DSA HANDS-ON

array ds

```
import java.io.*;
import java.util.*;

public class Solution {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		int length = sc.nextInt();
		int[] array = new int[length];
		for (int i = 0; i < length ; i++) {
			array[i] = sc.nextInt();
		}
		for (int i = 0; i < length; i++){
			System.out.print(array[length-i-1] + " ");
		}
	}
}
```

2d array ds

```
using System;
using System.Collections.Generic;
using System.IO;
class Solution {
	static void Main(String[] args) {
		List<List<int>> matrix = new List<List<int>>();
		for (int i = 0; i < 6; ++i)
		{
			string[] elements = Console.ReadLine().Split(' ');
			matrix.Add(new List<int>());
			foreach (var item in elements)
			{
				matrix[i].Add(int.Parse(item));
			}
		}
		int max = -100500;
```

```
            int temp = 0;

            for (int i = 1; i <= 4; ++i)
            {
                  for (int j = 1; j <= 4; ++j)
                  {
                        temp =          matrix[i - 1][j + 1] +                          + matrix[i + 1][j
+ 1] +
                                        matrix[i - 1][j]          + matrix[i][j] + matrix[i + 1][j] +
                                        matrix[i - 1][j - 1] +                          + matrix[i + 1][j
- 1];
                        max = Math.Max(max, temp);
                  }
            }

            Console.WriteLine(max);
      }
}
```

LEFT ROTATION

```
using System;
using System.Collections.Generic;
using System.IO;
class Solution {
      static void Main(String[] args) {
            String[] settingsInput = Console.ReadLine().Split(' ');
            String[] arrayInput = Console.ReadLine().Split(' ');
            int arraySize = Int32.Parse(settingsInput[0]);
            int shiftCount = Int32.Parse(settingsInput[1]);
            int[] array = new int[arraySize];
            int[] shifted = new int[arraySize];

            for(int i = 0; i < arraySize; i++)
            {
                array[i] = Int32.Parse(arrayInput[i]);
            }


            //[1, 2, 3, 4, 5]

            //array[4] = 1,
            //array[3] = 2,
```

```
            //array[2] = 3,
            //array[1] = 4
            //array[0] = 5,
            //5, 4, 3, 2, 1
            int ctr = 0;
            for(int i = shiftCount; i < arraySize; i++)
            {
                  shifted[ctr++] = array[i];
            }

            for(int i = 0; i < shiftCount; i++)
            {
                  shifted[ctr++] = array[i];
            }

            //array[array.Length - 1] = first;

            foreach(int i in shifted)
            {
                  Console.Write(i + " ");
            }
      }
}
```

sparse array

```
using System;
using System.Collections.Generic;
using System.IO;
class Solution {
      static void Main(String[] args) {
            /* Enter your code here. Read input from STDIN. Print output to STDOUT.
Your class should be named Solution */
            int noOfLines = Convert.ToInt32(Console.ReadLine());
            String[] Lines = new String[noOfLines];
            int line=0;
            while(noOfLines>0)
                  {
                  Lines[line++]=Console.ReadLine();
                  noOfLines--;
            }
```

```
            int tests = Convert.ToInt32(Console.ReadLine());
            while(tests>0)
            {
                  int count=0;
                  String test = Console.ReadLine();
                  for(int i=0;i<Lines.Length;i++)
                              if(Lines[i].Equals(test))
                                    count++;
                  Console.WriteLine(count);
                  tests--;
            }
      }
}
```

