Set-1

1. Write a function named factorTwoCount that returns the number of times that 2 divides the argument.
For example, factorTwoCount(48) returns 4 because
48/2 = 24
24/2 = 12
12/2 = 6
6/2 = 3
2 does not divide 3 evenly.
Another example: factorTwoCount(27) returns 0 because 2 does not divide 27.
The function signature is
int factorTwoCount(int n);

```
public int factorTwoCount(int n)
        {
            //48
            int count = 0,rem=0;
            while(n!=0)
            {
                rem= n%2;
                if (rem != 0)
                    break;
                n = n / 2;
                if(rem==0)
                count++;
            }
            return count;

        }
```

```
----Best
public static int factorTwoCount(int n) {
              int count = 0;
              while (n % 2 == 0) {
                     count++;
                     n = n / 2;
              }
              return count;
       }
```

@imp

2. A Daphne array is defined to be an array that contains at least one odd number and begins and ends with the same number of even numbers.
So {4, 8, 6, 3, 2, 9, 8,11, 8, 13, 12, 12, 6} is a Daphne array because it begins with three even numbers and ends with three even numbers and it contains at least one odd number
The array {2, 4, 6, 8, 6} is not a Daphne array because it does not contain an odd number.
The array {2, 8, 7, 10, -4, 6} is not a Daphne array because it begins with two even numbers but ends with three even numbers.

Write a function named *isDaphne* that returns 1 if its array argument is a Daphne array. Otherwise,

it returns 0.

If you are writing in Java or C#, the function signature is
int isDaphne (int[ ] a)
If you are writing in C or C++, the function signature is
int isDaphne (int a[ ], int len) where len is the number of elements in the array.

```
  public int isDaphne(int[] a)
        {
            //contain at lest one odd number
            //start and end with same number of even number
            int isOdd = 0, startevencount = 0, endevencount = 0, flag = 1, flagNew =
1,rtnval=0;
            for (int i = 0, j = a.Length - 1; i <= j; i++, j--)
            {
                if (a[i] % 2 == 0)
                {
                    if (flag == 1)
                    {
                        startevencount++;
                    }
                }
                else
                {
                    flag = 0;
                    isOdd = 1;
                }

                if (a[j] % 2 == 0)
                {
                    if (flagNew == 1)
                        endevencount++;
                }
                else
                {
                    flagNew = 0;
                }

            }

            if (startevencount == endevencount && isOdd == 1)
                rtnval = 1;
            return rtnval;

        }
```

3. Write a function called **goodSpread** that returns 1 if no value in its array argument occurs more than 3 times in the array.

For example, goodSpread(new int[] {2, 1, 2, 5, 2, 1, 5, 9} returns 1 because no value occurs more than three times.

But goodSpread(new int[ ] {**3**, 1, **3** ,1, **3**, 5, 5, **3**}) returns 0 because the value 3 occurs four times.

If you are writing in Java or C#, the function signature is
int goodSpread (int[ ] a)

If you are writing in C or C++, the function signature is
int goodSpread (int a[ ], int len) where len is the number of elements in the array.

```
public int goodSpread(int[] a)
        {
            //no value occurs more than three times.
            int count = 0, rtnval = 1;
            for (int i = 0; i < a.Length; i++)
            {
                for (int j = 0; j < a.Length; j++)
                {
                    if (a[i] == a[j])
                    {
                        count++;
                    }
                }

                if (count > 3)
                {
                    rtnval = 0;
                    break;
                }
                count = 0;
            }
            return rtnval;
        }
```

Set-2

1. Write a function named *sumDigits* that sums the digits of its integer argument. For example sumDigits(3114) returns 9, sumDigits(-6543) returns 18 and sumDigits(0) returns 0.

The signature of the function is
int sumDigits (int n)

```
public int sumDigits(int n)
        {
            int rem = 0,sum=0;
            if (n < 0)
                n = (-1) * n;
            while(n!=0)
            {
```

```
            rem = n % 10;
            n = n / 10;
            sum += rem;
        }
        return sum;
    }
```

2. Define a **Meera array** to be an array where a[n] is less than n for n = 0 to a.length-1.

For example, {-4, 0, 1, 0, 2} is a Meera array because
a[0] < 0
a[1] < 1
a[2] < 2
a[3] < 3
a[4] < 4

{-1, 0, 0, 8, 0} is not a Meera array because a[3] is 8 which is not less than 3.

Write a function named *isMeera* that returns 1 if its array argument is a Meera array. Otherwise it returns 0.

If you are programming in Java or C#, the function signature is
int isMeera (int[ ] a)

If you are programming in C or C++, the function signature is
int isMeera (int a[ ], int len) where len is the number of elements in the array.

```
public int isMeera(int[] a)
    {
        for(int i=0;i<a.Length;i++)
        {
            if(a[i]>=i)
            {
                return 0;
            }
        }
        return 1;
    }
```

3. Define a **Dual array** to be an array where every value occurs exactly twice.

For example, {1, 2, 1, 3, 3, 2} is a dual array.

The following arrays are **not** Dual arrays
{2, 5, 2, 5, 5} (5 occurs three times instead of two times)
{3, 1, 1, 2, 2} (3 occurs once instead of two times)

Write a function named *isDual* that returns 1 if its array argument is a Dual array. Otherwise it returns 0.

If you are programming in Java or C#, the function signature is
int isDual (int[ ] a)

If you are programming in C or C++, the function signature is

int isDual (int a[ ], int len) where len is the number of elements in the array.

Hint: you need a nested loop.

```
public int isDual(int[] a)
        {
            int count = 0;
            for (int i = 0; i < a.Length; i++)
            {
                for (int j = 0; j < a.Length; j++)
                {
                    if(a[i]==a[j])
                    {
                        count++;
                    }
                }
                if (count != 2)
                    return 0;
                count = 0;
            }
            return 1;
        }
```

Set-3

1 . An array is defined to be a **Bean** array if the sum of the primes in the array is equal to the first element of the array. If there are no primes in the array, the first element must be 0. So {21, 3, 7, 9, 11 4, 6} is a Bean array because 3, 7, 11 are the primes in the array and they sum to 21 which is the first element of the array. {13, 4, 4,4, 4} is also a Bean array because the sum of the primes is 13 which is also the first element. Other Bean arrays are {10, 5, 5}, {0, 6, 8, 20} and {3}. {8, 5, -5, 5, 3} is **not** a Bean array because the sum of the primes is 5+5+3 = 13 but the first element of the array is 8. Note that -5 is not a prime because prime numbers are positive.

Write a function named *isBeanArray* that returns 1 if its integer array argument is a Bean array. Otherwise it returns 0.

