Sample 1 questions answer for the MUM entrance exam

After applying in the Maharishi University of Management (MUM) for Master of Science in Computer Science, we get student id number provided by the University and we have to appear to the entrance exam, and in the entrance exam you will have three questions to be solved within 2 hours. I have collected some of the question and try to provide you the answers. Some of the sample questions that i am able to collect for the MUM entrance exam are as follows:

There are three questions on the exam. You have two hours to finish. **Please do your own work**.

1. Write a function named **primeCount** with signature

int primeCount(int start, int end);

The function returns the number of primes between *start* and *end* inclusive. Recall that a prime is a positive integer greater than 1 whose only integer factors are 1 and itself.

Examples

If start is	and end is	return	reason
10	30	6	The primes between 10 and 30 inclusive are 11, 13, 17, 19, 23 and 29
11	29	6	The primes between 11 and 29 inclusive are 11, 13, 17, 19, 23 and 29
20	22	0	20, 21, and 22 are all non-prime
1	1	0	By definition, 1 is not a prime number
5	5	1	5 is a prime number
6	2	0	start must be less than or equal to end
-10	6	3	primes are greater than 1 and 2, 3, 5 are prime

2. A **Madhav** array has the following property.

a[0] = a[1] + a[2] = a[3] + a[4] + a[5] = a[6] + a[7] + a[8] + a[9] = …

The length of a Madhav array must be n*(n+1)/2 for some n.

Write a method named *isMadhavArray* that returns 1 if its array argument is a Madhav array, otherwise it returns 0. If you are programming in Java or C# the function signature is

int isMadhavArray(int[ ] a)

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

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

Examples

if a is	return	reason
{2, 1, 1}	1	2 + 1 + 1
{2, 1, 1, 4, -1, -1}	1	2 = 1 + 1, 2 = 4 + -1 + -1
{6, 2, 4, 2, 2, 2, 1, 5, 0, 0}	1	6 = 2 + 4, 6 = 2 + 2 + 2, 6 = 1 + 5 + 0 + 0
{18, 9, 10, 6, 6, 6}	0	18 != 9 + 10
{-6, -3, -3, 8, -5, -4}	0	-6 != 8 + -5 + -4
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, -2, -1}	1	0 = 0 + 0, 0 = 0 + 0 + 0, 0 = 0 + 0 + 0 + 0, 0 = 1 + 1 + 1 + -2 + -1
{3, 1, 2, 3, 0}	0	The length of the array is 5, but 5 does not equal n*(n+1)/2 for any value of n.

3. An array is defined to be **inertial** if the following conditions hold:

a. it contains at least one odd value

b. the maximum value in the array is even

c. every odd value is greater than every even value that is not the maximum value.

So {11, 4, 20, 9, 2, 8} is inertial because

a. it contains at least one odd value

b. the maximum value in the array is 20 which is even

c. the two odd values (11 and 9) are greater than all the

even values that are not equal to 20 (the maximum), i.e., (4, 2, 8}.

However, {12, 11, 4, 9, 2, 3, 10} is **not** inertial because it fails condition (c), i.e., 10 (which is even) is greater 9 (which is odd) but 10 is not the maximum value in the array.

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

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

int isInertial(int[ ] a

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

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

Some other examples:

if the input array is	**return**	**reason**
{1}	0	fails condition (a) – the maximum value must be even
{2}	0	fails condition (b) – the array must contain at least one odd value.
{1, 2, 3, 4}	0	fails condition (c) – 1 (which is odd) is not greater than all even values other than the maximum (1 is less than 2 which is not the maximum)
{1, 1, 1, 1, 1, 1, 2}	1	there is no even number other than the maximum. Hence, there can be no other even values that are greater than 1.
{2, 12, 4, 6, 8, 11}	1	11, the only odd value is greater than all even values except 12 which is the maximum value in the array.
{2, 12, 12, 4, 6, 8, 11}	1	same as previous, i.e., it is OK if maximum value occurs more than once.
{-2, -4, -6, -8, -11}	0	-8, which is even, is not the maximum value but is greater than -11 which is odd
{2, 3, 5, 7}	0	the maximum value is odd
{2, 4, 6, 8, 10}	0	there is no odd value in the array.

There are three questions on this exam. You have two hours to complete it. Please do your own work.

1.Define a **square pair** to be the tuple $<x, y>$ where x and y are positive, non-zero integers, $x<y$ and $x + y$ is a perfect square. A perfect square is an integer whose square root is also an integer, e.g. 4, 9, 16 are perfect squares but 3, 10 and 17 are not. Write a function named *countSquarePairs* that takes an array and returns the number of square pairs that can be constructed from the elements in the array. For example, if the array is {11, 5, 4, 20} the function would return 3 because the only square pairs that can be constructed from those numbers are <5, 11>,

<5, 20> and <4, 5>. **You may assume that there exists a function named *isPerfectSquare* that returns 1 if its argument is a perfect square and 0 otherwise. E.G., isPerfectSquare(4) returns 1 and isPerfectSquare(8) returns 0.**

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

int countSquarePairs(int[ ] a)

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

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

You may assume that there are no duplicate values in the array, i.e, you don't have to deal with an array like {2, 7, 2, 2}.

