Ans.2)

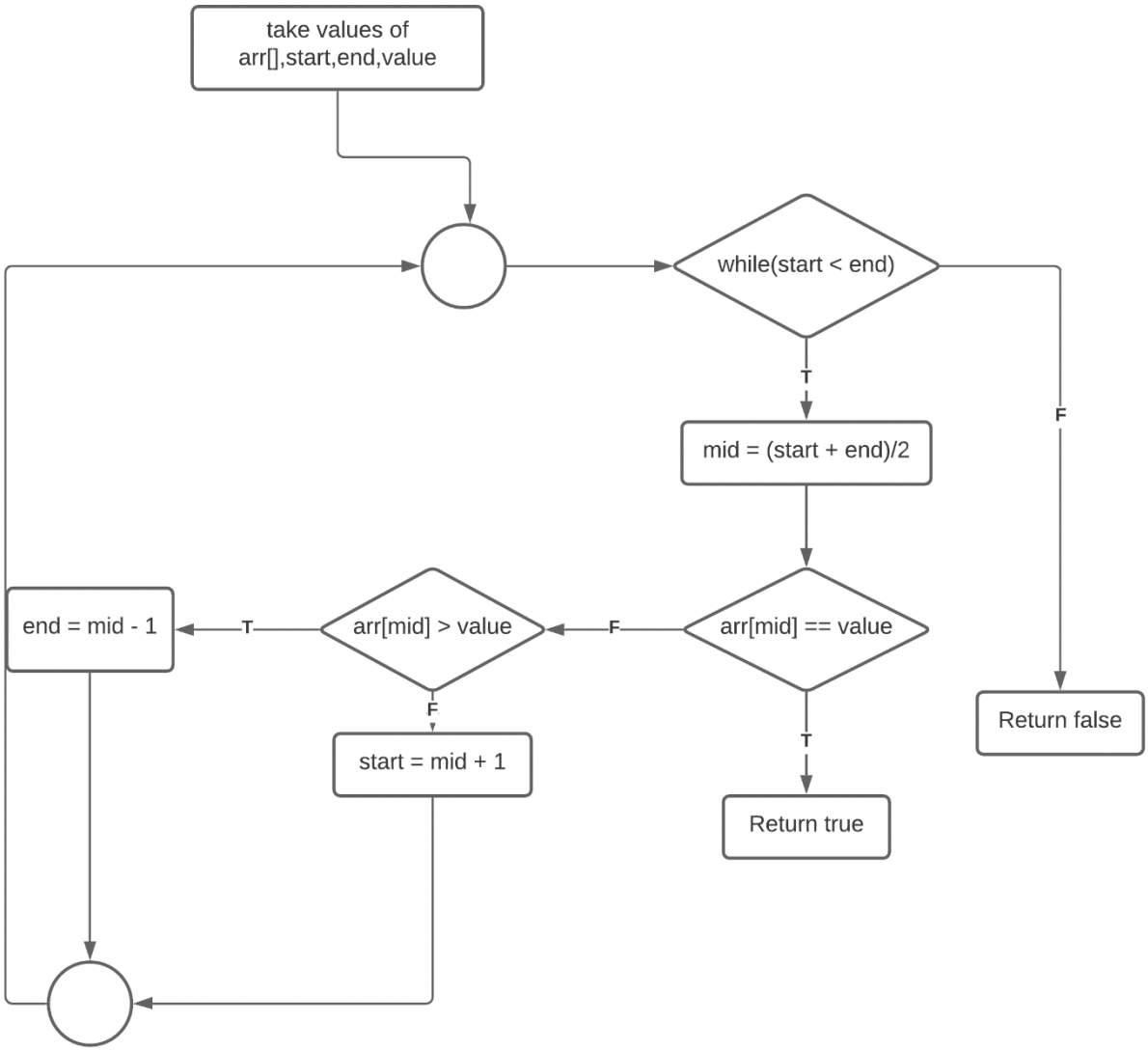
Binary Search Code:

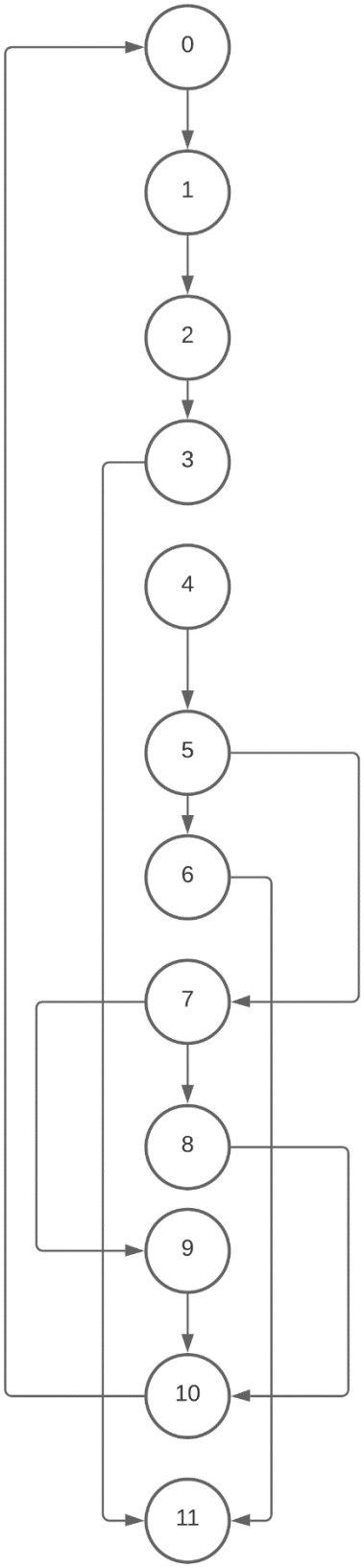
```
int binary_search(int array[], int startIndex, int endIndex,int element){
	if(startIndex<=endIndex){
		int midIndex  = startIndex+(endIndex-startIndex)/2;		//to avoid overflow
		if(array[midIndex]==element)
			return mid;
		else if(array[midIndex]<element)
			binary_search(array,midIndex+1,endIndex,element)
		else
			binary_search(array,startIndex,midIndex-1,element)
	}else{
		return -1; //Not Found
	}

}
```

