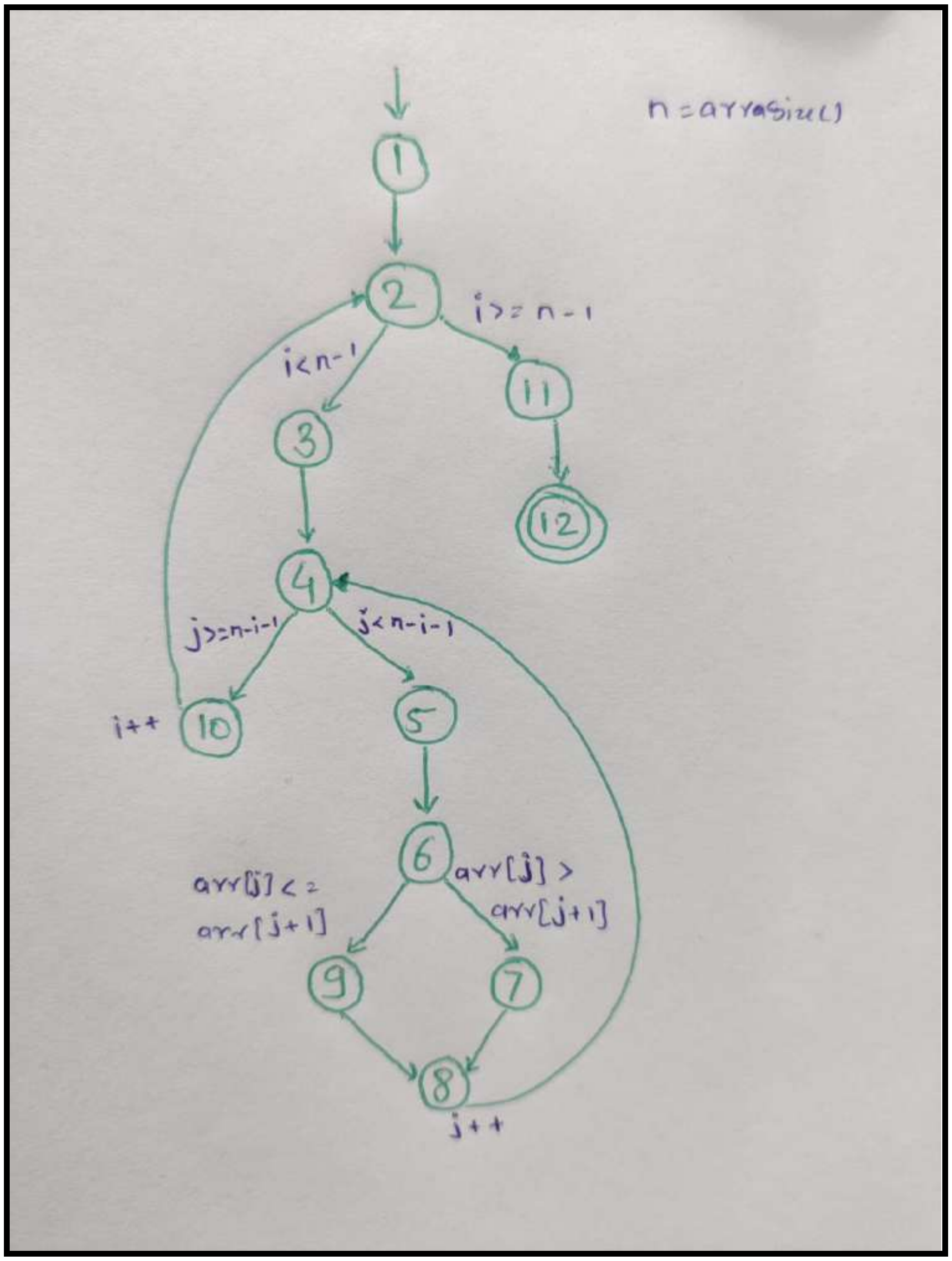
1) 1-5

2) 1-2-3

3) 1-2-4-2-4-5

6. Bubble Sort

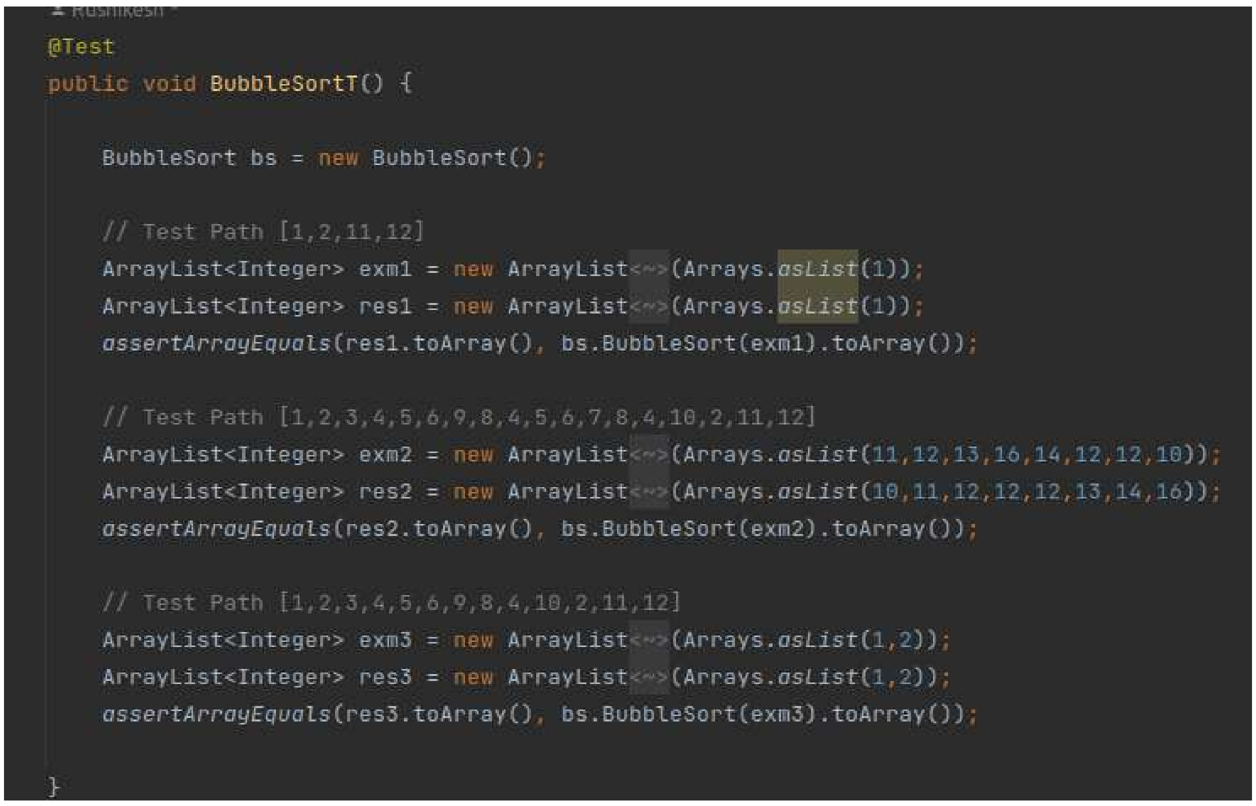
Control Flow Graph for Bubble Sort



BLOCK NUMBER VS LINE NUMBER OF CODE

NODE	LINE NUMBER
1	1-9
2	10
3	11
4	12
5	13
6	14
7	15-19
8	21
9	20
10	22
11	23-29
12	30

Test Cases for Bubble Sort



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

    BubbleSort bs = new BubbleSort();

    // Test Path [1,2,11,12]
    ArrayList<Integer> exm1 = new ArrayList<~>(Arrays.asList(1));
    ArrayList<Integer> res1 = new ArrayList<~>(Arrays.asList(1));
    assertArrayEquals(res1.toArray(), bs.BubbleSort(exm1).toArray());