array manipulation

```
using System;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;
using System.Globalization;
using System.Collections.Generic;
using System.Threading;
using kp.Algo;

namespace CodeForces
{
      internal class Solution
      {
            const int StackSize = 20 * 1024 * 1024;

            private void Solve()
            {
                  int n = NextInt(), m = NextInt();
                  var events = new List<Tuple<int, int, int>>();
                  while ( m-- > 0 )
                  {
                        int a = NextInt(), b = NextInt(), k = NextInt();
                        events.Add( new Tuple<int, int, int>( a, 0, k ) );
                        events.Add( new Tuple<int, int, int>( b, 1, k ) );
                  }
```

```
        events.Sort();
        long cur = 0, ans = 0;
        foreach ( var e in events )
        {
            if ( e.Item2 == 0 )
                cur += e.Item3;
            else cur -= e.Item3;
            ans = Math.Max( ans, cur );
        }
        Out.WriteLine( ans );
    }

    #region Local wireup

    public int[] NextIntArray( int size )
    {
        var res = new int[size];
        for ( int i = 0; i < size; ++i ) res[i] = NextInt();
        return res;
    }

    public long[] NextLongArray( int size )
    {
        var res = new long[size];
        for ( int i = 0; i < size; ++i ) res[i] = NextLong();
        return res;
    }

    public double[] NextDoubleArray( int size )
    {
        var res = new double[size];
        for ( int i = 0; i < size; ++i ) res[i] = NextDouble();
        return res;
    }

    public int NextInt()
    {
        return _in.NextInt();
    }

    public long NextLong()
    {
        return _in.NextLong();
    }

    public string NextLine()
    {
```

```
            return _in.NextLine();
        }

        public double NextDouble()
        {
            return _in.NextDouble();
        }

        readonly Scanner _in = new Scanner();
        static readonly TextWriter Out = Console.Out;

        void Start()
        {
#if KP_HOME
            var timer = new Stopwatch();
            timer.Start();
#endif
            var t = new Thread( Solve, StackSize );
            t.Start();
            t.Join();
#if KP_HOME
            timer.Stop();
            Console.WriteLine( string.Format( CultureInfo.InvariantCulture, "Done
in {0} seconds.\nPress <Enter> to exit.", timer.ElapsedMilliseconds / 1000.0 ) );
            Console.ReadLine();
#endif
        }

        static void Main()
        {
            new Solution().Start();
        }

        class Scanner : IDisposable
        {
            #region Fields

            readonly TextReader _reader;
            readonly int _bufferSize;
            readonly bool _closeReader;
            readonly char[] _buffer;
            int _length, _pos;

            #endregion

            #region .ctors
```

```
        public Scanner( TextReader reader, int bufferSize, bool closeReader )
        {
            _reader = reader;
            _bufferSize = bufferSize;
            _closeReader = closeReader;
            _buffer = new char[_bufferSize];
            FillBuffer( false );
        }

        public Scanner( TextReader reader, bool closeReader ) : this( reader, 1
<< 16, closeReader ) { }

        public Scanner( string fileName ) : this( new StreamReader( fileName,
Encoding.Default ), true ) { }

#if !KP_HOME
        public Scanner() : this( Console.In, false ) { }
#else
        public Scanner() : this( "input.txt" ) { }
#endif

        #endregion

        #region IDisposable Members

        public void Dispose()
        {
            if ( _closeReader )
            {
                _reader.Close();
            }
        }

        #endregion

        #region Properties

        public bool Eof
        {
            get
            {
                if ( _pos < _length ) return false;
                FillBuffer( false );
                return _pos >= _length;
            }
        }
```

```
        #endregion

        #region Methods

        private char NextChar()
        {
            if ( _pos < _length ) return _buffer[_pos++];
            FillBuffer( true );
            return _buffer[_pos++];
        }

        private char PeekNextChar()
        {
            if ( _pos < _length ) return _buffer[_pos];
            FillBuffer( true );
            return _buffer[_pos];
        }

        private void FillBuffer( bool throwOnEof )
        {
            _length = _reader.Read( _buffer, 0, _bufferSize );
            if ( throwOnEof && Eof )
            {
                throw new IOException( "Can't read beyond the end of file" );
            }
            _pos = 0;
        }

        public int NextInt()
        {
            var neg = false;
            int res = 0;
            SkipWhitespaces();
            if ( !Eof && PeekNextChar() == '-' )
            {
                neg = true;
                _pos++;
            }
            while ( !Eof && !IsWhitespace( PeekNextChar() ) )
            {
                var c = NextChar();
                if ( c < '0' || c > '9' ) throw new ArgumentException( "Illegal
character" );
                res = 10 * res + c - '0';
            }
            return neg ? -res : res;
        }
```

```
        public long NextLong()
        {
            var neg = false;
            long res = 0;
            SkipWhitespaces();
            if ( !Eof && PeekNextChar() == '-' )
            {
                neg = true;
                _pos++;
            }
            while ( !Eof && !IsWhitespace( PeekNextChar() ) )
            {
                var c = NextChar();
                if ( c < '0' || c > '9' ) throw new ArgumentException( "Illegal
character" );
                res = 10 * res + c - '0';
            }
            return neg ? -res : res;
        }

        public string NextLine()
        {
            SkipUntilNextLine();
            if ( Eof ) return "";
            var builder = new StringBuilder();
            while ( !Eof && !IsEndOfLine( PeekNextChar() ) )
            {
                builder.Append( NextChar() );
            }
            return builder.ToString();
        }

        public double NextDouble()
        {
            SkipWhitespaces();
            var builder = new StringBuilder();
            while ( !Eof && !IsWhitespace( PeekNextChar() ) )
            {
                builder.Append( NextChar() );
            }
            return double.Parse( builder.ToString(),
CultureInfo.InvariantCulture );
        }

        private void SkipWhitespaces()
        {
```

```
                while ( !Eof && IsWhitespace( PeekNextChar() ) )
                {
                    ++_pos;
                }
            }

            private void SkipUntilNextLine()
            {
                while ( !Eof && IsEndOfLine( PeekNextChar() ) )
                {
                    ++_pos;
                }
            }

            private static bool IsWhitespace( char c )
            {
                return c == ' ' || c == '\t' || c == '\n' || c == '\r';
            }

            private static bool IsEndOfLine( char c )
            {
                return c == '\n' || c == '\r';
            }

            #endregion
        }

        #endregion
    }
}

namespace kp.Algo { }
```