Examples:

if a is	return	reason
{9, 0, 2, -5, 7}	2	The square pairs are <2, 7> and <7, 9>. Note that <-5, 9> and <0, 9> are not square pairs, even though they sum to perfect squares, because both members of a square pair have to be greater than 0. Also <7,2> and <9,7> are not square pairs because the first number has to be less than the second number.
{9}	0	The array must have at least 2 elements

2. A **prime number** is an integer that is divisible only by 1 and itself. A **porcupine number** is a prime number whose last digit is 9 and the next prime number that follows it also ends with the digit 9. For example 139 is a porcupine number because:

a. it is prime

b. it ends in a 9

c. The next prime number after it is 149 which also ends in 9. Note that 140, 141, 142, 143, 144, 145, 146, 147 and 148 are **not** prime so 149 is the next prime number after 139.

Write a method named *findPorcupineNumber* which takes an integer argument *n* and returns the first porcupine number **that is greater than *n***. So findPorcupineNumber(0) would return 139 (because 139 happens to be the first porcupine number) and so would findPorcupineNumber(138). But findPorcupineNumber(139) would return 409 which is the second porcupine number.

The function signature is

int findPorcupineNumber(int n)

You may assume that a porcupine number greater than *n* exists.

You may assume that a function *isPrime* exists that returns 1 if its argument is prime, otherwise it returns 0. E.G., isPrime(7) returns 1 and isPrime(8) returns 0.

Hint: Use modulo base 10 arithmetic to get last digit of a number.

3. Consider the following algorithm

Start with a positive number n

if n is even then divide by 2

if n is odd then multiply by 3 and add 1

continue this until n becomes 1

The **Guthrie sequence** of a positive number *n* is defined to be the numbers generated by the above algorithm.

For example, the Guthrie sequence of the number 7 is

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

It is easy to see that this sequence was generated from the number 7 by the above algorithm. Since 7 is odd multiply by 3 and add 1 to get 22 which is the second number of the sequence. Since 22 is even, divide by 2 to get 11 which is the third number of the sequence. 11 is odd so multiply by 3 and add 1 to get 34 which is the fourth number of the sequence and so on.

Note: the first number of a Guthrie sequence is always the number that generated the sequence and the last number is always 1.

Write a function named *isGuthrieSequence* which returns 1 if the elements of the array form a Guthrie sequence. Otherwise, it returns 0.

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

int isGuthrieSequence(int[ ] a)

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

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

Examples

if a is	return	reason
{8, 4, 2, 1}	1	This is the Guthrie sequence for 8
{8, 17, 4, 1}	0	This is not the Guthrie sequence for 8
{8, 4, 1}	0	Missing the 2
{8, 4, 2}	0	A Guthrie sequence must end with 1

There are three questions on this exam. You have two hours to complete it. Please do your own work.

1. The **Stanton measure** of an array is computed as follows. Count the number of 1s in the array. Let this count be *n*. The Stanton measure is the number of times that *n* appears in the array. For example, the Stanton measure of {1, 4, 3, 2, 1, 2, 3, 2} is 3 because 1 occurs 2 times in the array and 2 occurs 3 times.

Write a function named *stantonMeasure* that returns the Stanton measure of its array argument.

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

int stantonMeasure(int[ ] a)

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

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

Examples

if a is	return	reason
{1}	1	1 occurs 1 time, 1 occurs 1 time
{0}	1	1 occurs 0 times, 0 occurs 1 time
{3, 1, 1, 4}	0	1 occurs 2 times, 2 occurs 0 times
{1, 3, 1, 1, 3, 3, 2, 3, 3, 3, 4}	6	1 occurs 3 times, 3 occurs 6 times
{}	0	1 occurs 0 times, 0 occurs 0 times

2. The **sum factor** of an array is defined to be the number of times that the sum of the array appears as an element of the array. So the sum factor of {1, -1, 1, -1, 1, -1, 1} is 4 because the sum of the elements of the array is 1 and 1 appears four times in the array. And the sum factor of

{1, 2, 3, 4} is 0 because the sum of the elements of the array is 10 and 10 does not occur as an element of the array. The sum factor of the empty array { } is defined to be 0.

Write a function named *sumFactor* that returns the sum factor of its array argument.

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

int sumFactor(int[ ] a)

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

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

Examples:

if a is	return	reason
{3, 0, 2, -5, 0}	2	The sum of array is 0 and 0 occurs 2 times
{9, -3, -3, -1, -1}	0	The sum of the array is 1 and 1 does not occur in array.
{1}	1	The sum of the array is 1 and 1 occurs once in the array
{0, 0, 0}	3	The sum of the array is 0 and 0 occurs 3 times in the array

3. Consider the following algorithm

Start with a positive number n

if n is even then divide by 2

if n is odd then multiply by 3 and add 1

continue this until n becomes 1

The **Guthrie index** of a positive number n is defined to be how many iterations of the above algorithm it takes before n becomes 1.

For example, the Guthrie index of the number 7 is 16 because the following sequence is 16 numbers long.

22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

It is easy to see that this sequence was generated by the above algorithm. Since 7 is odd multiply by 3 and add 1 to get 22 which is the first number of the sequence. Since 22 is even, divide by 2 to get 11 which is the second number of the sequence. 11 is odd so multiply by 3 and add 1 to get 34 which is the third number of the sequence and so on.

Write a function named *guthrieIndex* which computes the Guthrie index of its argument. Its signature is

int guthrieIndex(int n)

Examples

if n is	return	sequence
1	0	number is already 1
2	1	1
3	7	10, 5, 16, 8, 4, 2, 1
4	2	2, 1
42	8	21, 64, 32, 16, 8, 4, 2, 1

You may assume that the length of the sequence can be represented by a 32 bit signed integer.