If you are writing in Java or C#, the function signature is
int isBeanArray (int[ ] a)

If you are writing in C or C++, the function signature is
int isBeanArray (int a[ ], int len) where len is the number of elements in the array.

You may assume that a function named *isPrime* exists that returns 1 if its int argument is a prime, otherwise it returns 0. **You do **not** have to write this function!** You just have to call it.

```
public int isBeanArray(int[] a)
        {
            Prime objPrime = new Prime();
            int sum = 0;
```

```
            for(int i=0;i<a.Length;i++)
            {
                if(a[0]==0)
                {
                    if(objPrime.IsPrime(a[i]))
                    {
                        return 0;
                    }
                }
                else
                {
                    if (objPrime.IsPrime(a[i]))
                    {
                        sum += a[i];
                    }
                }
            }
            if (sum == a[0])
                return 1;
            return 0;
        }
```

2. An array is defined to be **complete** if all its elements are greater than 0 and all even numbers that are less than the maximum even number are in the array.

For example {2, 3, 2, 4, 11, 6, 10, 9, 8} is complete because
a. all its elements are greater than 0
b. the maximum even integer is 10
c. all even numbers that are less than 10 (2, 4, 6, 8) are in the array.

But {2, 3, 3, 6} is **not** complete because the even number 4 is missing. {2, -3, 4, 3, 6} is **not** complete because it contains a negative number.

Write a function named *isComplete* that returns 1 if its array argument is a complete array. Otherwise it returns 0.

If you are writing in Java or C#, the function signature is
int isComplete (int[ ] a)

If you are writing in C or C++, the function signature is
int isComplete (int a[ ], int len) where len is the number of elements in the array.

```
public int isComplete(int[] a)
        {
            int maxEvenNum = 0;
            for(int i =0;i<a.Length;i++)
            {
                if (a[i] <= 0)
                    return 0;
                if(a[i]%2==0)
                {
                    if(maxEvenNum<a[i])
                    {
                        maxEvenNum = a[i];
                    }
                }
```

```
            }

            for(int i =2;i<maxEvenNum;i=i+2)
            {
                bool flag = false;
                for(int j=0;j<a.Length;j++)
                {
                    if (i == a[j])
                        flag = true;
                }
                if (!flag)
                    return 0;
            }
            return 1;
        }
```

@imp

3. An integer is defined to be a **Bunker** number if it is an element in the infinite sequence 1, 2, 4, 7, 11, 16, 22, … Note that 2-1=1, 4-2=2, 7-4=3, 11-7=4, 16-11=5 so for k>1, the kth element of the sequence is equal to the k-1th element + k-1. E.G., for k=6, 16 is the kth element and is equal to 11 (the k-1th element) + 5 (k-1).

Write function named *isBunker* that returns 1 if its argument is a Bunker number, otherwise it returns 0. So *isBunker*(11) returns 1*, isBunker*(22) returns 1 and *isBunker*(8) returns 0 .

The function signature is
int isBunker (int n)

```
public static int isBunker(int n) {
             int result = 0;
            int prevNum = 1;
            int currentNum = 0;
            for (int k = 2; currentNum < n; k++)
            {
                currentNum = prevNum + k - 1;
                prevNum = currentNum;
            }
            if (currentNum == n)
            {
                result = 1;
            }
            return result;
        }
```

Set-4

1. An array is defined to be a **Filter array** if it meets the following conditions

a. If it contains 9 then it also contains 11.
b. If it contains 7 then it does **not** contain 13.

So {1, 2, 3, **9**, 6, **11**} and {3, 4, 6, **7**, 14, 16}, {1, 2, 3, 4, 10, 11, 13} and {3, 6, 5, 5, 13, 6, 13} are Filter arrays. The following arrays are **not** Filter arrays: {9, 6, 18} (contains 9 but no 11), {4, 7, 13} (contains both 7 and 13)

Write a function named *isFilter* that returns 1 if its array argument is a Filter array, otherwise it returns 0.

If you are programming in Java or C#, the function signature is
int isFilter(int[ ] a)
If you are programming in C or C++, the function signature is
int isFilter(int a[ ], int len) where len is the number of elements in the array.

```
public int isFilter(int[] a)
      {

          for (int i = 0; i < a.Length; i++)
          {
              bool flag = false;
              if (a[i] == 9)
              {
                  for (int j = 0; j < a.Length; j++)
                  {
                      if (a[j] == 11)
                      {
                          flag = true;
                      }
                  }
                  if (!flag)
                      return 0;
              }

              if (a[i] == 7)
              {

                  for (int k = 0; k < a.Length; k++)
                  {
                      if(a[k]==13)
                      {
                          return 0;
                      }
                  }
              }

          }
          return 1;
      }
```

2. A **Fibonacci number** is a number in the sequence 1, 1, 2, 3, 5, 8, 13, 21,…. Note that first two Fibonacci numbers are 1 and any Fibonacci number other than the first two is the sum of the previous two

Fibonacci numbers. For example, 2 = 1 + 1, 3 = 2 + 1, 5 = 3 + 2 and so on.

Write a function named *isFibonacci* that returns 1 if its integer argument is a Fibonacci number, otherwise it returns 0.

The signature of the function is
int isFibonacci (int n)

```
public int isFibonacci(int n)
        {
            if (n == 1)
                return 1;
            int a=1, b=1, c=0;
             while (c < n)
            {
                c = a + b;
                a = b;
                b = c;
                if(c==n)
                {
                    return 1;
                }

            }
            return 0;
        }
```

3. A **Meera array** is an array that contains the value 0 if and only if it contains a prime number. The array {7, 6, 0, 10, 1} is a Meera array because it contains a prime number (7) and also contains a 0. The array {6, 10, 1} is a Meera array because it contains no prime number and also contains no 0.

The array {7, 6, 10} is **not** a Meera array because it contains a prime number (7) but does not contain a 0. The array {6, 10, 0} is **not** a Meera array because it contains a 0 but does not contain a prime number.

It is okay if a Meera array contains more than one value 0 and more than one prime, so the array {3, 7, 0, 8, 0, 5} is a Meera array (3, 5 and 7 are the primes and there are two zeros.).

Write a function named *isMeera* that returns 1 if its array argument is a Meera array and returns 0 otherwise.

You may assume the existence of a function named *isPrime* that returns 1 if its argument is a prime and returns 0 otherwise. You do not have to write isPrime, you can just call it.

If you are programming in Java or C#, the function signature is
int isMeera(int [ ] a)

If you are are programming in C or C++, the function signature is
int isMeera(int a[ ], int len) where *len* is the number of elements in the array.