MINI MAX

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
class Solution {
    static void Main(String[] args) {
        long[] A = Array.ConvertAll(Console.ReadLine().Split(' '), long.Parse);
```

```
            long max = A.Sum(), min = max;

            max -= A.Min();
            min -= A.Max();
            Console.WriteLine(min + " " + max);
        }
}
```

TIME CONVERSION

```
using System;
using System.Collections.Generic;
using System.IO;
class Solution {
        static void Main(String[] args) {
            /* Enter your code here. Read input from STDIN. Print output to STDOUT.
Your class should be named Solution */

Console.WriteLine(DateTime.Parse(Console.ReadLine()).ToString("HH:mm:ss"));
        }
}
```

BETWEEN TWO SETS

```
using System;
using System.Collections.Generic;
using System.Linq;
class Solution {
        static void Main(String[] args) {
            Console.ReadLine();
            var A = Array.ConvertAll(Console.ReadLine().Split(' '), int.Parse).ToList();
            var B = Array.ConvertAll(Console.ReadLine().Split(' '), int.Parse).ToList();

            int c = 0;

            for (int i = 1; i < 10000; i++) {
                if (A.Any(x => i % x != 0)) continue;
                if (B.Any(x => x % i != 0)) continue;
```

```
                c++;
            }
            Console.WriteLine(c);
        }
}
```

DIVISiBLE SUM PAIR

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;

public class Solver
{
    public void Solve()
    {
        int n = ReadInt();
        int m = ReadInt();
        var a = ReadIntArray();

        int ans = 0;
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                if ((a[i] + a[j]) % m == 0)
                    ans++;
        Write(ans);
    }

    #region Main

    protected static TextReader reader;
    protected static TextWriter writer;
    static void Main()
    {
#if DEBUG
        reader = new StreamReader("..\\..\\input.txt");
        //reader = new StreamReader(Console.OpenStandardInput());
        writer = Console.Out;
        //writer = new StreamWriter("..\\..\\output.txt");
#else
```

```
            reader = new StreamReader(Console.OpenStandardInput());
            writer = new StreamWriter(Console.OpenStandardOutput());
            //reader = new StreamReader("input.txt");
            //writer = new StreamWriter("output.txt");
#endif
            try
            {
                  //var thread = new Thread(new Solver().Solve, 1024 * 1024 * 128);
                  //thread.Start();
                  //thread.Join();
                  new Solver().Solve();
            }
            catch (Exception ex)
            {
                  Console.WriteLine(ex);
#if DEBUG
#else
                  throw;
#endif
            }
            reader.Close();
            writer.Close();
      }

      #endregion

      #region Read / Write
      private static Queue<string> currentLineTokens = new Queue<string>();
      private static string[] ReadAndSplitLine() { return reader.ReadLine().Split(new[]
{ ' ', '\t', }, StringSplitOptions.RemoveEmptyEntries); }
      public static string ReadToken() { while (currentLineTokens.Count ==
0)currentLineTokens = new Queue<string>(ReadAndSplitLine()); return
currentLineTokens.Dequeue(); }
      public static int ReadInt() { return int.Parse(ReadToken()); }
      public static long ReadLong() { return long.Parse(ReadToken()); }
      public static double ReadDouble() { return double.Parse(ReadToken(),
CultureInfo.InvariantCulture); }
      public static int[] ReadIntArray() { return
ReadAndSplitLine().Select(int.Parse).ToArray(); }
      public static long[] ReadLongArray() { return
ReadAndSplitLine().Select(long.Parse).ToArray(); }
      public static double[] ReadDoubleArray() { return ReadAndSplitLine().Select(s =>
double.Parse(s, CultureInfo.InvariantCulture)).ToArray(); }
      public static int[][] ReadIntMatrix(int numberOfRows) { int[][] matrix = new
int[numberOfRows][]; for (int i = 0; i < numberOfRows; i++)matrix[i] = ReadIntArray();
return matrix; }
      public static int[][] ReadAndTransposeIntMatrix(int numberOfRows)
```

```
    {
        int[][] matrix = ReadIntMatrix(numberOfRows); int[][] ret = new
int[matrix[0].Length][];
        for (int i = 0; i < ret.Length; i++) { ret[i] = new int[numberOfRows]; for (int j
= 0; j < numberOfRows; j++)ret[i][j] = matrix[j][i]; } return ret;
    }
    public static string[] ReadLines(int quantity) { string[] lines = new
string[quantity]; for (int i = 0; i < quantity; i++)lines[i] = reader.ReadLine().Trim();
return lines; }
    public static void WriteArray<T>(IEnumerable<T> array)
{ writer.WriteLine(string.Join(" ", array)); }
    public static void Write(params object[] array) { WriteArray(array); }
    public static void WriteLines<T>(IEnumerable<T> array) { foreach (var a in
array)writer.WriteLine(a); }
    private class SDictionary<TKey, TValue> : Dictionary<TKey, TValue>
    {
        public new TValue this[TKey key]
        {
            get { return ContainsKey(key) ? base[key] : default(TValue); }
            set { base[key] = value; }
        }
    }
    private static T[] Init<T>(int size) where T : new() { var ret = new T[size]; for (int i
= 0; i < size; i++)ret[i] = new T(); return ret; }
    #endregion
}
```

