There are three questions on this exam. You have 2 hours to complete it. Please indent your program so that it is easy for the grader to read.

1. Write a function named **largestAdjacentSum** that iterates through an array computing the sum of adjacent elements and returning the largest such sum. You may assume that the array has at least 2 elements.

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

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

Examples:

if a is	return
{1, 2, 3, 4}	7 because 3+4 is larger than either 1+2 or 2+3
{18, -12, 9, -10}	6 because 18-12 is larger than -12+9 or 9-10
{1,1,1,1,1,1,1,1}	2 because all adjacent pairs sum to 2
{1,1,1,1,2,1,1,1}	3 because 1+2 or 2+1 is the max sum of adjacent pairs

2. The number 198 has the property that 198 = 11 + 99 + 88, i.e., if each of its digits is concatenated twice and then summed, the result will be the original number. It turns out that 198 is the only number with this property. However, the property can be generalized so that each digit is concatenated n times and then summed. For example, 2997 = 222+999+999+777 and here each digit is concatenated three times. Write a function named **checkContenatedSum** that tests if a number has this generalized property.
The signature of the function is
int checkConcatenatedSum(int n, int catlen) where *n* is the number and *catlen* is the number of times to concatenate each digit before summing.

The function returns 1 if *n* is equal to the sum of each of its digits contenated *catlen* times. Otherwise, it returns 0. You may assume that n and catlen are greater than zero

Hint: Use integer and modulo 10 arithmetic to sequence through the digits of the argument.

Examples:

if n is	and catlen is	return	reason
198	2	1	because 198 == 11 + 99 + 88
198	3	0	because 198 != 111 + 999 + 888
2997	3	1	because 2997 == 222 + 999 + 999 + 777
2997	2	0	because 2997 != 22 + 99 + 99 + 77
13332	4	1	because 13332 = 1111 + 3333 + 3333 + 3333 + 2222
9	1	1	because 9 == 9

3. Define *an m-n sequenced array* to be an array that contains one or more occurrences of all the integers between m and n inclusive. Furthermore, the array must be in ascending order and contain only those integers. For example, {2, 2, 3, 4, 4, 4, 5} is a 2-5 sequenced array. The array {2, 2, 3, 5, 5, 5} is **not** a 2-5 sequenced array because it is missing a 4. The array {0, 2, 2, 3, 3} is **not** a 2-3 sequenced array because the 0 is out of range. And {1,1, 3, 2, 2, 4} is not a 1-4 sequenced array because it is not in ascending order.

Write a method named **isSequencedArray** that returns 1 if its argument is a m-n sequenced array, otherwise it returns 0.

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

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

You may assume that m<=n

Examples

if a is	**and m is**	**and n is**	**return**	**reason**
{1, 2, 3, 4, 5}	1	5	1	because the array contains all the numbers between 1 and 5 inclusive in ascending order and no other numbers.
{1, 3, 4, 2, 5}	1	5	0	because the array is not in ascending order.
{-5, -5, -4, -4, -4, -3, -3, -2, -2, -2}	-5	-2	1	because the array contains all the numbers between -5 and -2 inclusive in ascending order and no other numbers**. Note that duplicates are allowed.**
{0, 1, 2, 3, 4, 5}	1	5	0	because 0 is not in between 1 and 5 inclusive
{1, 2, 3, 4}	1	5	0	because there is no 5
{1, 2, 5}	1	5	0	because there is no 3 or 4
{5, 4, 3, 2, 1}	1	5	0	because the array does not start with a 1. Furthermore, it is not in ascending order.

There are three questions on this exam. You have 2 hours to complete it. **Please indent your programs so that it is easy for the grader to read.**

1. Write a function named **largestPrimeFactor** that will return the largest prime factor of a number. If the number is <=1 it should return 0. Recall that a prime number is a number > 1 that is divisible only by 1 and itself, e.g., 13 is prime but 14 is not.

The signature of the function is **int largestPrimeFactor(int n)**

Examples:

if n is	**return**	**because**
10	5	because the prime factors of 10 are 2 and 5 and 5 is the largest one.
6936	17	because the distinct prime factors of 6936 are 2, 3 and 17 and 17 is the largest
1	0	because n must be greater than 1

2. The fundamental theorem of arithmetic states that every natural number greater than 1 can be written as a unique product of prime numbers. So, for instance, 6936=2*2*2*3*17*17. Write a method named encodeNumber what will encode a number n as an array that contains the prime numbers that, when multipled together, will equal n. So encodeNumber(6936) would return the array {2, 2, 2, 3, 17, 17}. If the number is <= 1 the function should return null;

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

If you are programming in C or C++, the function signature is
int *encodeNumber(int n) and the last element of the returned array is 0.

Note that if you are programming in Java or C#, the returned array should be big enough to contain the prime factors **and no bigger**. If you are programming in C or C++ you will need one additional element to contain the terminating zero.

Hint: proceed as follows:

1. Compute the total number of prime factors including duplicates.

2. Allocate an array to hold the prime factors. **Do not hard-code the size of the returned array!!**
3. Populate the allocated array with the prime factors. The elements of the array when multiplied together should equal the number.

Examples

if n is	return	reason
2	{2}	because 2 is prime
6	{2, 3}	because 6 = 2*3 and 2 and 3 are prime.
14	{2, 7}	because 14=2*7 and 2 and 7 are prime numbers.
24	{2, 2, 2, 3}	because 24 = 2*2*2*3 and 2 and 3 are prime
1200	{2, 2, 2, 2, 3, 5, 5}	because 1200=2*2*2*2*3*5*5 and those are all prime
1	null	because n must be greater than 1
-18	null	because n must be greater than 1

3. Consider a simple pattern matching language that matches arrays of integers. A pattern is an array of integers. An array matches a pattern if it contains sequences of the pattern elements in the same order as they appear in the pattern. So for example, the array {1, 1, 1, 2, 2, 1, 1, 3} matches the pattern {1, 2, 1, 3} as follows:

{**1, 1, 1**, 2, 2, 1, 1, 3} {**1**, 2, 1, 3} (first 1 of pattern matches three 1s in array)
{1, 1, 1, **2, 2**, 1, 1, 3} {1, **2**, 1, 3} (next element of pattern matches two 2s in array)
{1, 1, 1, 2, 2, **1, 1**, 3} {1, 2, **1**, 3} (next element of pattern matches two 1s in array)
{1, 1, 1, 2, 2, 1, 1, **3**} {1, 2, 1, **3**} (last element of pattern matches one 3 in array)

The pattern must be completely matched, i.e. the last element of the array must be matched by the last element of the pattern.

Here is an incomplete function that does this pattern matching. It returns 1 if the pattern matches the array, otherwise it returns 0.

```
static int matchPattern(int[] a, int len, int[] pattern, int patternLen) {
// len is the number of elements in the array a, patternLen is the number of elements in the pattern.
  int i=0; // index into a
  int k=0; // index into pattern
  int matches = 0;  // how many times current pattern character has been matched so far
  for (i=0; i<len; i++)  {
    if (a[i] == pattern[k])
      matches++;   // current pattern character was matched
    else if (matches == 0 || k == patternLen-1)
      return 0; // if pattern[k] was never matched (matches==0) or at end of pattern (k==patternLen-1)
    else  // advance to next pattern character {
       !!You write this code!!
    } // end of else
  } // end of for

  // return 1 if at end of array a (i==len) and also at end of pattern (k==patternLen-1)
  if (i==len && k==patternLen-1) return 1; else return 0;
}
```

Please finish this function by writing the code for the last else statement. Your answer just has to include this code, you do not have to write the entire function.