--AnojTrick

```
    public int isMeera(int[] a)
        {
            Prime objPrime = new Prime();
            for (int i = 0; i < a.Length; i++)
            {
                if (objPrime.IsPrime(a[i]) || a[i] == 0)
                {
                    bool flag = true;
                    for (int j = 0; j < a.Length; j++)
```

```
                    {
                        if (a[i] == 0 && objPrime.IsPrime(a[j]) ||
(objPrime.IsPrime(a[i]) && a[j] == 0))
                        {
                            flag = false;
                        }
                    }
                    if (flag)
                        return 0;
                }

            }
            return 1;
        }
```

Set-5

1. A **Bean array** is defined to be an array where for every value *n* in the array, there is also an element n-1 or n+1 in the array.

For example, {2, 10, 9, 3} is a Bean array because
2 = 3-1
10 = 9+1
3 = 2 + 1
9 = 10 -1

Other Bean arrays include {2, 2, 3, 3, 3}, {1, 1, 1, 2, 1, 1} and {0, -1, 1}.

The array {3, 4, 5, 7} is **not** a Bean array because of the value 7 which requires that the array contains either the value 6 (7-1) or 8 (7+1) but neither of these values are in the array.

Write a function named *isBean* that returns 1 if its array argument is a *Bean* array. Otherwise it returns a 0.

If you are programming in Java or C#, the function signature is
int isBean(int[ ] a)

If you are programming in C or C++, the function signature is
int isBean(int a[ ], int len) where *len* is the number of elements in the array.

```
public int isBean(int[] a)
        {
            for(int i=0;i<a.Length;i++)
            {
                bool flag = false;
                for(int j=0;j<a.Length;j++)
                {
                    if(a[i]+1 ==a[j] || a[i]-1==a[j])
                    {
                        flag = true;
                    }
                }
                if (!flag)
                    return 0;
            }
```

```
            return 1;
        }
```

2. Write a function named **minDistance** that returns the smallest distance between two factors of a number. For example, consider 13013 = 1*7*11*13. Its factors are 1, 7, 11, 13 and 13013. **minDistance**(13013) would return 2 because the smallest distance between any two factors is 2 (13 - 11 = 2). As another example, **minDistance** (8) would return 1 because the factors of 8 are 1, 2, 4, 8 and the smallest distance between any two factors is 1 (2 – 1 = 1).

The function signature is
  int **minDistance**(int n)

```
public int minDistance1(int n)
        {
            int d = n;
            int f = 1;
            for (int i = 2; i <= n; i++)
            {
                if (n % i == 0)
                {
                    if (i - f < d)
                    {
                        d = i - f;
                    }
                    f = i;
                }
            }
            return d;

        }
```

3. A **wave array** is defined to an array which does **not** contain two even numbers or two odd numbers in adjacent locations. So {7, 2, 9, 10, 5}, {4, 11, 12, 1, 6}, {1, 0, 5} and {2} are all **wave arrays**. But {**2, 6**, 3, 4} is not a wave array because the even numbers 2 and 6 are adjacent to each other.

Write a function named *isWave* that returns 1 if its array argument is a Wave array, otherwise it returns 0.

If you are programming in Java or C#, the function signature is
  int isWave (int [ ] a)

If you are programming in C or C++, the function signature is
  int isWave (int a[ ], int len) where len is the number of elements in the array.

```
public static int isWave(int[] a) {
            boolean flag = false;
            for (int i = 1; i < a.length; i++) {
                  if ((a[i - 1] % 2 == 0 && a[i] % 2 == 0)
                              || (a[i - 1] % 2 != 0 && a[i] % 2 != 0)) {
                        flag = true;
```

```
                }
            }
            if (flag) {
                return 0;
            }
            return 1;
      }
```

Set-6

1. An array is defined to be a **Bean array** if it meets the following conditions
   a. If it contains a 9 then it also contains a 13.
   b. If it contains a 7 then it does **not** contain a 16.

So {1, 2, 3, **9**, 6, **13**} and {3, 4, 6, **7**, 13, 15}, {1, 2, 3, 4, 10, 11, 12} and {3, 6, 9, 5, 7, 13, 6, 17} are Bean arrays. The following arrays are **not** Bean arrays:
   a. { 9, 6, 18} (contains a 9 but no 13)
   b. {4, 7, 16} (contains both a 7 and a 16)

Write a function named isBean that returns 1 if its array argument is a Bean array, otherwise it returns 0.

If you are programming in Java or C#, the function signature is
   int isBean (int[ ] a)

If you are programming in C or C++, the function signature is
   int isBean (int a[ ], int len) where len is the number of elements in the array.

```
public int isBean(int[] a)
      {
            int flag1 = 1, flag2 = 1;
          for(int i =0;i<a.Length;i++)
          {

              if(a[i]==9 || a[i]==7)
              {
                  for(int j=0;j<a.Length;j++)
                  {
                      if (a[i] == 9)
                          flag1 = 0;
                      if(a[i]==9 && a[j]==13)
                      {
                          flag1 = 1;
                      }
                      if(a[i]==7 && a[j]==16)
                      {
                          flag2 = 0;
                      }
                  }
              }
          }
```

```
            if (flag1 == 1 && flag2 == 1)
                return 1;
            return 0;
        }
```

2. A **Meera number** is a number such that the number of nontrivial factors is a factor of the number. For example, 6 is a Meera number because 6 has two nontrivial factors : 2 and 3. (A nontrivial factor is a factor other than 1 and the number). Thus 6 has two nontrivial factors. Now, 2 is a factor of 6. Thus the number of nontrivial factors is a factor of 6. Hence 6 is a Meera number. Another Meera number is 30 because 30 has 2, 3, 5, 6, 10, 15 as nontrivial factors. Thus 30 has 6 nontrivial factors. Note that 6 is a factor of 30. So 30 is a Meera Number. However 21 is **not** a Meera number. The nontrivial factors of 21 are 3 and 7. Thus the number of nontrivial factors is 2. Note that 2 is not a factor of 21. Therefore, 21 is not a Meera number.

Write a function named *isMeera* that returns 1 if its integer argument is a Meera number, otherwise it returns 0.

The signature of the function is

int isMeera(int n)

```
public int isMeera(int n)
        {
            int count=0;
            for(int i=2;i<n;i++)
            {
                if (n%i == 0)
                    count++;
            }

            if (n%count == 0)
                return 1;
           return 0;

        }
```

3. A **Bunker array** is an array that contains the value 1 if and only if it contains a prime number. The array {7, 6, 10, 1} is a Bunker array because it contains a prime number (7) and also contains a 1. The array {7, 6, 10} is **not** a Bunker array because it contains a prime number (7) but does not contain a 1. The array {6, 10, 1} is **not** a Bunker array because it contains a 1 but does not contain a prime number.