There are 3 questions on this test. You have two hours to do it. **Please do your own work.**

1. It is a fact that there exist two numbers x and y such that x! + y! = 10!. Write a method named **solve10** that returns the values x and y in an array.

The notation n! is called **n factorial** and is equal to *n * n-1 * n-2 * ... 2 * 1*, e.g., 5! = 5*4*3*2*1 = 120.

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

int[ ] solve10() where the length of the returned array is 2.

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

int * solve10() where the length of the returned array is 2.

Please be sure that the method solve10 returns an array, a, with two elements

where a[0] == x, a[1] == y and x! + y! = 10!.

2. An array can hold the digits of a number. For example the digits of the number 32053 are stored in the array {3, 2, 0, 5, 3}. Write a method call **repsEqual** that takes an array and an integer and returns 1 if the array contains ***only*** the digits of the number ***in the same order*** that they appear in the number. Otherwise it returns 0.

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

int repsEqual(int[ ] a, int n)

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

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

Examples (note: your program must work for all values of a and n, not just those given here!)

if a is	and n is	return	reason
{3, 2, 0, 5, 3}	32053	1	the array contains only the digits of the number, in the same order as they are in the number.
{3, 2, 0, 5}	32053	0	the last digit of the number is missing from the array.
{3, 2, 0, 5, 3, 4}	32053	0	an extra number (4) is in the array.
{2, 3, 0, 5, 3}	32053	0	the array elements are not in the same order as the digits of the number
{9, 3, 1, 1, 2}	32053	0	elements in array are not equal to digits of number.
{0, 3, 2, 0, 5, 3}	32053	1	you can ignore leading zeroes.

3. An array is called **centered-15** if some consecutive sequence of elements of the array sum to 15 and this sequence is preceded and followed by the same number of elements. For example

{*3, 2*, **10, 4, 1**, *6, 9*} is centered-15 because the sequence 10, 4, 1 sums to 15 and the sequence is preceded by two elements (3, 2) and followed by two elements(6,9).

Write a method called i*sCentered15* that returns 1 if its array argument is centered-15, otherwise it returns 0.

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

int isCentered15(int[ ] a)

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

int isCentered5(int a[ ], int len) where len is the number of elements in the array.

Examples

if a is	return	reason
{*3, 2*, **10, 4, 1**, *6, 9*}	1	the sequence 10, 4, 1 sums to 15 and is preceded by 2 elements and followed by 2 elements. Note that there is another sequence that sums to 15 (6,9}. It is okay for the array to have more than one sequence that sums to 15 as long as at least one of them is centered.
{*2*, **10, 4, 1**, *6, 9*}	0	(10, 4, 1) is preceded by one element but followed by two. (9,6) is preceded by five elements but followed by none. Hence neither qualify as centered.
{3, *2*, **10, 4, 1**, *6*}		(10, 4, 1) is preceded by two elements but followed by one. Note that the values 3, 2, 4, 6 sum to 15 but they are not consecutive.
{1,1,8, 3, 1, 1}		The entire array sums to 15, hence the sequence is preceded by zero elements and followed by zero elements.
{9, **15**, 6}	1	the sequence (15) is preceded by one element and followed by one element.
{1, 1, 2, 2, 1, 1}	0	no sequence sums to 15.
{**1, 1, 15 -1,-1**}	1	there are three different sequences that sum to 15, the entire array, (1, 15, -1) and (15). In this case they all are centered but the requirement is that just one of them has to be.

There are three questions on this test. You have two hours to finish it. Please do your own work.

1. A **perfect number** is one that is the sum of its factors, excluding itself. The 1st perfect number is 6 because 6 = 1 + 2 + 3. The 2nd perfect number is 28 which equals 1 + 2 + 4 + 7 + 14. The third is 496 = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248. In each case, the number is the sum of all its factors excluding itself.

Write a method named *henry* that takes two integer arguments, i and j and returns the sum of the ith and jth perfect numbers. So for example, henry (1, 3) should return 502 because 6 is the 1st perfect number and 496 is the 3rd perfect number and 6 + 496 = 502.

The function signature is

int henry (int i, int j)

You do not have to worry about integer overflow, i.e., you may assume that each sum that you have to compute can be represented as a 31 bit integer. Hint: use modulo arithmetic to determine if one number is a factor of another.

2. Write a method named *isDivisible* that takes an integer array and a divisor and returns 1 if all its elements are divided by the divisor with no remainder. Otherwise it returns 0.

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

int isDivisible(int [ ] a, int divisor)

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

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

Examples

if a is	and divisor is	return	because
{3, 3, 6, 36}	3	1	all elements of a are divisible by 3 with no remainder.
{4}	2	1	all elements of a are even
{3, 4, 3, 6, 36}	3	0	because when a[1] is divided by 3, it leaves a remainder of 1
{6, 12, 24, 36}	12	0	because when a[0] is divided by 12, it leaves a remainder of 6.
{}	anything	1	because no element fails the division test.

3. An array is defined to be **n-unique** if exactly one pair of its elements sum to *n*. For example, the array {2, 7, 3, 4} is 5-unique because only a[0] and a[2] sum to 5. But the array {2, 3, 3, 7} is not 5-unique because a[0] + a[1] = 5 and a[0] + a[2] = 5.

Write a function named is*NUnique* that returns 1 if its integer array argument is n-unique, otherwise it returns 0. So isNUnique(new int[ ]{2, 7, 3, 4}, 5) should return 1 and

isNUnique(new int[] {2, 3, 3, 7}, 5) should return 0.