FORMING A MAGIC SQUARE

```
using System;
using System.Collections.Generic;

public class Solution
{
    public string[] _matrix;
    public List<string[]> _possibleSquares;

    public enum SeqType
    {
        Row,
        Col,
        Diag
```

```
    }

    public static void Main(string[] args)
    {
        Solution p = new Solution();
        p.Run(args);
    }

    public void Run(string[] args)
    {
        InitPossibleSquares();
        //List<string> possibleSequences = GetPossibleSequences();

        IInput input;

        if (args.Length > 0)
        {
            input = new StringInput(System.IO.File.ReadAllLines(args[0]));
        }
        else
        {
            input = new ConsoleInput();
        }

        _matrix = new string[3];
        CreateMatrixRow(0, input.ReadLine());
        CreateMatrixRow(1, input.ReadLine());
        CreateMatrixRow(2, input.ReadLine());

        int minSum = int.MaxValue;
        //int idx = 0;
        foreach(var square in _possibleSquares)
        {
            //Console.WriteLine($"    =====    {idx++}    ====");
            int totSum = 0;
            for(int i = 0; i < 3; i ++)
                for(int j = 0; j < 3; j ++)
                {
                    int sum = square[i][j] - _matrix[i][j];

                    if (sum < 0)
                        sum *= -1;

                    totSum += sum;

                    //Console.WriteLine($"{square[i][j]} -- {square[i][j]} :
{sum}");
```

```
                }

            //Console.WriteLine($" -- TotSum: {totSum}");
            if (minSum > totSum)
                minSum = totSum;
        }

        Console.WriteLine(minSum);
/*
        int seqCount = possibleSequences.Count;
        for (int i = 0; i < seqCount; i ++)
            for (int j = 0; j < seqCount; j ++)
                for (int k = 0; k < seqCount; k ++)
                {
                    if (i == j || i == k || j == k)
                        continue;

                    _matrix[0] = possibleSequences[i];
                    _matrix[1] = possibleSequences[j];
                    _matrix[2] = possibleSequences[k];

                    if (IsMagic())
                    {
                        Console.WriteLine(_matrix[0]);
                        Console.WriteLine(_matrix[1]);
                        Console.WriteLine(_matrix[2]);
                        Console.WriteLine();
                    }
                }
*/
    }

    public void InitPossibleSquares()
    {
        _possibleSquares = new List<string[]>
        {
            new []
            {
            "276",
            "951",
            "438",
            },

            new []
            {
            "294",
            "753",
```

```
            "618",
            },

            new []
            {
            "438",
            "951",
            "276",
            },

            new []
            {
            "492",
            "357",
            "816",
            },

            new []
            {
            "618",
            "753",
            "294",
            },

            new []
            {
            "672",
            "159",
            "834",
            },

            new []
            {
            "816",
            "357",
            "492",
            },

            new []
            {
            "834",
            "159",
            "672",
            }
        };
}
```

```
public bool IsMagic()
{
	int sum = Sum("195");

	for (int i = 0; i < 3; i++)
	{
		if (Sum(GetSequence(SeqType.Row, i)) != sum)
			return false;

		if (Sum(GetSequence(SeqType.Col, i)) != sum)
			return false;

		if (i < 2 && Sum(GetSequence(SeqType.Diag, i)) != sum)
			return false;
	}

	bool[] num = new bool[9];

	for (int i = 0; i < 3; i++)
		for (int j = 0; j < 3; j++)
		{
			int idx = _matrix[i][j] - '0' - 1;
			if (num[idx])
				return false;
			num[idx] = true;
		}
	return true;
}

public int Sum(string seq)
{
	return (int)seq[0] + (int)seq[1] + (int)seq[2];
}

public void CreateMatrixRow(int row, string input)
{
	string rowStr = $"{input[0]}{input[2]}{input[4]}";
	_matrix[row] = rowStr;
}

public string GetSequence(SeqType seq, int n)
{
	switch (seq)
	{
		case SeqType.Row:
			return _matrix[n];
```

```
            case SeqType.Col:
                return $"{_matrix[0][n]}{_matrix[1][n]}{_matrix[2][n]}";

            default:
                if (n == 0)
                    return $"{_matrix[0][0]}{_matrix[1][1]}{_matrix[2][2]}";
                else
                    return $"{_matrix[0][2]}{_matrix[1][1]}{_matrix[2][0]}";
        }
    }

    public List<string> GetPossibleSequences()
    {
        List<string> p = new List<string>();

        for (int i = 1; i < 10; i++)
            for (int j = 1; j < 10; j++)
                for (int k = 1; k < 10; k++)
                {
                    if (i == j || i == k || j == k)
                        continue;

                    if (i + j + k != 15)
                        continue;

                    p.Add($"{i}{j}{k}");
                }

        return p;
    }

    public interface IInput
    {
        string ReadLine();
    }

    public class ConsoleInput : IInput
    {
        public string ReadLine()
        {
            return Console.ReadLine();
        }
    }

    public class StringInput : IInput
    {
        IEnumerable<string> _data;
```

```
        IEnumerator<string> _dataEnumerator;

        public StringInput (IEnumerable<string> data)
        {
            _data = data;
            _dataEnumerator = _data.GetEnumerator();
        }

        public string ReadLine()
        {
            _dataEnumerator.MoveNext();
            var retVal = _dataEnumerator.Current;
            return retVal;
        }
    }
}
```