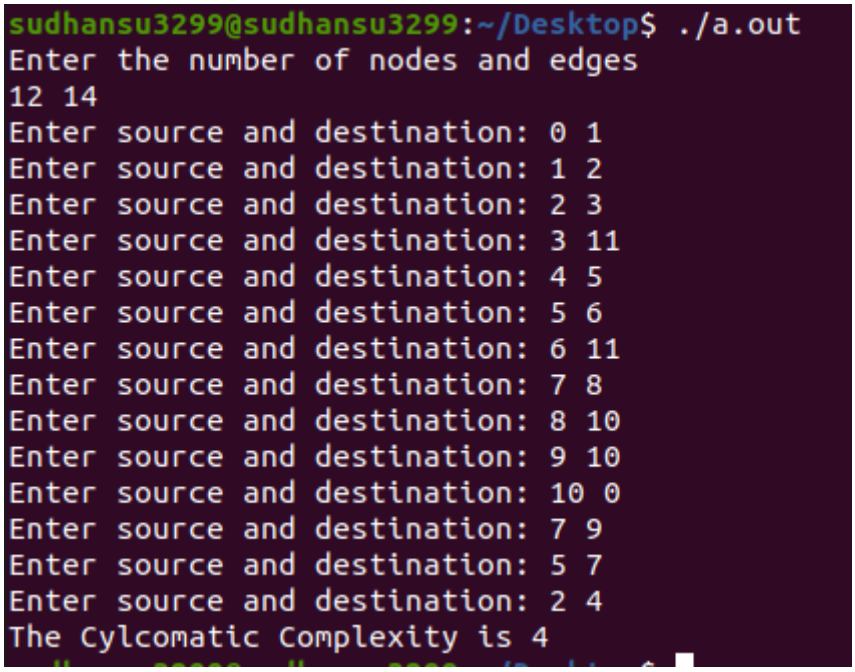
C++ Program to calculate the Cyclomatic Complexity:

```
#include<bits/stdc++.h>

using namespace std;

class Graph{
	public:
		int nodes,edges;
		map<int, vector<int> > adjList; //directed graph doesn't need 2-D vector
		int count_exit_nodes(){
			int count = 0;
			for(int i=0;i<nodes;i++)
				count += (adjList[i].size()==0);  //if the outdeg is 0
			return count;
		}

		void add_edge(int src, int dest){
			adjList[src].push_back(dest);
		}
		int calculate_cyclomatic_complexity(){
			return edges-nodes+(2*count_exit_nodes());
		}
};

int main(){
	int nodes, edges;
	Graph g;
	cout<<"Enter the number of nodes and edges"<<endl;
	cin>>nodes>>edges;
	g.nodes = nodes;
	g.edges = edges;

	for(int i=0;i<edges;i++){
```

```
            int src,dest;
            cout<<"Enter source and destination: ";
            cin>> src>> dest;
            g.add_edge(src,dest);
        }

        cout<< "The Cylcomatic Complexity is "<<g.calculate_cyclomatic_complexity()<<endl;
}
```




0
1
2
3
4
5
6
7
8
9
10
11



Ans.1)

```
#include<stdio.h>
#include<stdbool.h>

bool invalid(int side){
   return (side <= 0) || (side > 100);
}


int classify(int side1,int side2,int side3){

   if(invalid(side1) || invalid(side2) || invalid(side3))
      return -1;
   if((side1 + side2 <= side3 ) || ((side1 + side3 <= side2 ) || (side2 + side3 <= side1)))
      return 0; // Not a triangle
   if((side1 == side2) && (side1 == side3))
      return 1; // Equilateral Triangle
   if((side1 == side2) || ((side1 == side3) || (side2 == side3)))
      return 2; // Isosceles Triangle
   return 3;
}

char* triangleType(int type){
   switch(type){
      case 0:
         return "NOT A TRIANGLE";
      case 1:
         return "EQUILATERAL";
      case 2:
         return "ISOSCELES";
```

```
        case 3:
            return "SCALENE";
        default:
            return "NO VALID INPUT";
    }
}


void test(){
    printf("Starting Robustness Testing\n\n\n");
    int testSuite[19][3] = {
        {50,50,1},
        {50,50,2},
        {50,50,50},
        {50,50,100},
        {50,1,50},
        {50,2,50},
        {50,99,50},
        {50,100,50},
        {1,50,50},
        {2,50,50},
        {99,50,50},
        {50,50,50},
        {50,0,50},
        {0,50,50},
        {50,50,0},
        {50,101,50},
        {101,50,50},
        {50,50,101},
        {100,101,101}
    };

    for(int index=0;index < 19;index++){
        printf("Result for Sides %d %d %d is: ",testSuite[index][0],testSuite[index][1],testSuite[index]
[2]);
        printf("%s\n",triangleType(classify(testSuite[index][0],testSuite[index][1],testSuite[index]
[2])));
    }
}
int main(){
    test();
}
```