It is okay if a Bunker array contains more than one value 1 and more than one prime, so the array {3, 7, 1, 8, 1} is a Bunker array (3 and 7 are the primes).

Write a function named *isBunker* that returns 1 if its array argument is a Bunker array and returns 0 otherwise.

You may assume the existence of a function named *isPrime* that returns 1 if its argument is a prime and returns 0 otherwise. You do not have to write isPrime, you can just call it.

If you are programming in Java or C#, the function signature is
int isBunker(int [ ] a)

If you are programming in C or C++, the function signature is

int isBunker(int a[ ], int len) where len is the number of elements in the array.

```
public int isBunker(int[] a)
        {
            Prime objPrime = new Prime();

            for (int i = 0; i < a.Length; i++)
            {
                if(objPrime.IsPrime(a[i]))
                {
                  bool  flag = true;
                    for(int j=0;j<a.Length;j++)
                    {
                        if (a[j] == 1)
                            flag = false;
                    }
                    if (flag)
                        return 0;
                }

            }
            return 1;

        }
```

Set-7

1. A **Nice array** is defined to be an array where for every value n in the array, there is also an element n-1 or n+1 in the array.

For example, {2, 10, 9, 3} is a Nice array because
2 = 3-1
10 = 9+1
3 = 2 + 1
9 = 10 -1

Other Nice arrays include {2, 2, 3, 3, 3}, {1, 1, 1, 2, 1, 1} and {0, -1, 1}.

The array {3, 4, 5, 7} is **not** a Nice array because of the value 7 which requires that the array contains either the value 6 (7-1) or 8 (7+1) but neither of these values are in the array.

Write a function named *isNice* that returns 1 if its array argument is a Nice array. Otherwise it returns a 0.

If you are programming in Java or C#, the function signature is
int isNice(int[ ] a)

If you are programming in C or C++, the function signature is
int isNice(int a[ ], int len) where len is the number of elements in the array.

```
public int isNice(int[] a)
        {
            for (int i = 0; i < a.Length; i++)
            {
                bool flag = true;
                for (int j = 0; j < a.Length; j++)
                {
                    if (a[i] == a[j] + 1 || a[i] == a[j] - 1)
                    {
                        flag = false;
```

```
                }
            }
            if (flag)
                return 0;
        }
        return 1;

    }
```

2. A **Pascal number** is a number that is the sum of the integers from 1 to j for some j. For example 6 is a Pascal number because 6 = 1 + 2 + 3. Here j is 3. Another Pascal number is 15 because 15 = 1 + 2 + 3 + 4 + 5. An example of a number that is not a Pascal number is 7 because it falls between the Pascal numbers 6 and 10.

Write a function named *isPascal* that returns 1 if its integer argument is a Pascal number, otherwise it returns 0.

The signature of the function is

int isPascal (int n)

```
public int isPascal(int n)
    {
        int sum = 0;
        for (int i = 1; i < n; i++)
        {
            sum += i;
            if (n == sum)
                return 1;
        }
        return 0;

       }
```

March 7, 2015

QUESTION 3. An array is called ***balanced*** if its even numbered elements (a[0], a[2], etc.) are even and its odd numbered elements (a[1], a[3], etc.) are odd. Write a function named *isBalanced* that accepts an array of integers and returns 1 if the array is balanced, otherwise it returns 0. Examples: {2, 3, 6, 7} is balanced since a[0] and a[2] are even, a[1] and a[3] are odd. {6, 7, 2, 3, 12} is balanced since a[0], a[2] and a[4] are even, a[1] and a[3] are odd. {7, 15, 2, 3} is not balanced since a[0] is odd. {16, 6, 2, 3} is not balanced since a[1] is even.

If you are programming in Java or C#, the function signature is

int isBalanced(int[ ] a)

If you are programming in C or C++, the function signature is

int isBalanced(int a[ ], int len)

where *len* is the number of elements in the array.

```
public int isBalanced(int[] a)
        {
            for(int i=0;i<a.Length;i++)
            {
                if((i%2==0 && a[i]%2!=0 ) || (i%2!=0 && a[i]%2==0))
                {
                    return 0;
                }
            }
            return 1;

        }
```

Set-8

QUESTION 1. An array is defined to be ***odd-heavy*** if it contains at least one odd element and every odd element is greater than every even element. So {11, 4, 9, 2, 8} is odd-heavy because the two odd elements (11 and 9) are greater than all the even elements. And {11, 4, 9, 2, 3, 10} is not odd-heavy because the even element 10 is greater than the odd element 9. Write a function called *isOddHeavy* that accepts an integer array and returns 1 if the array is odd-heavy; otherwise it returns 0. Some other examples: {1} is odd-heavy, {2} is not odd-heavy, {1, 1, 1, 1} is odd-heavy, {2, 4, 6, 8, 11} is odd-heavy, {-2, -4, -6, -8, -11} is not odd-heavy.

If you are programming in Java or C#, the function signature is

int isOddHeavy(int[ ] a)

If you are programming in C or C++, the function signature is

int isOddHeavy(int a[ ], int len)

where *len* is the number of elements in the array.

Code:

```
  public static int isOddHeavy(int[] a)
       {
           bool isOdd = false;
           for(int i=0;i<a.Length;i++)
           {
               if(a[i]%2!=0)//Odd
               {
                   isOdd = true;
                   for(int j=0;j<a.Length;j++)
                   {
                       if (a[j] % 2 == 0 && a[j] > a[i])
                           return 0;
                   }
               }
           }
           if (!isOdd)
               return 0;
           return 1;


       }
```

QUESTION 2. A ***normal*** number is defined to be one that has no odd factors, except for 1 and possibly itself. Write a method named *isNormal* that returns 1 if its integer argument is normal, otherwise it returns 0. The function signature is

int isNormal(int n)

Examples: 1, 2, 3, 4, 5, 7, 8 are normal numbers. 6 and 9 are not normal numbers since 3 is an odd factor. 10 is not a normal number since 5 is an odd factor.

```
public int isNormal(int n)
       {
           for (int i = 2; i < n; i++)
           {
               if (n % i == 0 && i % 2 != 0)
               {
                   return 0;
               }
           }
           return 1;
```

```
}
```

Question 3. Write a function fill with signature

int[] fill(int[] arr, int k, int n)

which does the following: It returns an integer array arr2 of length n whose first k elements are the same as the first k elements of arr, and whose remaining elements consist of repeating blocks of the first k elements. You can assume array arr has at least k elements. The function should return null if either k or n is not positive.