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

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

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

Examples

If a is	and n is	return	because
{7, 3, 3, 2, 4}	6	0	because a[1]+a[2] == 6 and a[3]+a[4] also == 6.
{7, 3, 3, 2, 4}	10	0	because a[0]+a[1] == 10 and a[0] + a[2] also == 10
{7, 3, 3, 2, 4}	11	1	because only a[0] + a[4] sums to 11
{7, 3, 3, 2, 4}	8	0	because no pair of elements sum to 8. (Note that a[1]+a[2]+a[3] do sum to 8 but the requirement is that two elements sum to 8.)
{7, 3, 3, 2, 4}	4	0	no pair of elements sum to 4. (Note that the a[4]==4, but the requirement is that two elements have to sum to 4.)
{1}	anything	0	array must have at least 2 elements

Sample 2 question collections of MUM university MS computer science

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

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!**

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. A number with a base other than 10 can be written using its base as a subscript. For example, 1011_2 represents the binary number 1011 which can be converted to a base 10 number as follows:

$(1 * 2^0) + (1 * 2^1) + (0 * 2^2) + (1 * 2^3) = 1 + 2 + 0 + 8 = 11_{10}$

Similarly, the base 3 number 112_3 can be converted to base 10 as follows:

$(2 * 3^0) + (1 * 3^1) + (1 * 3^2) = 2 + 3 + 9 = 14_{10}$

And the base 8 number 325_8 can be converted to base 10 as follows:

$(5 * 8^0) + (2 * 8^1) + (3 * 8^2) = 5 + 16 + 192 = 213_{10}$

Write a method named ***isLegalNumber*** that takes two arguments. The first argument is an array whose elements are the digits of the number to test. The second argument is the base of the number represented by the first argument. The method returns 1 if the number represented by the array is a legal number in the given base, otherwise it returns 0.

For example the number 321_4 can be passed to the method as follows:

isLegalNumber(new int[] {3, 2, 1}, 4)

This call will return 1 because 321_4 is a legal base 4 number.

However, since all digits of a base *n* number must be less than *n*, the following call will return 0 because 371_6 is not a legal base 6 number (the digit 7 is not allowed)

isLegalNumber(new int[] {3, 7, 1}, 6)

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

int isLegalNumber(int[ ] a, int base)

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

int isLegalNumber(int a[ ], int len, int base) where len is the size of the array.

3. Using the <array, base> representation for a number described in the second question write a method named ***convertToBase10*** that converts its <array, base> arguments to a base 10 number if the input is legal for the specified base. If it is not, it returns -1.

Some examples:

convertToBase10(new int[ ] {1, 0, 1, 1}, 2) returns 11 convertToBase10(new int[ ] {1, 1, 2}, 3) returns 14

convertToBase10(new int[ ] {3, 2, 5}, 8) returns 213

convertToBase10 (new int[ ] {3, 7, 1}, 6) returns 0 because 371 is not a legal base 6 number.

Your convertToBase10 method must call the isLegalNumber method that you wrote for question 2.

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

int convertToBase10(int[ ] a, int base)

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

int convertToBase10(int a[ ], int len, int base) where len is the size of the array.

There are 3 questions on this test. You have 2 hours to finish it. Please use tabs or spaces to indent your program.

1. A simple pattern match on the elements of an array ***A*** can be defined using another array ***P***. Each element ***n*** of ***P*** is negative or positive (never zero) and defines the number of elements in a sequence in ***A***. The first sequence in ***A*** starts at ***A***[0] and its length is defined by ***P***[0]. The second sequence follows the first sequence and its length is defined by ***P***[1] and so on. Furthermore, for ***n*** in ***P***, if ***n*** is positive then the sequence of ***n*** elements of ***A*** must all be positive. Otherwise the sequence of abs(***n)*** elements must all be negative. The sum of the absolute values of the elements of ***P*** must be the length of ***A***. For example, consider the array

A = {1, 2, 3, -5, -5, 2, 3, 18}

If ***P*** = {3, -2, 3} then ***A*** matches ***P*** because

i. the first **3** elements of ***A*** (1, 2, 3) are positive (P[0] is **3** and is positive),

ii. the next **2** elements of ***A*** (-5, -5) are negative (P[1] is -**2** and is negative)

iii. and the last **3** elements of ***A*** (2, 3, 18) are positive (P[2] is **3** and is positive)

Notice that the absolute values of the elements of ***P*** sum to 8 which is the length of ***A***. The array ***A*** also matches the following patterns:

{2, 1, -1, -1, 2, 1}, {1, 2, -1, -1, 1, 2}, {2, 1, -2, 3}, {1, 1, 1, -1, -1, 1, 1, 1}

In each case the sum of the absolute values is 8, which is the length of ***A*** and each sequence of numbers in ***A*** defined in a pattern is negative or positive as required.

The array ***A*** = {1, 2, 3, -5, -5, 2, 3, 18} does **not** match the following patterns:

i. P = {4, -1, 3} (because the first 4 elements of ***A*** are not positive (***A***[3] is negative) as required by P)

ii. P = {2, -3, 3} (because even though the first 2 elements of ***A*** are positive, the next 3 are required to be negative but ***A***[2] is positive which does not satisfy this requirement.)

iii. P = {8} (because this requires all elements of ***A*** to be positive and they are not.)

Please note: Zero is neither positive nor negative.