    // Test Path [1,2,3,4,5,6,9,8,4,5,6,7,8,4,10,2,11,12]
    ArrayList<Integer> exm2 = new ArrayList<~>(Arrays.asList(11,12,13,16,14,12,12,10));
    ArrayList<Integer> res2 = new ArrayList<~>(Arrays.asList(10,11,12,12,12,13,14,16));
    assertArrayEquals(res2.toArray(), bs.BubbleSort(exm2).toArray());

    // Test Path [1,2,3,4,5,6,9,8,4,10,2,11,12]
    ArrayList<Integer> exm3 = new ArrayList<~>(Arrays.asList(1,2));
    ArrayList<Integer> res3 = new ArrayList<~>(Arrays.asList(1,2));
    assertArrayEquals(res3.toArray(), bs.BubbleSort(exm3).toArray());

}
```

These 3 test paths cover all edges and prime paths.

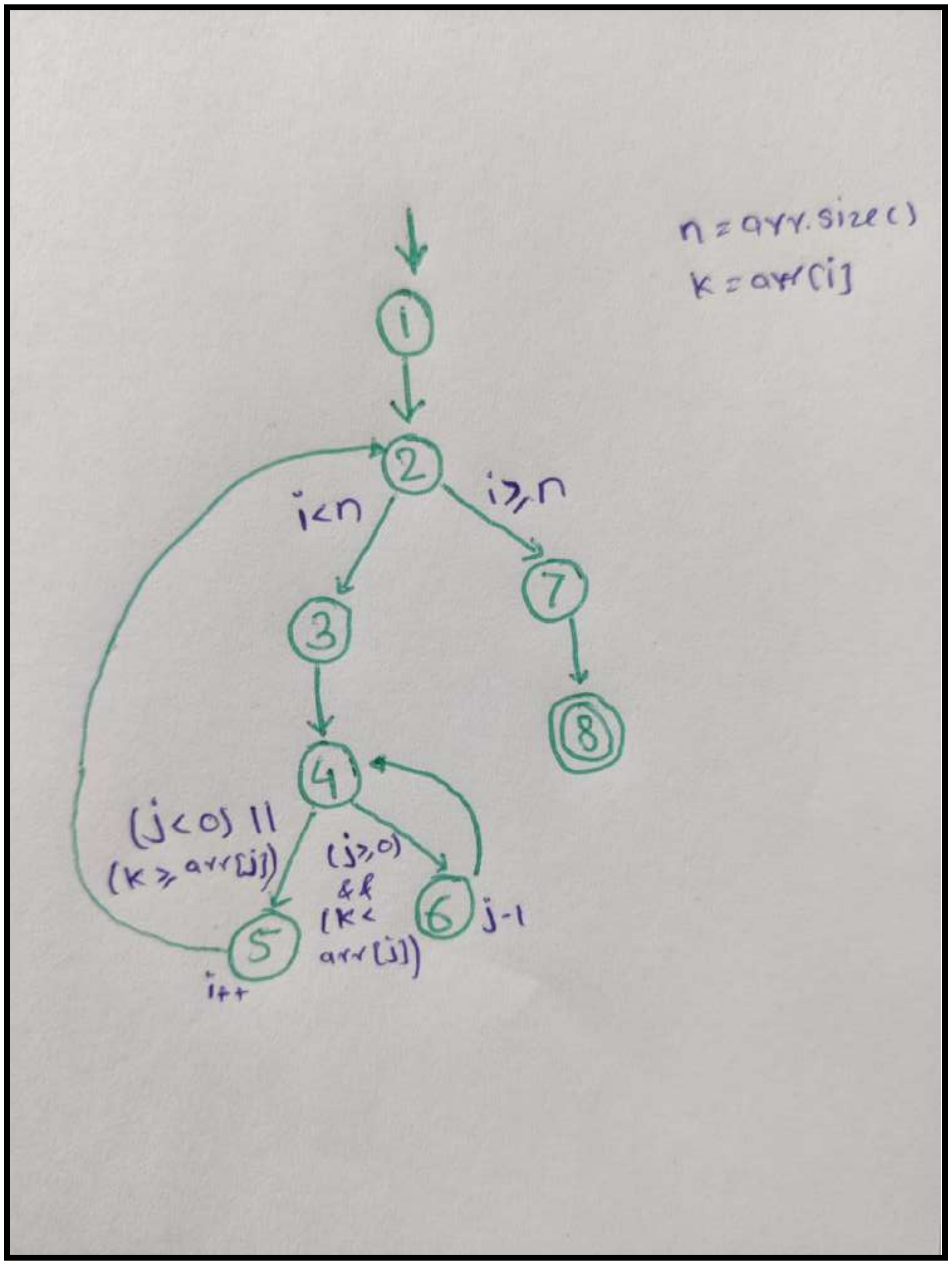
1) 1-2-11-12

2) 1-2-3-4-5-6-9-8-4-5-6-7-8-4-10-2-11-12

3) 1-2-3-4-5-6-9-8-4-10-2-11-12

7. Insertion Sort

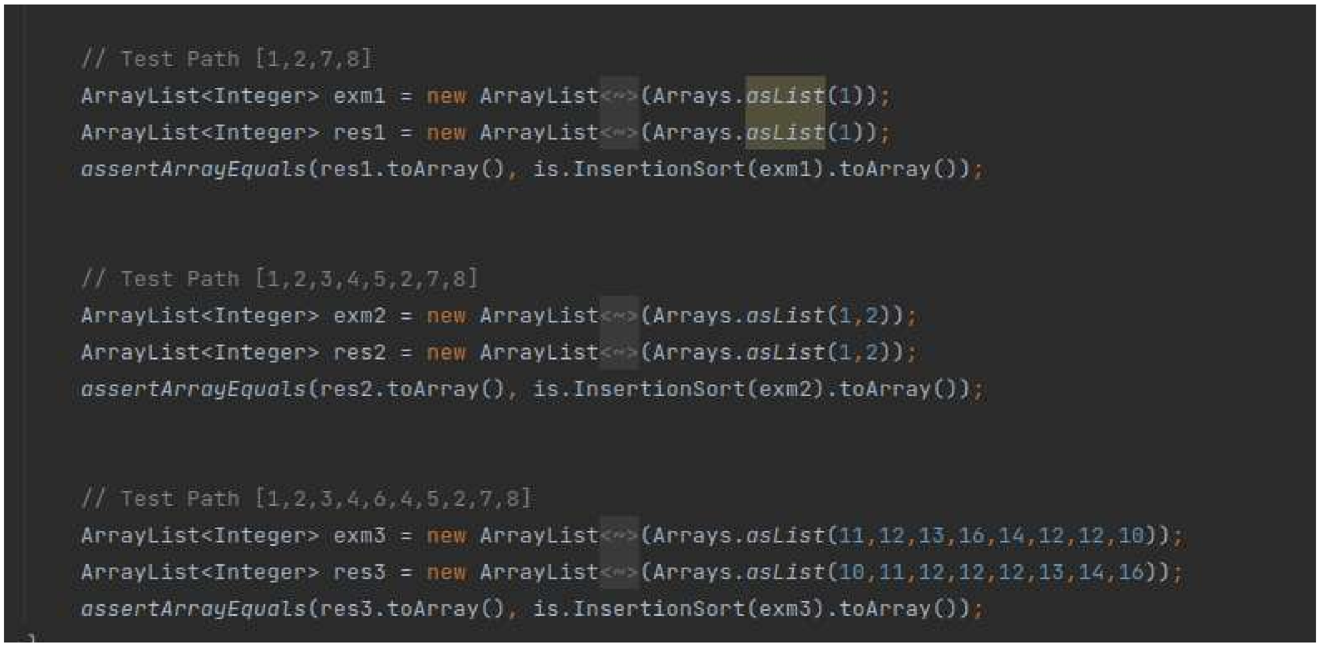
Control Flow Graph for Insertion Sort



BLOCK NUMBER VS LINE NUMBER OF CODE

NODE	LINE NUMBER
1	1-8
2	9
3	10-14
4	15
5	19-20
6	16-18
7	21-27
8	28

Test Cases for Insertion Sort