Examples: fill({1,2,3,5, 9, 12,-2,-1}, 3, 10) returns {1,2,3,1,2,3,1,2,3,1}. Fill({4, 2, -3, 12}, 1, 5) returns {4, 4, 4, 4, 4}. fill({2, 6, 9, 0, -3}, 0, 4) returns null.

```
public static int[] fill(int[] arr, int k, int n) {
		int[] a = null;
		int count = 0;
		while (count < n && k>0 && arr.length>k) {
			if(count==0){
				a = new int[n];
			}

			for (int i = 0; i < k && count < n; i++) {
				a[count] = arr[i];
				count++;
			}
		}
		return a;
	}
```

Set-9

Question 1. Write a function sumIsPower with signatuare

boolean sumIsPower(int[] arr)

which outputs true if the sum of the elements in the input array arr is a power of 2, false otherwise. Recall that the powers of 2 are 1, 2, 4, 8, 16, and so on. In general a number is a power of 2 if and only if it is of the form 2^n for some nonnegative integer n. You may assume (without verifying in your code) that all elements in the array are positive integers. If the input array arr is null, the return value should be false.

Examples: sumIsPower({8,8,8,8}) is true since 8 + 8 + 8 + 8 = 32 = 2^5. sumIsPower({8,8,8}) is false, since 8 + 8 +8 = 24, not a power of 2.

```
public bool sumIsPower(int[] arr)
	{
	    int sum = 0;
	    for(int i=0;i<arr.Length;i++)
	    {
```

```
                sum += arr[i];
            }
            if (IsPowerOf2(sum))
                return true;
            return false;
        }

        private bool IsPowerOf2(int n)
        {
            int x = (n & (n - 1));
            return n > 0 && (n & (n - 1)) == 0;
        }
```

-----Best Practice

```
  public static bool sumIsPower(int[] a)
        {
            bool result = false;
            int sum = 0;
            for (int i = 0; i < a.Length; i++)
            {
                sum += a[i];
            }
            while (sum % 2 == 0)
            {
                sum /= 2;
            }
            if (sum == 1)
            {
                result = true;
            }
            else
            {
                result = false;
            }
            return result;
        }
```

@imp

Question 2. An array is said to be hollow if it contains 3 or more zeros in the middle that are preceded and followed by the same number of non-zero elements. Write a function named isHollow that accepts an integer array and returns 1 if it is a hollow array, otherwise it returns 0. The function signature is

int isHollow(int[ ] a).

Examples: isHollow({1,2,4,0,0,0,3,4,5}) returns true. isHollow ({1,2,0,0,0,3,4,5}) returns false. : isHollow ({1,2,4,9, 0,0,0,3,4, 5}) returns false. isHollow ({1,2, 0,0, 3,4}) returns false.

```
   public int isHollow(int[] a)
        {
            int countZero = 0, firstcount = 0, lastcount = 0;
```

```
        //for firstcount
        for (int j = 0; j < a.Length; j++)
        {
            if (a[j] == 0)
                break;
            firstcount++;
        }
        //for lastcount
        for (int k = a.Length - 1; k >= 0; k--)
        {
            if (a[k] == 0)
                break;
            lastcount++;
        }
        //1,2,0,0,0,1,2
        if (firstcount == lastcount)
        {
            for (int i = firstcount; i < a.Length - lastcount; i++)
            {
                if (a[i] != 0)
                    return 0;
                countZero++;
                if (countZero > 2)
                {
                    return 1;
                }
            }
        }
        return 0;

    }
```

3. A Riley number is an integer whose digits are all even. For example 2426 is a Riley number but 3224 is not.

Write a function named *isRiley* that returns 1 if its integer argument is a Riley number otherwise it returns 0.

The function signature is
int isRiley (int n)

```
public int isRiley(int n)
    {
        //2426  all digits should be even
        while(n!=0)
        {
            if ((n % 10)%2 != 0)
                return 0;
            n = n / 10;
        }
        return 1;

    }
```

Set-10

1. Write a function named *lastEven* that returns the index of the last even value in its array argument. For example, lastEven will return 3 if the array is {3, 2, 5, 6, 7}, because that is the index of 6 which is the last even value in the array.

If the array has no even numbers, the function should return -1.

If you are programming in Java or C#, the function signature is
int lastEven (int[ ] a)

If you are programming in C or C++, the function signature is
int lastEven (int a[ ], int len) where len is the number of elements in a.

```
public int lastEvens(int[] a)
    {
        for (int i = a.Length-1; i>=0; i--)
        {
            if (a[i] % 2 == 0)
                return i;
        }
            return -1;

    }
```

2. Write a function named *countMax* that returns the number of times that the max value occurs in the array. For example, countMax would return 2 if the array is {6. 3, 1, 3, 4, 3, 6, 5} because 6 occurs 2 times in the array.

If you are programming in Java or C#, the function signature is
int countMax (int[ ] a)

If you are programming in C or C++, the function signature is
int countMax (int a[ ], int len) where len is the number of elements in a.

3. An integer is defined to be an **even subset** of another integer *n* if every even factor of *m* is also a factor of *n*. For example 18 is an even subset of 12 because the even factors of 18 are 2 and 6 and these are both factors of 12. But 18 is not an even subset of 32 because 6 is not a factor of 32.

Write a function with signature **int isEvenSubset(int m, int n)** that returns 1 if m is an even subset of n, otherwise it returns 0.

```
public int isEvenSubset(int m, int n)
    {
        for (int i = 1; i < m; i++)
        {
            if (m % i == 0 && i % 2 == 0)
            {
                bool flag = false;
                for (int j = 1; j < n; j++)
```

```
                {
                    if (n % j == 0 && j % 2 == 0 && i == j)
                        flag = true;
                }
                if (!flag)
                    return 0;
                flag = false;
            }
        }
        return 1;


    }
```

Set-10

1. A **twinoid** is defined to be an array that has exactly two even values that are adjacent to one another. For example {3, 3, 2, 6, 7} is a twinoid array because it has exactly two even values (2 and 6) and they are adjacent to one another. The following arrays are not twinoid arrays.

{3, 3, 2, 6, 6, 7} because it has three even values.

{3, 3, 2, 7, 6, 7} because the even values are not adjacent to one another

{3, 8, 5, 7, 3} because it has only one even value.

Write a function named **isTwinoid** that returns 1 if its array argument is a twinoid array. Otherwise it returns 0.

If you are programming in Java or C#, the function signature is
int isTwinoid (int [ ] a);
If you are programming in C or C++, the function signature is
int isTwinoid(int a[ ], int len) where len is the number of elements in the array.