Write a function named ***matches*** which takes arrays ***A*** and ***P*** as arguments and returns 1 if ***A*** matches ***P***. Otherwise it returns 0. **You may assume that *P* is a legal pattern, i.e., the absolute value of its elements sum to the length of *A* and it contains no zeros. So do not write code to check if P is legal!**

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

int matches(int[ ] a, int[ ] p)

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

int matches(int a[ ], int len, int p[ ]) where ***len*** is the number of elements of ***a***. Furthermore, the value of ***p[0]*** should be the length of ***p***. So, for example, if ***p***={5, 2, -1, -2, 4}, ***p***[0]=5 means that the array has 5 elements and that the last 4 define the pattern.

Hint: Your function should have one loop nested in another. The outer loop iterates through the elements of ***P***. The inner loop iterates through the next sequence of ***A***. The upper bound of the inner loop is the absolute value of the current element of ***P***. The lower bound of the inner loop is 0. The loop variable of the inner loop is **not** used to index **A**!

2. Define a **stacked number** to be a number that is the sum of the first ***n*** positive integers for some ***n***. The first 5 stacked numbers are

1 = 1

3 = 1 + 2

6 = 1 + 2 + 3

10 = 1 + 2 + 3+ 4

15 = 1 + 2 + 3 + 4 + 5

Note that from the above we can deduce that 7, 8, and 9 are not stacked numbers because they cannot be the sum of any sequence of positive integers that start at 1.

Write a function named **isStacked** that returns 1 if its argument is stacked. Otherwise it returns 0. Its signature is:

int isStacked(int n);

So for example, isStacked(10) should return 1 and isStacked(7) should return 0.

3. Define an array to be **sum-safe** if none of its elements is equal to the sum of its elements. The array

a = {5, -5, 0} is not sum-safe because the sum of its elements is 0 and a[2] == 0. However, the array a = {5, -2, 1} is sum-safe because the sum of its elements is 4 and none of its elements equal 4.

Write a function named **isSumSafe** that returns 1 if its argument is sum-safe, otherwise it returns 0.

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

int isSumSafe(int[ ]a)

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

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

For example, isSumSafe(new int[ ] {5, -5, 0}) should return 0 and isSumSafe(new int[ ]{5, -2, 1}) should return 1.

Return 0 if the array is empty.

There are three questions on this exam. You have two hours to finish. Please do your own work.

1. Define a positive number to be **isolated** if none of the digits in its square are in its cube. For example 163 is n isolated number because 69*69 = 26569 and 69*69*69 = 4330747 and the square does not contain any of the digits 0, 3, 4 and 7 which are the digits used in the cube. On the other hand 162 is **not** an isolated number because 162*162=**26244** and 162*162*162 = **42**51**52**8 and the digits 2 and 4 which appear in the square are also in the cube.

Write a function named *isIsolated* that returns 1 if its argument is an isolated number, it returns 0 if its not an isolated number and it returns -1 if it cannot determine whether it is isolated or not (see the note below). The function signature is:

int isIsolated(long n)

Note that the type of the input parameter is *long*. The maximum positive number that can be represented as a long is 63 bits long. This allows us to test numbers up to 2,097,15**1** because the cube of 2,097,151 can be represented as a long. However, the cube of 2,097,15**2** requires more than 63 bits to represent it and hence cannot be computed without extra effort. Therefore, your function should test if n is larger than 2,097,151 and return -1 if it is. If n is less than 1 your function should also return -1.

Hint: n % 10 is the rightmost digit of n, n = n/10 shifts the digits of n one place to the right.

The first 10 isolated numbers are

N	n*n	n*n*n
2	4	8
3	9	27
8	64	512
9	81	729
14	196	2744
24	576	13824
28	784	21952
34	1156	39304
58	3364	195112
63	3969	250047

Questions 2 and 3 are on the next page.

2. An array is called **vanilla** if all its elements are made up of the same digit. For example {1, 1, 11, 1111, 1111111} is a vanilla array because all its elements use only the digit 1. However, the array {11, 101, 1111, 11111} is **not** a vanilla array because its elements use the digits 0 and 1. Write a method called *isVanilla* that returns 1 if its argument is a vanilla array. Otherwise it returns 0.

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

int isVanilla(int[ ] a)

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

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

Example

if a is	Return	reason
{1}	1	all elements use only digit 1.
{11, 22, 13, 34, 125}	0	Elements used 5 different digits
{9, 999, 99999, -9999}	1	Only digit 9 is used by all elements. Note that negative numbers are okay.
{ }	1	There is no counterexample to the hypothesis that all elements use the same digit.

3. Define an array to be **trivalent** if all its elements are one of three different values. For example, {22, 19, 10, 10, 19, 22, 22, 10} is trivalent because all elements are either 10, 22, or 19. However, the array {1, 2, 2, 2, 2, 2, 2} is **not** trivalent because it contains only two different values (1, 2). The array {2, 2, 3, 3, 3, 3, 2, 41, 65} is **not** trivalent because it contains four different values (2, 3, 41, 65).

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

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

int isTrivalent (int[ ] a)

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

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

Hint: Remember that the elements of the array can be any value, so be careful how you initialize your local variables! For example using -1 to initialize a variable won't work because -1 might be one of the values in the array.

Examples

if a is	return	Reason
{-1, 0, 1, 0, 0, 0}	1	All elements have one of three values (0, -1, 1)
{ }	0	There are no elements
{ 2147483647, -1, -1 -2147483648}	1	Again only three different values. Note that the value of a[0] is the maximum integer and the value of a[3] is the minimum integer so you can't use those to initialize local variables.

There are 3 questions on this exam. You have 2 hours to complete it. Please do your own work and use indentation.