```
// Test Path [1,2,7,8]
ArrayList<Integer> exm1 = new ArrayList<~>(Arrays.asList(1));
ArrayList<Integer> res1 = new ArrayList<~>(Arrays.asList(1));
assertArrayEquals(res1.toArray(), is.InsertionSort(exm1).toArray());


// Test Path [1,2,3,4,5,2,7,8]
ArrayList<Integer> exm2 = new ArrayList<~>(Arrays.asList(1,2));
ArrayList<Integer> res2 = new ArrayList<~>(Arrays.asList(1,2));
assertArrayEquals(res2.toArray(), is.InsertionSort(exm2).toArray());


// Test Path [1,2,3,4,6,4,5,2,7,8]
ArrayList<Integer> exm3 = new ArrayList<~>(Arrays.asList(11,12,13,16,14,12,12,10));
ArrayList<Integer> res3 = new ArrayList<~>(Arrays.asList(10,11,12,12,12,13,14,16));
assertArrayEquals(res3.toArray(), is.InsertionSort(exm3).toArray());
```

These 3 test paths cover all edges and prime paths.

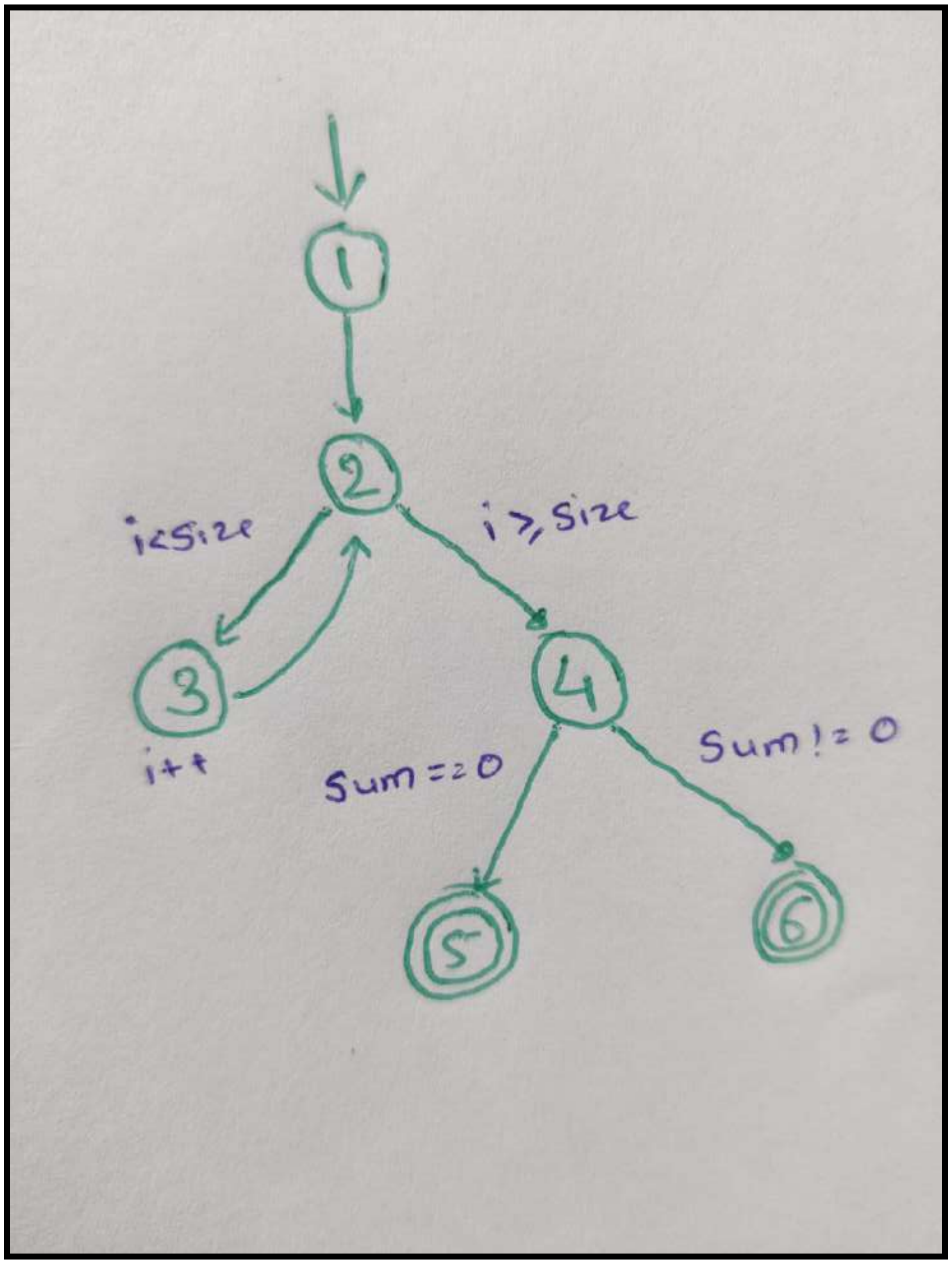
1) 1-2-7-8

2) 1-2-3-4-5-2-7-8

3) 1-2-3-4-6-4-5-2-7-8

8. Array Mean

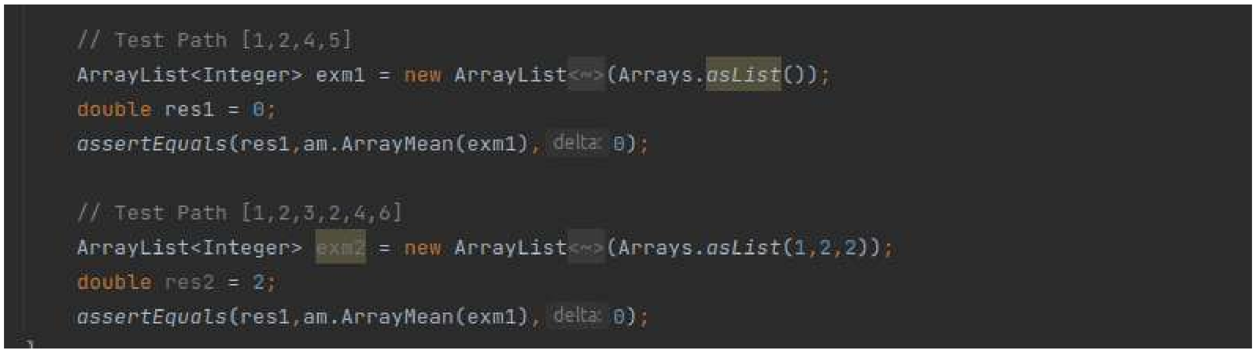
Control Flow Graph for Array Mean



BLOCK NUMBER VS LINE NUMBER OF CODE

NODE	LINE NUMBER
1	1-9
2	10
3	11-12
4	14
5	15
6	17-18

Test Cases for Array Mean



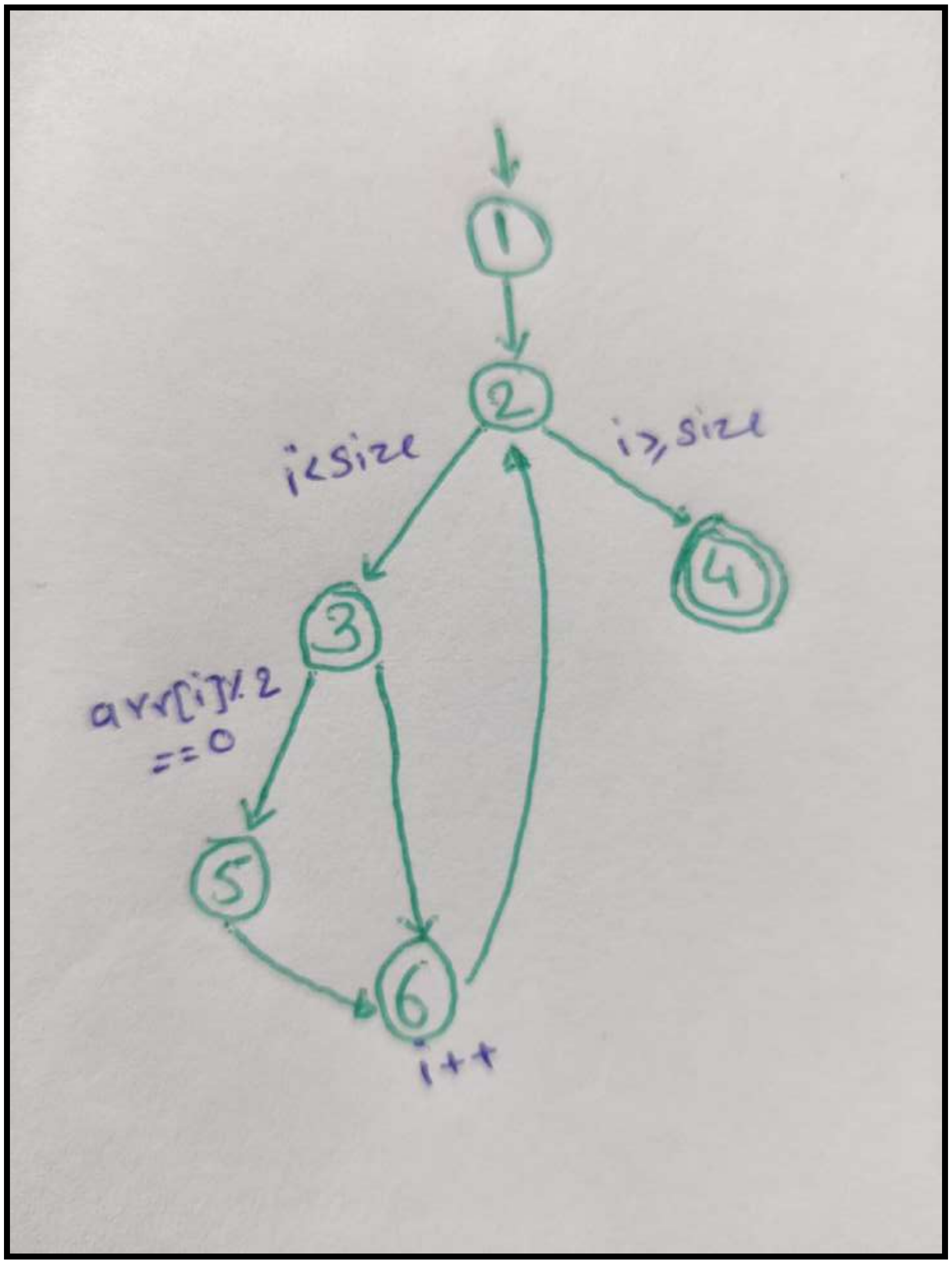
```
    // Test Path [1,2,4,5]
    ArrayList<Integer> exm1 = new ArrayList<~>(Arrays.asList());
    double res1 = 0;
    assertEquals(res1,am.ArrayMean(exm1), delta: 0);