Hint: You need at least 4 statements (one of them an if statement)

Examples

if a is	and pattern is	return	reason
{1, 1, 1, 1, 1}	{1}	1	because all elements of the array match the pattern element 1
{1}	{1}	1	because all elements of the array match the pattern element 1
{1, 1, 2, 2, 2, 2}	{1, 2}	1	because the first two 1s of the array are matched by the first pattern element, last four 2s of array are matched by the last pattern element
{1, 2, 3}	{1, 2}	0	because the 3 in the array is not in the pattern.
{1, 2}	{1, 2, 3}	0	because the 3 in the pattern is not in the array
{1, 1, 2, 2, 2, 2, 3}	{1, 3}	0	because at least one 3 must appear after the sequence of 1s.
{1, 1, 1, 1}	{1, 2}	0	because the array ends without matching the pattern element 2.
{1, 1, 1, 1, 2, 2, 3, 3}	{1, 2}	0	because the element 3 of the array is not matched
{1, 1, 10, 4, 4, 3}	{1, 4, 3}	0	because the 10 element is not matched by the 4 pattern element. Be sure your code handles this situation correctly!

There are three questions on this exam. You have 2 hours to complete it. Please indent your program so that it is easy for the grader to read.

1. Define the **n-based integer rounding** of an integer k to be the nearest multiple of n to k. If two multiples of n are equidistant use the greater one. For example

the 4-based rounding of **5** is 4 because **5** is closer to 4 than it is to 8,
the 5-based rounding of **5** is 5 because **5** is closer to 5 that it is to 10,
the 4-based rounding of **6** is 8 because **6** is equidistant from 4 and 8, so the greater one is used,
the 13-based rounding of **9** is 13, because **9** is closer to 13 than it is to 0,

Write a function named **doIntegerBasedRounding** that takes an integer array and rounds all its positive elements using n-based integer rounding.

A negative element of the array is **not** modified and if n <=0, **no** elements of the array are modified.
Finally you may assume that the array has at least two elements.

Hint: In integer arithmetic, (6/4) * 4 = 4

If you are programming in Java or C#, the function signature is
void doIntegerBasedRounding(int[ ] a, int n) where n is used to do the rounding

If you are programming in C or C++, the function signature is
void doIntegerBasedRounding(int a[ ], int n, int len) where n is used to do the rounding and len is the number of elements in the array a.

Examples

if a is	and n is	then a becomes	reason
{1, 2, 3, 4, 5}	2	{2, 2, 4, 4, 6}	because the 2-based rounding of **1** is 2, the 2-based rounding of **2** is 2, the 2-based rounding of **3** is 4, the

			2-based rounding of **4** is 4, and the 2-based rounding of **5** is 6.
{1, 2, 3, 4, 5}	3	{0, 3, 3, 3, 6}	because the 3-based rounding of **1** is 0, the 3-based roundings of **2, 3, 4** are all 3, and the 3-based rounding of **5** is 6.
{1, 2, 3, 4, 5}	-3	{1, 2, 3, 4, 5}	because the array is not changed if n <= 0.
{-1, -2, -3, -4, -5}	3	{-1, -2, -3, -4, -5}	because negative numbers are not rounded
{-18, 1, 2, 3, 4, 5}	4	{-18, 0, 4, 4, 4, 4}	because -18 is negative and hence is not modified, the 4-based rounding of **1** is 0, and the 4-based roundings of **2, 3, 4, 5** are all 4.
{1, 2, 3, 4, 5}	5	{0, 0, 5, 5, 5}	
{1, 2, 3, 4, 5}	100	{0, 0, 0, 0, 0}	

2. A number n>0 is called **cube-powerful** if it is equal to the sum of the cubes of its digits.

Write a function named **isCubePowerful** that returns 1 if its argument is cube-powerful; otherwise it returns 0.

The function prototype is
int isCubePowerful(int n);

Hint: use modulo 10 arithmetic to get the digits of the number.

Examples:

if n is	**return**	**because**
153	1	because 153 = $1^3 + 5^3 + 3^3$
370	1	because 370 = $3^3 + 7^3 + 0^3$
371	1	because 371 = $3^3 + 7^3 + 1^3$
407	1	because 407 = $4^3 + 0^3 + 7^3$
87	0	because 87 != $8^3 + 7^3$
0	0	because n must be greater than 0.
-81	0	because n must be greater than 0.

3. A number can be encoded as an integer array as follows. The first element of the array is any number and if it is negative then the encoded number is negative. Each digit of the number is the absolute value of the difference of two adjacent elements of the array. The most significant digit of the number is the absolute value of the difference of the first two elements of the array. For example, the array {2, -3, -2, 6, 9, 18} encodes the number 51839 because

- 5 is abs(2 - (-3))
- 1 is abs(-3 - (-2))
- 8 is abs(-2 - 6)
- 3 is abs(6-9)
- 9 is abs(9-18)

The number is positive because the first element of the array is >= 0.

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

If you are programming in C or C++, the function prototype is
int decodeArray(int a[ ], int len) where len is the length of array a;

You may assume that the encoded array is correct, i.e., the absolute value of the difference of any two adjacent elements is between 0 and 9 inclusive and the array has at least two elements.

Examples

a is	**then function returns**	**reason**
{0, -3, 0, -4, 0}	3344	because abs(0-(-3)=3, abs(-3-0)=3, abs(0-(-4))=4, abs(-4-0)=4
{-1, 5, 8, 17, 15}	-6392	because abs(-1-5)=6, abs(5-8)=3, abs(8-17)=9, abs(17-15)=2; the number is negative because the first element of the array is negative
{1, 5, 8, 17, 15}	4392	because abs(1-5)=4, remaining digits are the same as previous example; the number is positive because the first element of the array is >=0.
{111, 115, 118, 127, 125}	4392	because abs(111-115)=4, abs(115-118)=3, abs(118-127)=9, abs(127-125)=2; the number is positive because the first

		element of the array is >=0.
{1, 1}	0	because abs(1-1) = 0

****

There are three questions on this exam. You have 2 hours to complete it.

1. An array is **zero-plentiful** if it contains at least one 0 and every sequence of 0s is of length at least 4.

Write a method named **isZeroPlentiful** which returns the number of zero sequences if its array argument is zero-plentiful, otherwise it returns 0.

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

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

Examples

a is	**then function returns**	**reason**
{0, 0, 0, 0, 0}1	1	because there is one sequence of 0s and its length >= 4.
{1, 2, 0, 0, 0, 0, 2, -18, 0, 0, 0, 0, 0, 12}1	2	because there are two sequences of 0s and both have lengths >= 4.
{0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0}1	3	because three are three sequences of zeros and all have length >=4
{1, 2, 3, 4}1	0	because there must be at least one 0.
{1, 0, 0, 0, 2, 0, 0, 0, 0}	0	because there is a sequence of zeros whose length is less < 4.
{0}	0	because there is a sequence of zeroes whose length is < 4.

{}	0	because there must be at least one 0.

2. A number is called **digit-increasing** if it is equal to n + nn + nnn + ... for some digit n between 1 and 9. For example 24 is digit-increasing because it equals 2 + 22 (here n = 2)

Write a function called **isDigitIncreasing** that returns 1 if its argument is digit-increasing otherwise, it returns 0.