1. Write a function named *countRepresentations* that returns the number of ways that an amount of money in rupees can be represented as rupee notes. For this problem we only use rupee notes in denominations of 1, 2, 5, 10 and 20 rupee notes.

The signature of the function is:

int countRepresentations(int numRupees)

For example, countRepresentations(12) should return 15 because 12 rupees can be represented in the following 15 ways.

1. 12 one rupee notes

2. 1 two rupee note plus 10 one rupee notes

3. 2 two rupee notes plus 8 one rupee notes

4. 3 two rupee notes plus 6 one rupee notes

5. 4 two rupee notes plus 4 one rupee notes

6. 5 two rupee notes plus 2 one rupee notes

7. 6 two rupee notes

8. 1 five rupee note plus 7 one rupee notes

9. 1 five rupee note, 1 two rupee note and 5 one rupee notes

10. 1 five rupee note, 2 two rupee notes and 3 one rupee notes

11. 1 five rupee note, 3 two notes and 1 one rupee note

12. 2 five rupee notes and 2 one rupee notes

13. 2 five rupee notes and 1 two rupee note

14. 1 ten rupee note and 2 one rupee notes

15. 1 ten rupee note and 1 two rupee note

Hint: Use a nested loop that looks like this. Please fill in the blanks intelligently, i.e. minimize the number of times that the if statement is executed.

```
for (int rupee20=0; rupee20<=__; rupee20++)

for (int rupee10=0; rupee10<=__; rupee10++)

for (int rupee5=0; rupee5<=__; rupee5++)

for (int rupee2=0; rupee2<=__; rupee2++)

for (int rupee1=0;  rupee1<=__; rupee1++)

{

if (___)

count++

}
```

2. An integer array is defined to be **sequentially-bounded** if it is in ascending order and each value, n, in the array occurs less than n times in the array. So {2, 3, 3, 99, 99, 99, 99, 99} is sequentially-bounded because it is in ascending order and the value 2 occurs less than 2 times, the value 3 occurs less than 3 times and the value 99 occurs less than 99 times. On the other hand, the array {1, 2, 3} is not sequentially-bounded because the value 1 does not occur < 1 times. The array {2, 3, 3, 3, 3} is not sequentially-bounded because the maximum allowable occurrences of 3 is 2 but 3 occurs 4 times. The array {12, 12, 9} is not sequentially-bounded because it is not in ascending order.

Write a function named *isSequentiallyBounded* that returns 1 if its array argument is sequentially-bounded, otherwise it returns 0.

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

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

Examples

if a is	return	Reason
{0, 1}	0	the value 0 has to occur less than 0 times, but it doesn't
{-1, 2}	0	if array contains a negative number, return 0.
{}	1	since there are no values, there are none that can fail the test.
{5, 5, 5, 5}	1	5 occurs less than 5 times
{5, 5, 5, 2, 5}	0	array is not in ascending order.

3. An array is defined to be **minmax-disjoint** if the following conditions hold:

a. The minimum and maximum values of the array are not equal.

b. The minimum and maximum values of the array are not adjacent to one another.

c. The minimum value occurs exactly once in the array.

d. The maximum value occurs exactly once in the array.

For example the array {5, 4, 1, 3, 2} is minmax-disjoint because

a. The maximum value is 5 and the minimum value is 1 and they are not equal.

b. 5 and 1 are not adjacent to one another

c. 5 occurs exactly once in the array

d. 2 occurs exactly once in the array

Write a function named *isMinMaxDisjoint* that returns 1 if its array argument is minmax-disjoint, otherwise it returns 0.

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

int isMinMaxDisjoint(int[ ] a)

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

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

Examples of arrays that are **not** minMaxDisjoint

if a is	return	Reason
{18, -1, 3, 4, 0}	0	The max and min values are adjacent to one another.
{9, 0, 5, 9}	0	The max value occurs twice in the array.
{0, 5, 18, 0, 9|	0	The min value occurs twice in the array.
{7, 7, 7, 7}	0	The min and the max value must be different.
{}	0	There is no min or max.
{1, 2}	0	The min and max elements are next to one another.

{1}	0	The min and the max are the same.

There are 3 questions on this exam. You have 2 hours to complete it. Please do your own work.

1. The number 124 has the property that it is the smallest number whose first three multiples contain the digit 2. Observe that

124*1 = 124, 124*2 = 248, 124*3 = 372 and that 124, 248 and 372 each contain the digit 2. It is possible to generalize this property to be the smallest number whose first ***n*** multiples each contain the digit 2. Write a function named **smallest(n)** that returns the smallest number whose first n multiples contain the digit 2. Hint: use modulo base 10 arithmetic to examine digits.

Its signature is

int smallest(int n)

You may assume that such a number is computable on a 32 bit machine, i.e, you do not have to detect integer overflow in your answer.

Examples

If n is	**return**	**because**
4	624	because the first four multiples of 624 are 624, 1248, 1872, 2496 and they all contain the digit 2. Furthermore 624 is the smallest number whose first four multiples contain the digit 2.
5	624	because the first five multiples of 624 are 624, 1248, 1872, 2496, 3120. Note that 624 is also the smallest number whose first 4 multiples contain the digit 2.
6	642	because the first five multiples of 642 are 642, 1284, 1926, 2568, 3210, 3852
7	4062	because the first five multiples of 4062 are 4062, 8124, 12186, 16248, 20310, 24372, 28434. Note that it is okay for one of the multiples to contain the digit 2 more than once (e.g., 24372).