    // Test Path [1,2,3,2,4,6]
    ArrayList<Integer> exm2 = new ArrayList<~>(Arrays.asList(1,2,2));
    double res2 = 2;
    assertEquals(res1,am.ArrayMean(exm1), delta: 0);
}
```

These 2 test paths cover all edges and prime paths.

1) 1-2-4-5
2) 1-2-3-2-4-6

8. Count of Even Digits in Array

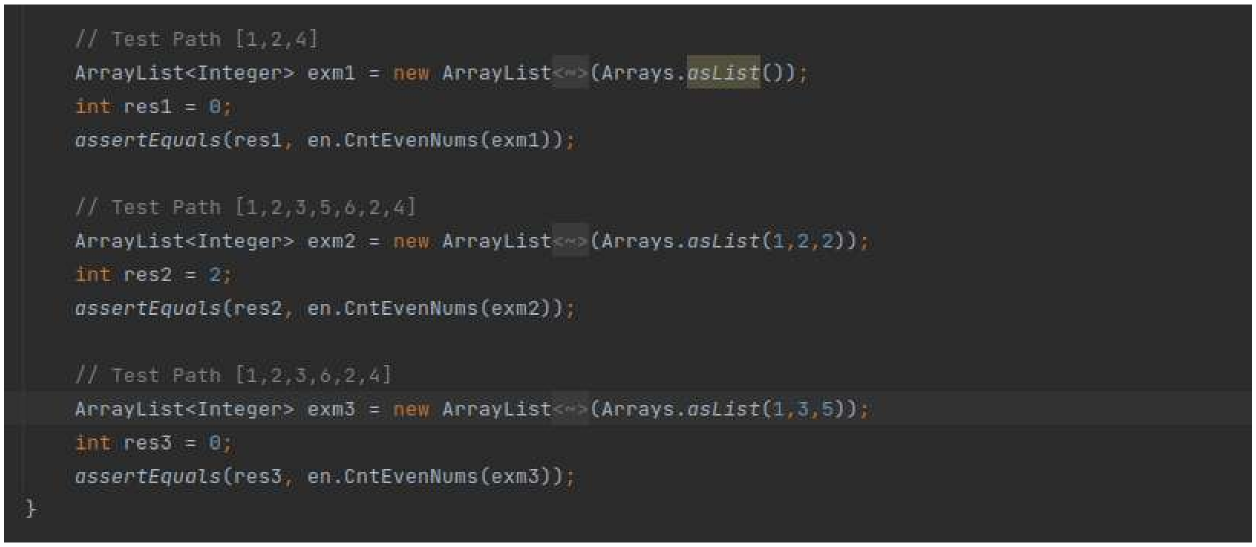
Control Flow Graph for Even Count



BLOCK NUMBER VS LINE NUMBER OF CODE

NODE	LINE NUMBER
1	1-11
2	12
3	14
4	18-21
5	15-16
6	17

Test Cases for Even Count