The signature of the method is
int isDigitIncreasing(int n)

Examples

if n is	**then function returns**	**reason**
7	1	because 7 = 7 (here n is 7)
36	1	because 36 = 3 + 33
984	1	because 984 = 8 + 88 + 888
7404	1	because 7404 = 6 + 66 + 666 + 6666

3. An integer number can be encoded as an array as follows. Each digit n of the number is represented by n zeros followed by a 1. So the digit 5 is represented by 0, 0, 0, 0, 0, 1. The encodings of each digit of a number are combined to form the encoding of the number. So the number 1234 is encoded as the array {0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1}. The first 0, 1 is contributed by the digit 1, the next 0, 0, 1 is contributed by the digit 2, and so on.

There is one other encoding rule: if the number is negative, the first element of the encoded array must be -1, so -201 is encoded as {-1, 0, 0, 1, 1, 0, 1}. Note that the 0 digit is represented by no zeros, i.e. there are two consecutive ones!

Write a method named **decodeArray** that takes an encoded array and decodes it to return the number.

You may assume that the input array is a legal encoded array, i.e., that -1 will only appear as the first element, all elements are either 0, 1 or -1 and that the last element is 1.

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

If you are programming in C or C++, the function prototype is
int decodeArray(int a[ ], int len);

Examples

a is	**then function returns**	**reason**
{1}	0	because the digit 0 is represented by no zeros followed by a one.
{0, 1}	1	because the digit 1 is represented by one zero followed by a one.
{-1, 0, 1}	-1	because the encoding of a negative number begins with a -1 followed by the encoding of the absolute value of the number.
{0, 1, 1, 1, 1, 1, 0, 1}	100001	because the encoding of the first 1 is 0, 1, the encoding of each of the four 0s is just a 1 and the encoding of the last 1 is 0, 1.
{0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1}	999	because each 9 digit is encoded as 0,0,0,0,0,0,0,0,0,1.

This exam consists of three questions. You have two hours in which to complete it.

1. An **onion** array is an array that satisfies the following condition for all values of j and k:
if j>=0 and k>=0 and j+k=length of array and j!=k then a[j]+a[k] <= 10

Write a function named **isOnionArray** that returns 1 if its array argument is an onion array and returns 0 if it is not.

Your solution must not use a nested loop (i.e., a loop executed from inside another loop). Furthermore, once you determine that the array is not an onion array your function must return 0; no wasted loops cycles please!

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

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

Examples

a is	then function returns	reason
{1, 2, 19, 4, 5}	1	because 1+5 <= 10, 2+4 <=10
{1, 2, 3, 4, 15}	0	because 1+15 > 10
{1, 3, 9, 8}	0	because 3+9 > 10
{2}	1	because there is no j, k where a[j]+a[k] > 10 and j+k=length of array and j!=k
{}	1	because there is no j, k where a[j]+a[k] > 10 and j+k=length of array and j!=k
{-2, 5, 0, 5, 12}	1	because -2+12 <= 10 and 5+5 <= 10

2. A number n is called **prime happy** if there is at least one prime less than n and the sum of all primes less than n is evenly divisible by n.

Recall that a prime number is an integer > 1 which has only two integer factors, 1 and itself

The function prototype is int **isPrimeHappy(int n)**;

Examples:

if n is	return	because
5	1	because 2 and 3 are the primes less than 5, their sum is 5 and 5 evenly divides 5.
25	1	because 2, 3, 5, 7, 11, 13, 17, 19, 23 are the primes less than 25, their sum is 100 and 25 evenly divides 100
32	1	because 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31 are the primes less than 32, their sum is 160 and 32 evenly divides 160
8	0	because 2, 3, 5, 7 are the primes less than 8, their sum is 17 and 8 does not evenly divide 17.
2	0	because there are no primes less than 2.

3. An integer number can be encoded as an array as follows. Each digit n of the number is represented by n zeros followed by a 1. So the digit 5 is represented by 0, 0, 0, 0, 0, 1. The encodings of each digit of a number are combined to form the encoding of the number. So the number 1234 is encoded as the array {0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1}. The first 0, 1 is contributed by the digit 1, the next 0, 0, 1 is contributed by the digit 2, and so on. There is one other encoding rule: if the number is negative, the first element of the encoded array must be -1, so -201 is encoded as {-1, 0, 0, 1, 1, 0, 1}. Note that the 0 digit is represented by no zeros, i.e. there are two consecutive ones!

use array list

Write a method named **encodeArray** that takes an integer as an argument and returns the encoded array.

If you are programming in Java or C#, the function prototype is
int[] encodeArray(int n)

If you are programming in C or C++, the function prototype is
int * encodeArray(int n);

Hints

Use modulo 10 arithmetic to get digits of number
Make one pass through the digits of the number to compute the size of the encoded array.
Make a second pass through the digits of the number to set elements of the encoded array to 1.

n is	then function returns	reason
0	{1}	because the digit 0 is represented by no zeros and the representation of each digit ends in one.
1	{0, 1}	because the digit 1 is represented by one zero and the representation of each digit ends in one.
-1	{-1, 0, 1}	because the encoding of a negative number begins with a -1 followed by the encoding of the absolute value of the number.
100001	{0, 1, 1, 1, 1, 1, 0, 1}	because the encoding of the first 1 is 0, 1, the encoding of each of the four 0s is just a 1 and the encoding of the last 1 is 0, 1.
999	0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1	because each 9 digit is encoded as 0,0,0,0,0,0,0,0,1.

Examples

1. An array is called **systematically increasing** if it consists of increasing sequences of the numbers from 1 to n.

The first six (there are over 65,000 of them) systematically increasing arrays are:

{1}
{1, 1, 2}
{1, 1, 2, 1, 2, 3}
{1, 1, 2, 1, 2, 3, 1, 2, 3, 4}

{1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5}
{1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6}

Write a function named **isSystematicallyIncreasing** which returns 1 if its array argument is systematically increasing. Otherwise it returns 0.

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

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

Examples

a is	then function returns	reason
{1}	1	because 1 is a sequence from 1 to 1 and is the only sequence.
{1, 2, 1, 2, 3}	0	because it is missing the sequence from 1 to 1.
{1, 1, 3}	0	because {1, 3} is not a sequence from 1 to n for any n.
{1, 2, 1, 2, 1, 2}	0	because it contains more than one sequence from 1 to 2.
{1, 2, 3, 1, 2, 1}	0	because it is "backwards", i.e., the sequences from 1 to n are not ordered by increasing value of n
{1, 1, 2, 3}	0	because the sequence {1, 2} is missing (it should precede {1, 2, 3})

2. A positive, non-zero number n is a **factorial prime** if it is equal to factorial(n) + 1 for some n and it is prime. Recall that factorial(n) is equal to 1 * 2 * ... * n-1 * n. If you understand recursion, the recursive definition is
factorial(1) = 1;
factorial(n) = n*factorial(n-1).
For example, factorial(5) = 1*2*3*4*5 = 120.

Recall that a prime number is a natural number which has exactly two distinct natural number divisors: 1 and itself.

Write a method named **isFactorialPrime** which returns 1 if its argument is a factorial prime number, otherwise it returns 0.