2. Define a **cluster** in an integer array to be a maximum sequence of elements that are all the same value. For example, in the array {3, 3, 3, 4, 4, 3, 2, 2, 2, 2, 4} there are 5 clusters, {3, 3, 3}, {4, 4}, {3}, {2, 2, 2, 2} and {4}. A **cluster-compression** of an array replaces each cluster with the number that is repeated in the cluster. So, the cluster compression of the previous array would be {3, 4, 3, 2, 4}. The first cluster {3, 3, 3} is replaced by a single 3, and so on.

Write a function named **clusterCompression** with the following signature

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

int[ ] clusterCompression(int[ ] a)

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

int *clusterCompression(int a[ ], int len) where len is the length of the array.

The function returns the cluster compression of the array a. The length of the returned array must be equal to the number of clusters in the original array! This means that someplace in your answer you must dynamically allocate the returned array.

In Java or C# you can use

int[ ] result = new int[numClusters];

In C or C++ you can use

int *result = (int *)calloc(numClusters, sizeof(int));

Examples

a is	then function returns
{0, 0, 0, 2, 0, 2, 0, 2, 0, 0}	{0, 2, 0, 2, 0, 2, 0}
{18}	{18}
{}	{}
{-5, -5, -5, -5, -5}	{-5}
{1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1}	{1, 2, 1}
{8, 8, 6, 6, -2, -2, -2}	{8, 6, -2}

3. Define an array to be a **railroad-tie** array if the following three conditions hold

a. The array contains at least one non-zero element

b. Every non-zero element has exactly one non-zero neighbor

c. Every zero element has two non-zero neighbors.

For example, {1, 2, 0, 3, -18, 0, 2, 2} is a railroad-tie array because

a[0] = 1 has exactly one non-zero neighbor (a[1]) a[1] = 2 has exactly one non-zero neighbor (a[0])

a[2] = 0 has two non-zero neighbors (a[1] and a[3])

a[3] = 3 has exactly one non-zero neighbor (a[4])

a[4] = -18 has exactly one non-zero neighbor (a[3])

a[5] = 0 has two non-zero neighbors (a[4] and a[6])

a[6] = 2 has exactly one non-zero neighbor (a[7])

a[7] = 2 has exactly one non-zero neighbor (a[6])

The following are not railroad-tie arrays

{1, 2, 3, 0, 2, 2}, because a[1]=2 has two non-zero neighbors.

{0, 1, 2, 0, 3, 4}, because a[0]=0 has only one non-zero neighbor (it has no left neighbor)

{1, 2, 0, 0, 3, 4}, because a[2]=0 has only one non-zero neighbor (a[1])

{1}, because a[0]=1 does not have any non-zero neighbors.

{}, because the array must have at least one non-zero element

{0}, because the array must have at lease one non-zero element.

Write a function named **isRailroadTie** which returns 1 if its array argument is a railroad-tie array; otherwise it returns 0.

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

int isRailroadTie(int[ ] a)

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

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

More examples:

if a is	**return**
{1, 2}	1
{1, 2, 0, 1, 2, 0, 1, 2}	1
{3, 3, 0, 3, 3, 0, 3, 3, 0, 3, 3}	1

{0, 0, 0, 0}	0 (must have non-zero element)
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	0 (a[1] has two non-zero neighbors)
{1, 3, 0, 3, 5, 0}	0 (a[5] has no right neighbor)

This exam has three questions. You have two hours to complete it. Please format your answers so that blocks are indented. This makes it easier for the grader to read your answers. And do your own work!

1. Define the **fullness quotient** of an integer n > 0 to be the number of representations of n in bases 2 through 9 that have no zeroes anywhere after the most significant digit. For example, to see why the fullness quotient of 94 is 6 examine the following table which shows the representations of 94 in bases 2 through 9.

base	**representation of 94**	**because**
2	1011110	$2^6 + 2^4 + 2^3 + 2^2 + 2^1 = 94$
3	10111	$3^4 + 3^2 + 3^1 + 3^0 = 94$
4	1132	$4^3 + 4^2 + 3*4^1 + 2*4^0 = 94$
5	334	$3*5^2 + 3*5^1 + 4*4^0 = 94$
6	234	$2*6^2 + 3*6^1 + 4*6^0 = 94$
7	163	$1*7^2 + 6*7^1 + 3*7^0 = 94$
8	136	$1*8^2 + 3*8^1 + 6*8^0 = 94$
9	114	$1*9^2 + 1*9^1 + 4*9^0 = 94$

Notice that the representations of 94 in base 2 and 3 both have 0s somewhere after the most significant digit, but the representations in bases 4, 5, 6, 7, 8, 9 do not. Since there are 6 such representations, the fullness quotient of 94 is 6.

Write a method named **fullnessQuotient** that returns the fullness quotient of its argument. If the argument is less than 1 return -1. Its signature is

int fullnessQuotient(int n)

Hint: use modulo and integer arithmetic to convert n to its various representations

Examples:

if n is	return	Because
1	8	Because all of its representations do not have a 0 anywhere after the most significant digit: 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9
9	5	Because 5 of the representations (4, 5, 6, 7, 8) do not have a 0 anywhere after

		the most significant digit: 2: 1001, 3: 100, 4: 21, 5: 14, 6: 13, 7: 12, 8: 11, 9: 10 All its representations have a 0 somewhere after the most significant digit:
360	0	2: 101101000, 3: 111100, 4: 11220, 5: 2420, 6: 1400, 7: 1023,8: 550, 9: 440
-4	-1	The argument must be > 0

2. Define an array to be **packed** if all its values are positive, each value n appears n times and all equal values are in consecutive locations. So for example, {2, 2, 3, 3, 3} is packed because 2 appears twice and 3 appears three times. But {2, 3, 2, 3, 3} is not packed because the 2s are not in consecutive locations. And {2, 2, 2, 3, 3, 3} is not packed because 2 appears three times.