```
    // Test Path [1,2,4]
    ArrayList<Integer> exm1 = new ArrayList<~>(Arrays.asList());
    int res1 = 0;
    assertEquals(res1, en.CntEvenNums(exm1));

    // Test Path [1,2,3,5,6,2,4]
    ArrayList<Integer> exm2 = new ArrayList<~>(Arrays.asList(1,2,2));
    int res2 = 2;
    assertEquals(res2, en.CntEvenNums(exm2));

    // Test Path [1,2,3,6,2,4]
    ArrayList<Integer> exm3 = new ArrayList<~>(Arrays.asList(1,3,5));
    int res3 = 0;
    assertEquals(res3, en.CntEvenNums(exm3));
}
```

These 3 test paths cover all edges and prime paths.

1) 1-2-4
2) 1-2-3-5-6-2-4
3) 1-2-3-6-2-4

Team Members Contribution:

Kushal Agrawal [MT2021071]:

Implemented source code, created control flow graphs, and designed test cases for the following functions.

1) Jump Search
2) Binary Search
3) Linear Search
4) Upper Bound
5) Lower Bound

Rushikesh Kale [MT2021111]:

Implemented source code, created control flow graphs, and designed test cases for the following functions.

1) Bubble Sort
2) Insertion Sort
3) Selection Sort
4) Array Mean
5) Array Median
6) Array Sum
7) Count Even Digits in Array
8) Count Odd Digits in Array

Github Link:

https://github.com/ku685/SoftwareTestingProject