The signature of the method is
int isFactorialPrime(int n)

Examples

if n is	then function returns	reason
2	1	because 2 is prime and is equal to factorial(1) + 1
3	1	because 3 is prime and is equal to factorial(2) + 1

7	1	because 7 prime and is equal to factorial(3) + 1
8	0	because 8 is not prime
11	0	because 11 does not equal factorial(n) + 1 for any n (factorial(3)=6, factorial(4)=24)
721	0	because 721 is not prime (its factors are 7 and 103)

3. Write a function named **largestDifferenceOfEvens** which returns the largest difference between even valued elements of its array argument. For example largestDifferenceOfEvens(new int[ ]{-2, 3, 4, 9}) returns 6 = (4 - (-2)). If there are fewer than 2 even numbers in the array, largestDifferenceOfEvens should return -1.

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

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

Examples

a is	then function returns	reason
{1, 3, 5, 9}	-1	because there are no even numbers
{1, 18, 5, 7, 33}	-1	because there is only one even number (18)
{[2, 2, 2, 2]}	0	because 2-2 == 0
{1, 2, 1, 2, 1, 4, 1, 6, 4}	4	because 6 - 2 == 4

1. A **hodder number** is one that is prime and is equal to 2^j-1 for some j. For example, 31 is a hodder number because 31 is prime and is equal to 2^5-1 (in this case j = 5). The first 4 hodder numbers are 3, 7, 31, 127

Write a function with signature **int isHodder(int n)** that returns 1 if n is a hodder number, otherwise it returns 0.

Recall that a prime number is a whole number greater than 1 that has only two whole number factors, itself and 1.

2. One word is an **anagram** of another word if it is a rearrangement of all the letters of the second word. For example, the character arrays {'s', 'i', 't'} and {'i', 't', 's'} represent words that are anagrams of one another because "its" is a rearrangement of all the letters of "sit" and vice versa. Write a function that accepts two character arrays and returns 1 if they are anagrams of one another, otherwise it returns 0. For simplicity, if the two input character arrays are equal, you may consider them to be anagrams.

If you are programming in Java or C#, the function signature is:
int areAnagrams(char [ ] a1, char [ ] a2)

If you are programming in C or C++, the function signature is
int areAnagrams(a1 char[ ], a2 char[ ], int len) where len is the length of a1 and a2.

Hint: Please note that "pool" is not an anagram of "poll" even though they use the same letters. Please be sure that your function returns 0 if given these two words! You can use another array to keep track of each letter that is found so that you don't count the same letter twice (e.g., the attempt to find the second "o" of "pool" in "poll" should fail.)

Hint: do not modify either a1 or a2, i.e., your function should have no side effects! If your algorithm requires modification of either of these arrays, you must work with a copy of the array and modify the copy!

Examples

if input arrays are	return
{'s', 'i', 't'} and {'i', 't', 's'}	1
{'s', 'i', 't'} and {'i', 'd', 's'}	0
{'b', 'i', 'g'} and {'b', 'i', 't'}	0
{'b', 'o', 'g'} and {'b', 'o', 'o'}	0
{} and {}	1
{'b', 'i', 'g'} and {'b', 'i', 'g'}	1

3. The Fibonacci sequence of numbers is 1, 1, 2, 3, 5, 8, 13, 21, 34, ... The first and second numbers are 1 and after that $n_i = n_{i-2} + n_{i-1}$, e.g., 34 = 13 + 21. A number in the sequence is called a Fibonacci number. Write a method with signature **int closestFibonacci(int n)** which returns the largest Fibonacci number that is less than or equal to its argument. For example, closestFibonacci(13) returns 8 because 8 is the largest Fibonacci number less than 13 and closestFibonacci(33) returns 21 because 21 is the largest Fibonacci number that is <= 33. closestFibonacci(34) should return 34. If the argument is less than 1 return 0. Your solution must **not** use recursion because unless you cache the Fibonacci numbers as you find them, the recursive solution recomputes the same Fibonacci number many times.

1. A number n is **vesuvian** if it is the sum of two different pairs of squares. For example, 50 is vesuvian because 50 == 25 + 25 and 1 + 49. The numbers 65 (1+64, 16+49) and 85 (4+81, 36+49) are also vesuvian. 789 of the first 10,000 integers are vesuvian.

Write a function named **isVesuvian** that returns 1 if its parameter is a vesuvian number, otherwise it returns 0. Hint: be sure to verify that your function detects that 50 is a vesuvian number!

2. Define an array to be **one-balanced** if begins with zero or more 1s followed by zero or more non-1s and concludes with zero or more 1s. Write a function named **isOneBalanced** that returns 1 if its array argument is one-balanced, otherwise it returns 0.

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

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

Examples

if a is	**then function returns**	**reason**
{1, 1, 1, 2, 3, -18, 45, 1}	1	because it begins with **three** 1s, followed by **four** non-1s and ends with **one** 1 and 3+1 == 4
{1, 1, 1, 2, 3, -18, 45, 1, **0**}	0	because the 0 starts another sequence of non-1s. There can be only one sequence of non-1s.
{1, 1, **2, 3**, 1, **-18, 26**, 1}	0	because there are two sequences of non-1s ({2, 3} and {-18, 26}
{}	1	because 0 (# of beginning 1s) + 0 (# of ending 1s) = 0 (# of non-1s)
{3, 4, 1, 1}	1	because 0 (# of beginning 1s) + 2 (# of ending 1s) = 2 (# of non-1s)
{1, 1, 3, 4}	1	because 2 (# of beginning 1s) + 0 (# of ending 1s) = 2 (# of non-1s)
{3, 3, 3, 3, 3, 3}	0	because 0 (# of beginning 1s) + 0 (# of ending 1s) != 6 (# of non-1s)
{1, 1, 1, 1, 1, 1}	0	because 6 (# of beginning 1s) + 0 (# of ending 1s) != 0 (# of non-1s)

3. The Fibonacci sequence of numbers is 1, 1, 2, 3, 5, 8, 13, 21, 34, ... The first and second numbers are 1 and after that $n_i = n_{i-2} + n_{i-1}$, e.g., 34 = 13 + 21. Write a method with signature **int isFibonacci(int n)** which returns 1 if its argument is a number in the Fibonacci sequence, otherwise it returns 0. For example, isFibonacci(13) returns a 1 and isFibonacci(27) returns a 0. Your solution must **not** use recursion because unless you cache the Fibonacci numbers as you find them, the recursive solution recomputes the same Fibonacci number many times.

1. A number n is **triangular** if n == 1 + 2 +...+j for some j. Write a function
int isTriangular(int n)
that returns 1 if n is a triangular number, otherwise it returns 0. The first 4 triangular numbers are 1 (j=1), 3 (j=2), 6, (j=3), 10 (j=4).

2. Define an array to be a **Mercurial** array if a 3 does not occur between any two 1s. Write a function named **isMercurial** that returns 1 if its array argument is a Mercurial array, otherwise it returns 0.

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

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

Hint: if you encounter a 3 that is preceded by a 1, then there can be no more 1s in the array after the 3.