QUEUE USING TWO STACK

```
using System;
using System.Text;
using System.Collections.Generic;
using System.IO;
class Solution {
    static void Main(String[] args) {
        var c = int.Parse(Console.ReadLine());
        var stack = new MyStack();
        var sb = new StringBuilder();
        for (var i = 0; i < c; i++)
        {
            var a = Array.ConvertAll(Console.ReadLine().Split(' '), int.Parse);
            switch (a[0])
            {
                case 1:
                stack.Enqueue(a[1]);
                break;
                case 2:
                stack.Dequeue();
                break;
                case 3:
                stack.PrintFront(sb);
                break;
```

```
            }
        }

        Console.WriteLine(sb.ToString());
    }

    class MyStack
    {
        Stack<int> main = new Stack<int>();
        Stack<int> slave = new Stack<int>();
        int front;

        public void Enqueue(int x)
        {
            main.Push(x);

            if (main.Count == 1 && slave.Count == 0) front = x;
        }

        public void Dequeue()
        {
            if (slave.Count == 0)
            {
                while (main.Count != 0)
                {
                    slave.Push(main.Pop());
                }
            }

            slave.Pop();

            if (slave.Count > 0)
                front = slave.Peek();
            else
            {
                if (main.Count != 0)
                {
                    while (main.Count != 0)
                    {
                        slave.Push(main.Pop());
                    }

                    front = slave.Peek();
                }
            }
        }
```

```
        public void PrintFront(StringBuilder sb)
        {
            sb.AppendLine(front.ToString());
        }
    }

}
```