--Best Method

```
public static int isTwinoid(int[] a)
        {
            int result = 0;
            for (int i = 1; i < a.Length; i++)
            {
                if (a[i] % 2 == 0)
                {
                    if (a[i - 1] % 2 != 0 && a[i + 1] % 2 == 0 || a[i - 1] % 2 == 0 &&
a[i + 1] % 2 != 0)
                    {
                        result = 1;
                    }
                    else
                    {
```

```
                    result = 0;
                    break;
                }
            }
        }
        return result;
    }
--Next Method...
   public int isTwinoid(int[] a)
    {
        int count=0,flag=0;
        for (int i = 1; i < a.Length; i++)
        {
            if (a[i - 1] % 2 == 0 && a[i] % 2 == 0)
                flag = 1;
            if ( a[i] % 2 == 0)
                count++;
        }
        if (a[0] % 2 == 0)
            count = count + 1;
        if (count == 2 && flag==1)
            return 1;
        return 0;

    }
```

2. A **balanced** array is defined to be an array where for every value n in the array, -n also is in the array. For example {-2, 3, 2, -3} is a balanced array. So is {-2, 2, 2, 2}. But {-5, 2, -2} is not because 5 is not in the array.

Write a function named isBalanced that returns 1 if its array argument is a balanced array. Otherwise it returns 0.

If you are programming in Java or C#, the function signature is

int isBalanced (int [ ] a);

If you are programming in C or C++, the function signature is

int isBalanced(int a[ ], int len) where len is the number of elements in the array.

```
public int isBalanced(int[] a)
    {
        for(int i=0;i<a.Length;i++)
        {
            bool flag = false;
            for(int j=0;j<a.Length;j++)
            {
                if(a[i]+a[j]==0)
                {
                    flag = true;
                }
            }
            if (!flag)
```

```
                return 0;
            }
            return 1;

        }
```

3. Write a method named **getExponent(n, p)** that returns the largest exponent x such that p^x evenly divides n. If p is <= 1 the method should return -1.

For example, getExponent(162, 3) returns 4 because $162 = 2^1 * 3^4$, therefore the value of x here is 4.

The method signature is
int getExponent(int n, int p)

Examples:

if n is	and p is	return	Because
27	3	3	3^3 divides 27 evenly but 3^4 does not.
28	3	0	3^0 divides 28 evenly but 3^1 does not.
280	7	1	7^1 divides 280 evenly but 7^2 does not.
-250	5	3	5^3 divides -250 evenly but 5^4 does not.
18	1	-1	if p <=1 the function returns -1.
128	4	3	4^3 divides 128 evenly but 4^4 does not.

```
--Best Method
  public static int getExponent(int n, int p)
        {
            int num = n;
            int count = 0;
            if (p <= 1)
            {
                count = -1;
            }
            else
            {
                while (num % p == 0)
                {
                    num /= p;
                    count++;
                }
            }
            return count;
        }

--My method
public int getExponent(int n, int p)
        {
            int dividend = 1;
            if (p <= 1)
                return -1;
            if (n < 0)
```

```
            n = n * -1;
        for(int i=0;i<n;i++)
        {
            if (n % dividend !=0)
            {
                return i-1;
            }
            dividend *= p;
        }
        return dividend;


    }

public static int getExponent(int n, int b) {

          int ne = 1;
          int i = 0;
          if (b <= 1) {
                i = 1;
          } else {
                for (i = 0; i < ne; i++) {
                      int product = 1;
                      for (int j = 0; j < i; j++) {
                            product *= b;
                      }
                      if (n % product == 0) {
                            ne++;
                      }
                }
          }
          return (i - 2);
    }
//next logic
    int getExponent(int n, int p) {
    if(p>1)
    {
          int  i=1;
          int count=0;
          while(n%i==0)
          {
                i=i*p;
                if(n%i==0)
                {
                      count++;
                }

          }
          return count;
    }
    return -1;
    }
```

Set-11

1. An array is defined to be **maxmin equal** if it contains at least two **different** elements and the number of times the maximum value occur is the same as the number of times the minimum value occur. So {11, 4, 9, 11, 8, 5 , 4, 10} is **maxmin equal**, because the max value 11 and min value 4 both appear two times in the array.

Write a function called *isMaxMinEqual* that accepts an integer array and returns 1 if the array is **maxmin equal**; otherwise it returns 0.

If you are programming in Java or C#, the function signature is
int *isMaxMinEqual*(int[ ] a)

If you are programming in C or C++, the function signature is
int *isMaxMinEqual*(int a[ ], int len) where len is the number of elements in the array

Some other examples:

if the input array is	*isMaxMinEqual* **should return**
{}	0 (array must have at least two *different* elements)
{2}	0 (array must have at least two *different* elements)
{1, 1, 1, 1, 1, 1}	0 (array must have at least two *different* elements)
{2, 4, 6, 8, 11}	1 (Both max value (11) and min value 2 appear exactly one time)
{-2, -4, -6, -8, -11}	1 (Both max value (-2) and min value -11 appear exactly one time)

```
public int isMaxMinEqual(int[] a)
     {
         if (a.Length <= 1)
             return 0;
         int maxCount = 0, mincount = 0,flag=0;
         int minval = a[0], maxval = a[0];
         for (int i = 0; i < a.Length; i++)
         {
             if (maxval < a[i])
             {
                 maxval = a[i];
             }
             if(minval>a[i])
             {
                 minval = a[i];
             }
             if (a[0] != a[i])
                 flag = 1;
         }
         for(int j=0;j<a.Length;j++)
         {
             if (maxval == a[j])
                 maxCount++;
             if (minval == a[j])
                 mincount++;
         }
         if (maxCount == mincount && flag == 1)
             return 1;
         return 0;
```

```
}
```

2. An integer array is said to be *oddSpaced*, if the difference between the largest value and the smallest value is an odd number. Write a function *isOddSpaced(int[] a)* that will return 1 if it is*oddSpaced* and 0 otherwise. If array has less than two elements, function will return 0. If you are programming in C or C++, the function signature is:

int *isOddSpaced* (int a[ ], int len) where *len* is the number of elements in the array.