Examples

a is	**then function returns**	**reason**
{**1**, 2, 10, **3**, 15, **1**, 2, 2}	0	because 3 occurs after a 1 (a[0]) and before another 1 (a[5])
{5, 2, 10, **3**, 15, **1**, 2, 2}	1	because the 3 is not between two 1s.
{**1**, 2, 10, **3**, 15, 16, 2, 2}	1	because the 3 is not between two 1s.
{3, 2, 18, **1**, 0, **3**, -11, **1**, 3}	0	because a[5] is a 3 and is between a[3] and a[7] which are both 1s.
{2, **3, 1, 1**, 18}	1	because there are no instances of a 3 that is between two 1s
{8, 2, **1, 1**, 18, **3**, 5}	1	because there are no instances of a 3 that is between two 1s
{3, 3, 3, 3, 3, 3}	1	because there are no instances of a 3 that is between two 1s
{1}	1	because there are no instances of a 3 that is between two 1s
{}	1	because there are no instances of a 3 that is between two 1s

3. An array is defined to be a **235 array** if the number of elements divisible by 2 plus the number of elements divisible by 3 plus the number of elements divisible by 5 plus the number of elements not divisible by 2, 3, or 5 is equal to the number of elements of the array. Write a method named **is123Array** that returns 1 if its array argument is a 235 array, otherwise it returns 0.

If you are writing in Java or C#, the function signature is
int is235Array(int[ ] a)

If you are writing in C or C++, the function signature is
int is235Array(int a[ ], int len) where len is the length of a

Hint: remember that a number can be divisible by more than one number

Examples

In the following: <**a, b, c, d**> means that the array has **a** elements that are divisible by 2, **b** elements that are divisible by 3, **c** elements that are divisible by 5 and **d** elements that are not divisible by 2, 3, or 5.

if a is	**return**	**reason**
{2, 3, 5, 7, 11}	1	because one element is divisible by 2 (a[0]), one is divisible by 3 (a[1]), one is divisible by 5 (a[2]) and two are not divisible by 2, 3, or 5 (a[3] and a[4]). So we have <1, 1, 1, 2> and 1+1+1+2 == the number of elements in the array.
{2, 3, 6, 7, 11}	0	because two elements are divisible by 2 (a[0] and a[2]), two are divisible by 3 (a[1] and a[2]), none are divisible by 5 and two are not divisible by 2, 3, or 5 (a[3] and a[4]). So we have <2, 2, 0, 2> and 2 + 2 + 0 + 2 == 6 **!=** the number of elements in the array.
{2, 3, 4, 5, 6, 7, 8, 9, 10}	0	because <5, 3, 2, 1> and 5 + 3 + 2 + 1 == 11 **!=** the number of elements in the array.
{2, 4, 8, 16, 32}	1	because <5, 0, 0, 0> and 5 + 0 + 0 + 0 == 5 == the number of elements in the array.
{3, 9, 27, 7, 1, 1, 1, 1, 1}	1	because <0, 3, 0, 6> and 0 + 3 + 0 + 6 == 9 == the number of elements in the array.
{7, 11, 77, 49}	1	because <0, 0, 0, 4> and 0 + 0 + 0 + 4 == 4 == the number of elements in the array.
{2}	1	because <1, 0, 0, 0> and 1 + 0 + 0 + 0 == 1 == the number of elements in the array.
{}	1	because <0, 0, 0, 0> and 0 + 0 + 0 + 0 == 0 == the number of elements in the array.
{7, 2, 7, 2, 7, 2, 7, 2, 3, 7, 7}	1	because <4, 1, 0, 6> and 4 + 1 + 0 + 6 == 11 == the number of elements in the array.

1. Write a method named **computeHMS** that computes the number of hours, minutes and seconds in a given number of seconds.

If you are programming in Java or C#, the method signature is
int[] computeHMS(int seconds);

If you are programming in C or C++, the method signature is
int * computeHMS(int seconds);

The returned array has 3 elements; arr[0] is the hours, arr[1] is the minutes and arr[2] is the seconds contained within the seconds argument.

Recall that there are 3600 seconds in an hour and 60 seconds in a minute. You may assume that the numbers of seconds is non-negative.

Examples

If seconds is	**then function returns**	**reason**
3735	{1, 2, 15}	because 3735 = **1***3600 + **2***60 +**15**. In other words, 3,735 is the number of seconds in 1 hour 2 minutes and 15 seconds
380	{0, 6, 20}	because 380 = **0***3600 + **6***60 + **20**
3650	{1, 0, 50}	because 3650 = **1***3600 + **0***60 + **50**
55	{0, 0, 55}	because 55 = **0***3600 + **0***60 + **55**
0	{0, 0, 0}	because 0 = **0***3600 + **0***60 + **0**

2. Define an array to be a **Martian array** if the number of 1s is greater than the number of 2s and no two adjacent elements are equal. Write a function named isMartian that returns 1 if its argument is a Martian array; otherwise it returns 0.

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

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

There are two additional requirements.

1. You should return 0 as soon as it is known that the array is not a Martian array; continuing to analyze the array would be a waste of CPU cycles.

2. There should be exactly one loop in your solution.

Examples

a is	**then function returns**	**reason**
{1, 3}	1	There is one 1 and zero 2s, hence the number of 1s is greater than the number of 2s. Also, no adjacent elements have the same value (1 does not equal 3)
{1, 2, 1, 2, 1, 2, 1, 2, 1}	1	There are five 1s and four 2s, hence the number of 1s is greater than the number of 2s. Also, no two adjacent elements have the same value.

{1, 3, 2}	0	There is one 1 and one 2, hence the number of 1s is not greater than the number of 2s.
{1, 3**, 2, 2**, 1, 5, 1, 5}	0	There are two 2s adjacent to each other.
{1, 2, **-18, -18**, 1, 2}	0	The two -18s are adjacent to one another. Note that the number of 1s is not greater than than the number of 2s but your method should return 0 before determining that! (See the additional requirements above.)
{}	0	There are zero 1s and zero 2s hence the number of 1s is not greater than the number of 2s.
{1}	1	There is one 1 and zero 2s hence the number of 1s is greater than the number of 2s. Also since there is only one element, there cannot be adjacent elements with the same value.
{2}	0	There are zero 1s and one 2 hence the number of 1s is not greater than the number of 2s.

~~Hint: Make sure that your solution does not exceed the boundaries of the array!~~

3. An array is defined to be **paired-N** if it contains two distinct elements that sum to N for some specified value of N and the indexes of those elements also sum to N. Write a function named **isPairedN** that returns 1 if its array parameter is a paired-N array, otherwise it returns 0. The value of N is passed as the second parameter.
If you are writing in Java or C#, the function signature is
int isPairedN(int[ ] a, int n)

If you are writing in C or C++, the function signature is
int isPairedN(int a[ ], int n, int len) where len is the length of a

There are two additional requirements.

1. Once you know the array is paired-N, you should return 1. No wasted loop iterations please.

2. Do not enter the loop unless you have to. You should test the length of the array and the value of n to determine whether the array could possibly be a paired-N array. If the tests indicate no, return 0 before entering the loop.

Examples

if a is	and n is	return	reason
{1, 4, 1, 4, 5, 6}	5	1	because a[**2**] + a[**3**] == 5 and **2**+**3**==5. In other words, the sum of the values is equal to the sum of the corresponding indexes and both are equal to n (5 in this case).
{1, 4, 1, 4, 5, 6}	6	1	because a[**2**] + a[**4**] == 6 and **2**+**4**==6
{0, 1, 2, 3, 4, 5, 6, 7, 8}	6	1	because a[**1**]+a[**5**]==6 and **1**+**5**==6
{1, 4, 1}	5	0	because although a[**0**] + a[**1**] == 5, **0**+**1** != 5; and although a[**1**]+a[**2**]==5, **1**+**2** != 5
{8, 8, 8, 8, 7, 7, 7}	15	0	because there are several ways to get the values to sum to 15 but there is no way to get the corresponding indexes to sum to 15.