BALANCE BRACKETS

```
using System;
using System.Collections.Generic;
using System.IO;
class Solution {
    static void Main(String[] args) {
        int n = Int32.Parse(Console.ReadLine());
        for (int idx = 0; idx < n; idx++) {
            string input = Console.ReadLine();

            Stack<char> stack = new Stack<char>();
            bool fault = false;
            foreach (char s in input) {
                if (s == '{' || s == '[' || s == '(') {
                    stack.Push(s);
                    continue;
                }

                if (stack.Count == 0) {
                    fault = true;
                    break;
                }

                char v = stack.Peek();
                if ((v != '{' && s == '}') ||
                    (v != '[' && s == ']') ||
                    (v != '(' && s == ')')) {
                    fault = true;
                    break;
                }

                stack.Pop();
            }
```

```
                if (stack.Count == 0 && !fault) {
                        Console.WriteLine("YES");
                }
                else {
                        Console.WriteLine("NO");
                }
        }
    }
}
```

Component in graph

```
using System;
using System.Collections.Generic;
using System.IO;
class Solution {

    private sealed class UnionFind {
        private readonly int[] store;
        private readonly int[] size;

        public UnionFind(int count) {
            store = new int[count];
            size = new int[count];
            for(int i = 0; i < count; i++) {
                store[i] = i;
                size[i] = 1;
            }
        }

        public void union(int a, int b) {
            var componentA = find(a);
            var componentB = find(b);
            if (componentA == componentB) return;
            if (size[componentB] > size[componentA]) {
                store[componentA] = componentB;
                size[componentB] += size[componentA];
            } else {
                store[componentB] = componentA;
                size[componentA] += size[componentB];
            }
        }
```

```
        public int find(int v) {
            int parent = v;
            while(store[parent] != parent) {
                parent = store[parent];
            }
            return parent;
        }

        public void getMinAndMaxComponentSizes(out int min, out int max) {
            min = size.Length;
            max = 0;

            for(int c = 1; c < size.Length; c++) {
                //Console.Write(store[c] + ":" + size[c] + " ");
                if (store[c] != c || size[c] == 1) continue; // not component
                if (size[c] < min) min = size[c];
                if (size[c] > max) max = size[c];
            }
            //Console.WriteLine();
        }

    }

    static void Main(String[] args) {
        int n = int.Parse(Console.ReadLine());
        var uf = new UnionFind(2*n + 1);

        for(int i = 0; i < n; i++) {
            var verticies = Console.ReadLine().Split(new [] {' '});
            var a = int.Parse(verticies[0]);
            var b = int.Parse(verticies[1]);
            uf.union(a, b);
        }
        int min = 1;
        int max = 0;
        uf.getMinAndMaxComponentSizes(out min, out max);
        Console.WriteLine(min + " " + max);
    }
}
```

FIND THE RUNNING MEDIAN

```
using System;
using System.Collections.Generic;
using System.IO;
class Solution {
        static void Main(String[] args)
        {
                List<int> slist = new     List<int>();
                int n = int.Parse(Console.ReadLine());

                for(int i = 0; i<n; i++)
                {
                        int newNum = int.Parse(Console.ReadLine());

                        int index = slist.BinarySearch(newNum);
                        if(index < 0)
                        {
                                slist.Insert(~index,newNum);
                        }
                        else
                        {
                                slist.Insert(index,newNum);
                        }

                        int insCount = i +1;
                        if(insCount == 1)
                        {
                                Console.WriteLine(slist[0].ToString("0.0"))        ;
                        }
                        else if(insCount %2 == 0)
                        {
                                int lower = i/2;
                                float avg = ((float)slist[lower] + (float)slist[lower+1])/2;
                                Console.WriteLine(avg.ToString("0.0"));
                        }
                        else
                        {
                                int lower = i/2;
                                float avg = ((float)slist[lower]);
                                Console.WriteLine(avg.ToString("0.0"));
                        }

                }

        }

}
```

DELETE DUPLICATE VALUE NODE

```
SinglyLinkedListNode* removeDuplicates(SinglyLinkedListNode* head) {
        if (!head) return head;
      SinglyLinkedListNode* current = head->next;
      SinglyLinkedListNode* previous = head;

      while(current !=NULL) {

         if (current->data == previous->data)
         {
               SinglyLinkedList* temp = current;
               current = current->next;
               previous->next = current;
               free(temp);



         }
         else{
               previous = current;
               current = current->next;

         }

      }
      return head;

}
```