Examples

Array	Largest value	Smallest value	Difference	Return value
{100, 19, 131, 140}	140	19	140 -19 = 121	1
{200, 1, 151, 160}	200	1	200 -1 = 199	1
{200, 10, 151, 160}	200	10	200 -10 = 190	0
{100, 19, -131, -140}	100	-140	100 - (-140) = 240	0
{80, -56, 11, -81}	80	-81	-80 - 80 = -161	1

```
    public int isOddSpaced(int[] a)
        {
            if (a.Length <= 1)
                return 0;
            int minval = a[0], maxval = a[0],flag=0;
            for (int i = 0; i < a.Length; i++)
            {
                if (maxval < a[i])
                {
                    maxval = a[i];
                }
                if (minval > a[i])
                {
                    minval = a[i];
                }
                if (a[0] != a[i])
                   flag = 1;
            }
            if ((maxval - minval) % 2 != 0 && flag==1)
                return 1;
            return 0;


        }
```

3. An *Super array* is defined to be an array in which each element is greater than sum of all elements before that. See examples below:
{2, 3, 6, 13} is a *Super* array. Note that 2 < 3, 2+3 < 6, 2 + 3 + 6 < 13.
{2, 3, 5, 11} is a NOT a *Super* array. Note that 2 + 3 not less than 5.
Write a function named *isSuper* that returns 1 if its array argument is a *isSuper* array, otherwise it returns 0.
If you are programming in Java or C#, the function signature is:

int *isSuper* (int [ ] a)
If you are programming in C or C++, the function signature is:
int *isSuper* (int a[ ], int len) where *len* is the number of elements in the array.

```
  public int isSuper (int [ ] a)
        {
            int sum = 0;
           for(int i=1;i<a.Length;i++)
           {
             for(int j=0;j<i;j++)
             {
                 sum += a[j];
             }
             if (a[i] <= sum)
                 return 0;
             sum = 0;
           }
           return 1;

        }
```

Set-12

1. An *isSym* (even/odd Symmetric) *array* is defined to be an array in which even numbers and odd numbers appear in the same order from "*both directions*". You can assume array has at least one element. See examples below:
{2, 7, 9, 10, 11, 5, 8} is a *isSym* array.
Note that from left to right or right to left we have even, odd, odd, even, odd, odd, even.
{9, 8, 7, 13, 14, 17} is a *isSym* array.
Note that from left to right or right to left we have {odd, even, odd, odd, even, odd}.
However, {2, 7, 8, 9, 11, 13, 10} is not a *isSym* array.
From left to right we have {even, odd, even, odd, odd, odd, even}.
From right to left we have {even, odd, odd, odd, even, odd, even},
which is not the same.
Write a function named *isSym* that returns 1 if its array argument is a *isSym* array, otherwise it returns 0.
If you are programming in Java or C#, the function signature is:
int *isSym* (int [ ] a)
If you are programming in C or C++, the function signature is:
int *isSym* (int a[ ], int len) where *len* is the number of elements in the array.

```
public int isSym (int [ ] a)
        {
            int rtnVal = 0;
            if(a.Length>=1)
            {
                for (int i = 0, j = a.Length - 1; i < j; i++, j--)
                {
                    //cheking odd or Even
                    if ((a[i] % 2 == 0 && a[j] % 2 == 0) || (a[i] % 2 != 0 && a[j] % 2 !=
0))//
                    {
                        rtnVal = 1;
                    }
```

```
                else
                {
                    rtnVal = 0;
                    break;
                }

            }
        }

        return rtnVal;

    }
```

2. An integer array is said to be *evenSpaced*, if the difference between the largest value and the smallest value is an even number. Write a function *isEvenSpaced(int[] a)* that will return 1 if it is *evenSpaced* and 0 otherwise. If array has less than two elements, function will return 0. If you are programming in C or C++, the function signature is:

int *isEvenSpaced* (int a[ ], int len) where *len* is the number of elements in the array.

Examples

Array	Largest value	Smallest value	Difference	Return value
{100, 19, 131, 140}	140	19	140 -19 = 121	0
{200, 1, 151, 160}	200	1	200 -1 = 199	0
{200, 10, 151, 160}	200	10	200 -10 = 190	1
{100, 19, -131, -140}	100	-140	100 - (-140) = 240	1
{80, -56, 11, -81}	80	-81	-80 - 80 = -161	0

```
    public int isEvenSpaced(int[] a)
    {
        //find maximum and minimum number

        int maxNum = a[0], minNum = a[0], rtnVal = 0;
        for(int i=0;i<a.Length;i++)
        {
            if(maxNum<a[i])
            {
                maxNum = a[i];
            }
            if(minNum>a[i])
            {
                minNum = a[i];
            }
        }
        if((maxNum-minNum)%2==0)
        {
            rtnVal = 1;
        }
        return rtnVal;
```

```
}
```

3. An *Sub array* is defined to be an array in which each element is greater than sum of all elements after that. See examples below: {13, 6, 3, 2} is a *Sub* array. Note that 13 > 2 + 3 + 6, 6 > 3 + 2, 3 > 2. {11, 5, 3, 2} is a NOT a *Sub* array. Note that 5 is not greater than 3 + 2.Write a function named *isSub* that returns 1 if its array argument is a *Sub* array, otherwise it returns 0.
If you are programming in Java or C#, the function signature is:
int *isSub* (int [ ] a) If you are programming in C or C++, the function signature is:
int *isSub* (int a[ ], int len) where *len* is the number of elements in the array.

```
public int isSub(int[] a)
      {
          int sum = 0;
          for (int i = a.Length-2; i >=0; i--)
          {
              for (int j = a.Length-1; j > i; j--)
              {
                  sum += a[j];
              }
              if (a[i] <= sum)
                  return 0;
              sum = 0;
          }
          return 1;

      }
```

Set-13

QUESTION 1. An array a is called ***paired*** if its even numbered elements (a[0], a[2], etc.) are odd and its odd numbered elements (a[1], a[3], etc.) are even. Write a function named *isPaired* that accepts an array of integers and returns 1 if the array is paired, otherwise it returns 0. Examples: {7, 2, 3, 6, 7} is paired since a[0], a[2] and a[4] are odd, a[1] and a[3] are even. {7, 15, 9, 2, 3} is not paired since a[1] is odd. {17, 6, 2, 4} is not paired since a[2] is even.
If you are programming in Java or C#, the function signature is
int isPaired(int[ ] a)
If you are programming in C or C++, the function signature is
int isPaired(int a[ ], int len)
where *len* is the number of elements in the array.

```
public int isPaired(int[] a)
      {
          for (int i = 0; i < a.Length; i++)
          {
              if ((i % 2 == 0 && a[i] % 2 == 0) || (i % 2 != 0 && a[i] % 2 != 0))
```

```
            {
                return 0;
            }
        }
        return 1;
    }
```

2. A Bunker array is defined to be an array in which at least one odd number is immediately followed by its square. So {4, 9, 6, 7, 49} is a Bunker array because the odd number 7 is immediately followed by 49. But {2, 4, 9, 3, 15, 21} is not a Bunker array because none of the odd numbers are immediately followed by its square.

Write a function named isBunkerArray that returns 1 if its array argument is a Bunker array, otherwise it returns 0.