{8, -8, 8, 8, 7, 7, -7}	-15	0	because although a[**1**]+a[**6**]==-15, **1**+**6**!=-15
{3}	3	0	because the array has only one element
{}	0	0	because the array has no elements

This exam tests very basic programming skills and hence will be graded strictly. However, simple syntax errors will be forgiven. The following examples gives you an idea of how the exam will be graded.

Sample problem: Write a method int allEven(int a[ ], int len) that returns 1 if all elements of the array a are even, otherwise it returns 0. Assume that the array has at least one element.

Solution 1:

```
int allEven (int a[ ], int len)
{
	int result = 1;
	for (int i=0; i<len && result==1; i++)
	{
		if (a[i] % 2 == 1)
			result = 0;  // exit loop, found a non-even element
	}

	return result;
}
```

Grading result: **Correct**; full marks. Will also accept breaking or returning from loop.

Solution 2:

```
static int allEven (int a[ ], int len)
{
	int result = 1;
	for (int i=0; i<len; i++)
	{
		if (a[i] % 2 == 1)
			result = 0;  // found non-even element
	}

	return result;
}
```

Grading result: **Correct, but inefficient**; marks will be deducted because program continues to loop even though it is known that the result is 0.

Solution 3

```
static int allEven (int a[ ], int len)
{
	int result = 1;
	for (int i=0; i<len; i++)
	{
		if (a[i] % 2 == 1)
			result = 0;
		else
			result = 1;
	}

	return result;
```

}
Grading result: **Incorrect**; no marks. Program returns status of the last element of the array.

1. Define an array to be **n-primeable** if for a given n, all elements of the array when incremented by n are prime. Recall that a prime number is a number that has no factors except 1 and itself. Write a method named **isNPrimeable** that has an array parameter and an integer parameter that defines n; the method returns 1 if its array parameter is n-primeable; otherwise it returns 0.

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

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

Examples

If a is	and n is	then function returns	reason
{5, 15, 27}	**2**	1	5+***2***=7 is prime, and 15+***2***=17 is prime, and 27+***2***=29 is prime
{5, 15, 27}	**3**	0	5+***3***=8 is **not** prime
{5, 15, 26}	**2**	0	26+***2***=28 is **not** prime
{1, 1, 1, 1, 1, 1, 1}	**4**	1	1+***4***=5 is prime. This obviously holds for all elements in the array
{}	**2**	1	Since there are no elements in the array, there cannot exist one that is non-prime when 2 is added to it.

2. Define an array to be a **121 array** if all elements are either 1 or 2 and the array begins with one or more 1s followed by a one or more 2s and then ends with the same number of 1s that it begins with. Write a method named **is121Array** that returns 1 if its array argument is a 121 array, otherwise, it returns 0.

If you are programming in Java or C#, the function signature is
int is121Array(int[ ] a)

If you are programming in C or C++, the function signature is
int is121Array(int a[ ], int len) where len is the number of elements in the array a.

Examples

a is	then function returns	reason
{1, 2, 1}	1	because the same number of 1s are at the beginning and end of the array and there is at least one 2 in between them.
{1, 1, 2, 2, 2, 1, 1}	1	because the same number of 1s are at the beginning and end of the array and there is at least one 2 in between them.
{1, 1, 2, 2, 2, 1, 1, 1}	0	Because the number of 1s at the end does not equal the number of 1s at the beginning.
{1, 1, 1, 2, 2, 2, 1, 1}	0	Because the number of 1s at the end does not equal the number of 1s at the beginning.
{1, 1, 1, 2, 2, 2, 1, 1, 1, 3}	0	Because the array contains a number other than 1 and 2.

{1, 1, 1, 1, 1, 1}	0	Because the array does not contain any 2s
{2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 1}	0	Because the first element of the array is not a 1.
{1, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2}	0	Because the last element of the array is not a 1.
{2, 2, 2}	0	Because there are no 1s in the array.

Hint: Walk through your solution with **all** the above test cases!

3. Write a method named **pairwiseSum** that has an array with an even number of elements as a parameter and returns an array that contains the pairwise sums of the elements of its parameter array.

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

If you are writing in C or C++, the function signature is
int * pairwiseSum(int a[ ], int len) where len is the length of a

The method returns null if

1. The array has no elements
2. The array has an odd number of elements

Otherwise, the method returns an array with arrayLength/2 elements. Each element of the returned array is the sum of successive pairs of elements of the original array. See examples for more details.

Examples

if a is	**return**	**reason**
{2, 1, 18, -5}	{3, 13}	because 2+1=3, 18+-5=13. Note that there are exactly 2 elements in the returned array. You will lose full marks for this question if you return {3, 13, **0, 0**}!
{2, 1, 18, -5, -5, -15, 0, 0, 1, -1}	{3, 13, -20, 0, 0}	because 2+1=3, 18+-5=13, -5+-15=-20, 0+0=0, 1+-1=0. Note that there are exactly 5 elements in the returned array. You will lose full marks for this question if you return {3, 13, -20, 0, 0, **0, 0, 0, 0, 0**}!
{2, 1, 18}	null	because there are an odd number of elements in the array.
{}	null	because there are no elements in the array

1. Write a function named **isSquare** that returns 1 if its integer argument is a square of some integer, otherwise it returns 0. **Your function must not use any function or method (e.g. sqrt) that comes with a runtime library or class library!** You will need to write a loop to solve this problem. **Furthermore, your method should return as soon as the status of its parameter is known**. So once it is known that the input parameter is a square of some integer, your method should return 1 and once it is known that the input is not a square, the method should return 0. There should be no wasted loop cycles, your method should be efficent!

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

Examples:

if n is	return	reason
4	1	because 4 = 2*2
25	1	because 25 = 5*5
-4	0	because there is no integer that when squared equals -4. (note, -2 squared is 4 not -4)
8	0	because the square root of 8 is not an integer.
0	1	because 0 = 0*0

2. An array is called **complete** if it contains an even element, a perfect square and two different elements that sum to 8. For example, {3, 2, 9, 5} is complete because 2 is even, 9 is a perfect square and a[0] + a[3] = 8.

Write a function named isComplete that accepts an integer array and returns 1 if it is a complete array, otherwise it returns 0. **Your method must be efficient. It must return as soon as it is known that the array is complete.** Hint: reuse the method you wrote for question 1.

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

int isComplete(int[ ] a)

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

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

Other examples

if the input array is	return	reason
{36, -28}	1	36 is even, 36 is a perfect square, 36-28 = 8
{36, 28}	0	There are no two elements that sum to 8
{4}	0	It does not have two different elements that sum to 8 (you can't use a[0]+a[0])
{3, 2, 1, 1, 5, 6}	0	there is no perfect square.
{3, 7, 23, 13, 107, -99, 97, 81}	0	there is no even number.

3. Write a function that takes two arguments, an array of integers and a positive, non-zero number n. It sums n elements of the array starting at the beginning of the array. If n is greater than the number of elements in the array, the function loops back to the beginning of the array and continues summing until it has summed n elements. You may assume that the array contains at least one element and that n is greater than 0.

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

int loopSum(int[ ] a, int n)

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

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

Examples

If a is	and n is	then function returns
{1, 2, 3}	2	3 (which is a[0] + a[1])
{-1, 2, -1}	7	-1 (which is a[0] + a[1] + a[2] + a[0] + a[1] + a[2] + a[0])
{1, 4, 5, 6}	4	16 (which is a[0] + a[1] + a[2] + a[3])
{3}	10	30 (a[0]+a[0]+a[0]+a[0]+a[0]+a[0]+a[0]+a[0]+a[0]+a[0])

1 Write a function named **allValuesTheSame** that returns 1 if all elements of its argument array have the same value. Otherwise, it returns 0.