Write a method named **isPacked** that returns 1 if its array argument is packed, otherwise it returns 0. You may assume that the array is not null

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

int isPacked(int[ ] a)

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

int isPacked(int a[ ], int len) where len is the length of the array.

Examples

a is	**then function returns**	**reason**
{2, 2, 1}	1	because there are two 2s and one 1 and equal values appear in consecutive locations.
{2, 2, 1, 2, 2}	0	Because there are four 2s (doesn't matter that they are in groups of 2)
{4, 4, 4, 4, 1, 2, 2, 3, 3, 3}	1	because 4 occurs four times, 3 appears three times, 2 appears two times and 1 appears once and equal values are in consecutive locations.
{7, 7, 7, 7, 7, 7, 7, 1}	1	because 7 occurs seven times and 1 occurs once.
{7, 7, 7, 7, 1, 7, 7, 7}	0	because the 7s are not in consecutive locations.
{7, 7, 7, 7, 7, 7, 7}	1	because 7 occurs seven times
{}	1	because there is no value that appears the wrong number of times

{1, 2, 1}	0	because there are too many 1s
{2, 1, 1}	0	because there are too many 1s
{-3, -3, -3}	0	because not all values are positive
{0, 2, 2}	0	because 0 occurs one time, not zero times.
{2, 1, 2}	0	because the 2s are not in consecutive locations

Hint: Make sure that your solution handles all the above examples correctly!

3. An array is defined to be **odd-heavy** if it contains at least one odd element and every element whose value is odd is greater than every even-valued 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.

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

Some other examples:

if the input array is	**isOddHeavy should return**
{1}	1 (true vacuously)
{2}	0 (contains no odd elements)
{1, 1, 1, 1, 1, 1}	1
{2, 4, 6, 8, 11}	1 (11, the only odd-valued element is greater than all even-valued elements.
{-2, -4, -6, -8, -11}	0 (-8, an even-valued element is greater than -11 an odd-valued element.)

This exam is two hours long and contains three questions. Please indent your code so it is easy for the grader to read it.

1. 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.

2. Define an array to be a 121 array if all its elements are either 1 or 2 and it 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, 2, 1, 2, 1, 1}	0	Because the middle does not contain only 2s.
{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}	0	Because the first element of the array is not a 1.

{1, 1, 1, 2, 2, 2, 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.

3. A binary representation of a number can be used to select elements from an array. For example,

n: $88 = 2^3 + 2^4 + 2^6$ (1011000)

array: 8, 4, 9, 0, 3, 1, 2

indexes 0 1 2 3 4 5 6

selected * * *

result 0, 3, 2

so the result of filtering {8, 4, 9, 0, 3, 1, 2} using 88 would be {0, 3, 2}

In the above, the elements that are selected are those whose indices are used as exponents in the binary representation of 88. In other words, a[3], a[4], and a[6] are selected for the result because 3, 4 and 6 are the powers of 2 that sum to 88.

Write a method named **filterArray** that takes an array and a non-negative integer and returns the result of filtering the array using the binary representation of the integer. The returned array must big enough to contain the filtered elements and no bigger. So in the above example, the returned array has length of 3, not 7 (which is the size of the original array.) **Futhermore**, **if the input array is not big enough to contain all the selected elements, then the method returns null.** For example, if n=3 is used to filter the array a = {18}, the method should return null because $3=2^0+2^1$ and hence requires that the array have at least 2 elements a[0] and a[1], but there is no a[1].

If you are using Java or C#, the signature of the function is

int[ ] filterArray(int[ ] a, int n)

If you are using C or C++, the signature of the function is

int * filterArray(int a[ ], int len, int n) where len is the length of the array a

Hint: Proceed as follows

a. Compute the size of the returned array by counting the number of 1s in the binary representation of n (You can use modulo 2 arithmetic to determine the 1s in the binary represention of n)

b. Allocate an array of the required size

c. Fill the allocated array with elements selected from the input array

Examples

if a is	and n is	return	because
{9, -9}	0	{}	because there are no 1s in the binary representation of 0
{9, -9}	1	{9}	because $1 = 2^0$ and a[0] is 9
{9, -9}	2	{-9}	because $2 = 2^1$ and a[1] is -9
{9, -9}	3	{9, -9}	because $3 = 2^0 + 2^1$ and a[0]=9, a[1]=-9
{9, -9}	4	null	because $4 = 2^2$ and there is no a[2]
{9, -9, 5}	3	{9, -9}	because $3 = 2^0 + 2^1$ and a[0]=9, a[1]=-9
{0, 9, 12, 18, -6}	11	{0, 9, 18}	because $11 = 2^0 + 2^1 + 2^3$ and a[0]=0, a[1]=9, a[3]=18

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
{1,1,1,1,1,1,1,1,1}	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

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 $87 \mathrel{!=} 8^3 + 7^3$
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

Sample 3 MUM entrance exam solutions

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, 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!

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.

Examples

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	0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1	because each 9 digit is encoded as 0,0,0,0,0,0,0,0,0,1.

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

MUM test entrance exam solutions

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.