If you are programming in Java or C#, the function signature is
int isBunkerArray(int [ ] a)

If you are programming in C or C++, the function signature is
int isBunkerArray(int a[ ], int len) where len is the number of elements in the array.

```
public static int isBunkerArray (int[] a)
    {
        for (int i = 0; i < a.Length - 1; i++)
        {
            Squre obj = new Squre();
            if (a[i] % 2 != 0)
            {
                if (obj.isSquare(a[i + 1]) == 1)
                    return 1;
            }
        }
        return 0;
    }
```

3. A Meera array is defined to be an array such that for all values n in the array, the value -n is not in the array. So {3, 5, -2} is a Meera array. But {8, 3, -8} is not a Meera array because forn=8, -n = -8 is in the array.

Write a function named isMeera that returns 1 if its array argument is a Meera array. Otherwise it returns 0.

If you are programming in Java or C#, the function signature is

int isMeera(int [ ] a)

If you are programming in C or C++, the function signature is
int isMeera(int a[ ], int len) where len is the number of elements in the array.

```
public static int isMeera(int[] a)
     {
         for (int i = 0; i < a.Length; i++)
         {
             if (a[i] > 0)
             {
                 for (int j = 0; j < a.Length; j++)
                 {
                     if (a[j] == -a[i])
                         return 0;
                 }
             }
         }
         return 1;
     }
```

Set-14

1. Write a function named maxDistance that returns the largest distance between two non -trivial factors of a number. For example, consider 1001 = 7*11*13. Its non-trivial factors are 7, 11 , 13, 77, 91, 143. Note that 1 and 1001 are trivial factors. maxDistance(1001) would return 136 because the largest distance between any two non-trivial factors is 136 (143- 7 = 136). As another example, maxDistance (8) would return 2 because the non-trivial factors of 8 are 2 and 4 and the largest distance between any two non-trivial factors is 2 (4 - 2 = 2). Also, maxDistance (7) would return -1 since 7 has no non-trivial factors. Further, maxDistance (49) would return 0 since 49 has only one nontrivial factor 7. Hence maxDistance (49) is 0 (7 - 7 = 0).

The function signature is

int maxDistance(int n)

Code:

```
public static int  maxDistance(int n)
     {
         int d = 0;
         int f = 0,flag=0;
         for (int i = 2; i < n; i++)
         {
             if (n % i == 0 )
             {
                 flag++;
                 if (i - f > d)
                 {
                     d = i - f;
                 }
                 if (f == 0)
                     f = i;
```

```
            }
        }
        if (flag == 0)
            return -1;
        if (flag == 1)
            return 0;
        return d;

    }
```

2. Write a function named maxOccurDigit that returns the digit that occur the most. If there is no such digit, it will return -1. For example maxOccurDigit(327277) would return 7 because 7 occurs three times in the number and all other digits occur less than three times. Other examples:
maxOccurDigit(33331) returns 3
maxOccurDigit(32326) returns -1
maxOccurDigit(5) returns 5
maxOccurDigit(-9895) returns 9

The function signature is
maxOccurDigit(int n)

```
public static int maxOccurDigit(int n)
        {
            if (n < 0)
            {
                n = -n;
            }
            int[] digitCount = new int[10];
            while (n != 0)
            {
                int digit = n % 10;
                n = n / 10;
                digitCount[digit] += 1;
            }
            int maxCount = digitCount[0];
            int maxOccurDigit = 0;
            for (int i = 1; i < digitCount.Length; i++)
            {
                if (maxCount < digitCount[i])
                {
                    maxCount = digitCount[i];
                    maxOccurDigit = i;
                }
            }
            for (int j = 0; j < digitCount.Length; j++)
            {
                if (j != maxOccurDigit && digitCount[j] == maxCount)
                    return -1;
            }
            return maxOccurDigit;
        }
```

3. A Bunker array is defined to be an array in which at least one odd number is immediately followed by its square. So {4, 9, 6, 7, 49} is a Bunker array because the odd number 7 is immediately followed by 49. But {2, 4, 9, 3, 15, 21} is not a Bunker array because none of the odd numbers are immediately followed by its square.

Write a function named isBunkerArray that returns 1 if its array argument is a Bunker array, otherwise it returns 0.

If you are programming in Java or C#, the function signature is
int isBunkerArray(int [ ] a)

If you are programming in C or C++, the function signature is
int isBunkerArray(int a[ ], int len) where len is the number of elements in the array.

```
public static int isBunkerArray(int[] a)
        {
            for (int i = 0; i < a.Length - 1; i++)
            {
                Squre obj = newSqure();
                if (a[i] % 2 != 0)
                {
                    if (obj.isSquare(a[i + 1]) == 1)
                        return 1;
                }
            }
            return 0;
        }
```

Set-15

1. A Meera array is defined to be an array such that for all values n in the array, the value -n is not in the array. So {3, 5, -2} is a Meera array. But {8, 3, -8} is not a Meera array because forn=8, -n = -8 is in the array.

Write a function named isMeera that returns 1 if its array argument is a Meera array. Otherwise it returns 0.

If you are programming in Java or C#, the function signature is

int isMeera(int [ ] a)

If you are programming in C or C++, the function signature is
int isMeera(int a[ ], int len) where len is the number of elements in the array.

```
public static int isMeera(int[] a)
        {
            for (int i = 0; i < a.Length; i++)
```

```
            {
                if (a[i] > 0)
                {
                    for (int j = 0; j < a.Length; j++)
                    {
                        if (a[j] == -a[i])
                            return 0;
                    }
                }
            }
            return 1;
        }
```

2. **Mode** is the most frequently appearing value. Write a function named *hasSingleMode* that takes an array argument and returns 1 if the mode value in its array argument occurs exactly once in the array, otherwise it returns 0. If you are writing in Java or C#, the function signature is *int hasSingleMode(int[ ]).*

If you are writing in C or C++, the function signature is
int hasSingleMode(int a[ ], intlen)
where len is the length of a.
Examples

Array elements	Mode values	Value returned	Comments
1, -29, 8, 5, -29, 6	-29	1	single mode
1, 2, 3, 4, 2, 4, 7	2, 4	0	no single mode
1, 2, 3, 4, 6	1, 2, 3, 4, 6	0	no single mode
7, 1, 2, 1, 7, 4, 2, 7,	7	1	single mode

```
    public static int hasSingleMode (int[] a)
        {
            int count = 0, maxcount = 0, mode = 0;
            for (int i = 0; i < a.Length; i++)
            {
                count = 0;
                for (int j = 0; j < a.Length; j++)
                {
                    if (a[i] == a[j])
                        count++;
                }
                if (count == maxcount)
                    mode++;
                if (count > maxcount)
                {
                    maxcount = count;
                    mode = 1;
                }

            }
            if (mode == maxcount)
                return 1;



            return 0;
```

```
}
```