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

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

Examples:

if a is	return
{1, 1, 1, 1}	1
{83, 83, 83}	1
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}	1
{1, -2343456, 1, -2343456}	0 (because there are two different values, 1 and -2343456)
{0, 0, 0, 0, -1}	0 (because there are two different values, 0 and -1)
{432123456}	1
{-432123456}	1
{ }	0

2 Write a function named **hasNValues** which takes an array and an integer *n* as arguments. It returns true if all the elements of the array are one of n different values.

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

If you are writing in C or C++, the function signature is
int hasNValues(int a[ ], int n, int len) where len is the length of a

Note that an array argument is passed by reference so that any change you make to the array in your function will be visible when the function returns. Therefore, you must not modify the array in your function! In other words, your function should have no side effects.

Examples

if a is	if n is	return
{1, 2, 2, 1}	2	1 (because there are 2 different element values, 1 and 2)
{1, 1, 1, 8, 1, 1, 1, 3, 3}	3	1 (because there are 3 different element values, 1, 3, 8)
{1, 2, 3, 4, 5, 6, 7, 8 ,9, 10}	10	1 (because there are 10 different element values)
{1, 2, 2, 1}	3	0 (because there are 2 different element values, not 3 as required)
{1, 1, 1, 8, 1, 1, 1, 3, 3}	2	0 (because there are 3 different element values, not 2 as required)
{1, 2, 3, 4, 5, 6, 7, 8 ,9, 10}	20	0 (because there are 10 different element values, not 20 as required)

Hint: There are many ways to solve this problem. One way is to allocate an array of n integers and add each unique element found in the array parameter to it. If you add n elements to the array, return 1, otherwise return 0.

3 Write a function named **sameNumberOfFactors** that takes two integer arguments and returns 1 if they have the same number of factors. If either argument is negative, return -1. Otherwise return 0.

int sameNumberOfFactors(int n1, int n2)

Examples:

if n1 is	and n2 is	return
-6	21	-1 (because one of the arguments is negative)
6	21	1 (because 6 has four factors (1, 2, 3, 6) and so does 21 (1, 3, 7, 21)
8	12	0 (because 8 has four factors(1, 2, 4, 8) and 12 has six factors (1, 2, 3, 4, 6, 12)
23	97	1 (because 23 has two factors (1, 23) and so does 97 (1, 97))

0	1	0 (because 0 has no factors, but 1 has one (1))
0	0	1 (always true if n1 == n2)

1. Write a function named *eval* that returns the value of the polynomial $a_n x^n + a_{n-1}x^{n-1} + ... + a_1x^1 + a_0$.

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

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

Examples:

if x is	if the input array is	this represents	eval should return
1.0	{0, 1, 2, 3, 4}	$4x^4 + 3x^3 + 2x^2 + x + 0$	10.0
3.0	{3, 2, 1}	$x^2 + 2x + 3$	18.0
2.0	{3, -2, -1}	$-x^2 - 2x + 3$	-5.0
-3.0	{3, 2, 1}	$x^2 + 2x + 3$	6.0
2.0	{3, 2}	$2x + 3$	7.0
2.0	{4, 0, 9}	$9x^2 + 4$	40.0
2.0	{10}	10	10.0
10.0	{0, 1}	x	10.0

Copy and paste your answer here and click the "Submit answer" button

4. **A non-empty array *a* of length n is called an array of all possiblities** if it contains all numbers between 0 and a.length-1 inclusive. Write a method named **isAllPossibilities** that accepts an integer array and returns 1 if the array is an array of all possiblities, otherwise it returns 0.

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

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

Examples

If the input array is	return
{1, 2, 0, 3}	1
{3, 2, 1, 0}	1
{1, 2, 4, 3}	0 (because 0 not included and 4 is too big)
{0, 2, 3}	0 (because 1 is not included)
{0}	1
{}	0

Copy and paste your answer here and click the "Submit answer" button

5. **An array is called *layered*** if its elements are in ascending order and each element appears two or more times. For example, {1, 1, 2, 2, 2, 3, 3} is layered but {1, 2, 2, 2, 3, 3} and {3, 3, 1, 1, 1, 2, 2} are not. Write a method named **isLayered** that accepts an integer array and returns 1 if the array is layered, otherwise it returns 0.

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

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

Examples:

If the input array is	return
{1, 1, 2, 2, 2, 3, 3}	1
{3, 3, 3, 3, 3, 3, 3}	1
{1, 2, 2, 2, 3, 3}	0 (because there is only one occurence of the value 1)
{2, 2, 2, 3, 3, 1, 1}	0 (because values are not in ascending order)
{2}	0
{}	0

Copy and paste your answer here and click the "Submit answer" button

6. A mileage counter is used to measure mileage in an automobile. A mileage counter looks something like this

The above mileage counter says that the car has travelled 5,998 miles. Each mile travelled by the automobile increments the mileage counter. Here is how the above mileage counter changes over a 3 mile drive.

After the first mile

After the second mile

After the third mile

A mileage counter can be represented as an array. The mileage counter

can be represented as the array
int a[ ] = new int[ ] {8, 9, 9, 5, 0}
Note that the mileage counter is "backwards" in the array, a[0] represents ones, a[1] represents tens, a[2] represents hundreds, etc.

Write a function named updateMileage that takes an array representation of a mileage counter (which can be arbitrarily long) and adds a given number of miles to the array. Since arrays are passed by reference you can update the array in the function, you do not have to return the updated array.

You do not have to do any error checking. You may assume that the array contains non-negative digits and that the mileage is non-negative

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

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

Examples:

if the input array is	and the mileage is	the array becomes
{8, 9, 9, 5, 0}	1	{9, 9, 9, 5, 0}
{8, 9, 9, 5, 0}	2	{0, 0, 0, 6, 0}
{9, 9, 9, 9, 9, 9, 9, 9, 9, 9}	1	{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
{9, 9, 9, 9, 9, 9, 9, 9, 9, 9}	13	{2, 1, 0, 0, 0, 0, 0, 0, 0, 0}

Note that the mileage counter wraps around if it reaches all 9s and there is still some mileage to add.

Hint: Write a helper function that adds 1 to the mileage counter and call the helper function once for each mile

Copy and paste your answer here and click the "Submit answer" button

7. 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. Furthermore, all the zeroes in the array must be in the middle of the array. Write a function named *isHollow* that accepts an integer array and returns 1 if it is a hollow array, otherwise it returns 0.

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

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

Examples:

if the input array is	is hollow?	reason
{1,2,0,0,0,3,4}	yes	2 non-zeros precede and follow 3 zeros in the middle
{1,1,1,1,0,0,0,0,0,2,1,2,18}	yes	4 non-zeros precede and follow the 5 zeros in the middle
{1, 2, 0, 0, 3, 4}	no	There are only 2 zeroes in the middle; at least 3 are required

{1,2,0,0,0,3,4,5}	no	The number of preceding non-zeros(2) is not equal to the number of following non-zeros(3)
{3,8,3,0,0,0,3,3}	no	The number of preceding non-zeros(3) is not equal to the number of following non-zeros(2)
{1,2,0,0,0,3,4,0}	no	Not all zeros are in the middle
{0,1,2,0,0,0,3,4}	no	Not all zeros are in the middle
{0,0,0}	yes	The number of preceding non-zeros is 0 which equals the number of following non-zeros. And there are three zeros "in the middle".

Hint: Write three loops. The first counts the number of preceding non-zeros. The second counts the number of zeros in the middle. The third counts the number of following non-zeros. Then analyze the results.

Copy and paste your answer here and click the "Submit answer" button

8. A positive number n is *consecutive-factored* if and only if it has factors, i and j where i > 1, j > 1 and j = i + 1. Write a function named **isConsecutiveFactored** that returns 1 if its argument is consecutive-factored, otherwise it returns 0.

the function signature is
int isConsectiveFactored(int n)

Examples:

If n is	**return**	**because**
24	1	24 = 2*3*4 and 3 = 2 + 1
105	0	105 = 3*5*7 and 5 != 3+1 and 7 != 5+1
90	1	factors of 90 include 2 and 3 and 3 = 2 + 1
23	0	has only 1 factor that is not equal to 1
15	0	15 = 3*5 and 5 != 3 + 1
2	0	2 = 1*2, 2 = 1 + 1 but factor 1 is not greater than 1
0	0	n has to be positive
-12	0	n has to be positive

Copy and paste your answer here and click the "Submit answer" button

9. A twin prime is a prime number that differs from another prime number by 2. Write a function named **isTwinPrime** with an integer parameter that returns 1 if the parameter is a twin prime, otherwise it returns 0. Recall that a prime number is a number with no factors other than 1 and itself.

the function signature is
int isTwinPrime(int n)

Examples:

number	is twin prime?
5	yes, 5 is prime, 5+2 is prime
7	yes, 7 is prime, 7-2 is prime
53	no, 53 is prime, but neither 53-2 nor 53+2 is prime
9	no, 9 is not prime

10. Write a function named **largestAdjacentSum** that iterates through an array computing the sum of adjacent elements and returning the largest such sum. You may assume that the array has at least 2 elements.

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

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

Examples:

if a is	return
{1, 2, 3, 4}	7 because 3+4 is larger than either 1+2 or 2+3
{18, -12, 9, -10}	6 because 18-12 is larger than -12+9 or 9-10
{1,1,1,1,1,1,1,1,1,1}	2 because all adjacent pairs sum to 2
{1,1,1,1,1,2,1,1,1}	3 because 1+2 or 2+1 is the max sum of adjacent pairs

11. An array is called *zero-balanced* if its elements sum to 0 and for each positive element n, there exists another element that is the negative of n. Write a function named **isZeroBalanced** that returns 1 if its argument is a zero-balanced array. Otherwise it returns 0.

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

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

Examples:

if a is	return
{1, 2, -2, -1}	1 because elements sum to 0 and each positive element has a corresponding negative element.
{-3, 1, 2, -2, -1, 3}	1 because elements sum to 0 and each positive element has a corresponding negative element.
{3, 4, -2, -3, -2}	0 because even though this sums to 0, there is no element whose value is -4
{0, 0, 0, 0, 0, 0}	1 this is true vacuously; 0 is not a positive number
{3, -3, -3}	0 because it doesn't sum to 0. (Be sure your function handles this array correctly)
{3}	0 because this doesn't sum to 0
{}	0 because it doesn't sum to 0

12. Write a function named **largestAdjacentSum** that iterates through an array computing the sum of adjacent elements and returning the largest such sum. You may assume that the array has at least 2 elements.

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

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

Examples:

if a is	return
{1, 2, 3, 4}	7 because 3+4 is larger than either 1+2 or 2+3
{18, -12, 9, -10}	6 because 18-12 is larger than -12+9 or 9-10
{1,1,1,1,1,1,1,1}	2 because all adjacent pairs sum to 2
{1,1,1,1,1,2,1,1}	3 because 1+2 or 2+1 is the max sum of adjacent pairs

13. An array is called *zero-balanced* if its elements sum to 0 and for each positive element n, there exists another element that is the negative of n. Write a function named **isZeroBalanced** that returns 1 if its argument is a zero-balanced array. Otherwise it returns 0.

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

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

Examples:

if a is	return
{1, 2, -2, -1}	1 because elements sum to 0 and each positive element has a corresponding negative element.
{-3, 1, 2, -2, -1, 3}	1 because elements sum to 0 and each positive element has a corresponding negative element.
{3, 4, -2, -3, -2}	0 because even though this sums to 0, there is no element whose value is -4
{0, 0, 0, 0, 0, 0}	1 this is true vacuously; 0 is not a positive number
{3, -3, -3}	0 because it doesn't sum to 0. (Be sure your function handles this array correctly)
{3}	0 because this doesn't sum to 0
{}	0 because it doesn't sum to 0

sort a numeric array in ascending order if that is already sorted in descending order.//ans **DA-short.c**

1. accepts an array and return the percent of integers in the array that is divisible by 3.
2. test an array if that is symmetric. {1,2,3,2,1}, {4,5,6,6,5,4} are symmetric arrays.
3. accepts array of integers and returns an integer that represents the biggest difference between any two values in the array.
4. word game-------
   j, q, x, z---10 points
   k, v----5 points
   A to Z and other small case----1 point
   Other----0 points.

Find the values of {a,r,t,f,z}

5. one word in an anagram of another word if it is a re-arrangement of all the letters of the second word. Write a function that accepts two character arrays and return 1 if they are anagrams of each other else return 0.
6. inputs an array of positive integer and a character array. The integers in the first arrays are indexes into the array of characters. The function should return an array of character containing the character referenced by the integer array.
   e.g.---input—{0,4,7}&{h,a,p,p,i,n,e,s,s} –output—{h,i,s}.
7. input two character arrays and returns the one that is greater in texic order (dictionary order).
   Returns first array---if both are equal
   Returns null ----if either of array is null.
8. write a function that accepts an array of positive integer as its argument and returns an array containing of the odd integers in the input array. The length of the input array should be equal to the number of odd integers that it contains.
   Input----{1,2,3,4,5} ---output---{1,3,5}---should not be---{1,3,5,0,0}
9. write a function that accepts an array of integer and returns the number of distinct integers in the array.
   Input—{1,2,3,10}---output—4
   Input---{5,5,5,5}-----output---1
10. write a function that accepts a character array that contains digits as input and returns its integer equivalent. If a non-digit character is found or the input array is null, or contains no element the function should return -1.
    Input---{1,3,9}—output—139
    Input---{0,4,7}—ouput---47
    Input---{1,+,-2}—output…..-1
    input{}----output…….-1
11. accepts an array and find if the array is happy. If each elements in the array exists exactly two times then the array is happy.
12. balanced array: even elements is even and odd elements is odd
    e.g.{1,2,3,4,7,8}
13. odd heavy: array if it contains at least one odd element and every element whose value is odd is greater than every even-valued element{11,4,9,2,8}---11,9 are greater than all even elements\
14. accepts a character array and print the * for each character and also count the repeatable character and mark it by *.
    Input—{a,b,a,b,c,d,a,c,a,b,d,f}---
    output—
    a--****
    b--***
    c--**
    d--**
    f--*
15. Fibonacci series (0, 1, 1, 2, 3, 5, 8, 13, 21, 34 …….)
16. prime no
17. factorial no

18. eliminating duplicate no from array
19. finding out largest and second largest.