sample 2 question collections of MUM university MS computer science

Posted by: rupaknpl Posted date: **February 16, 2013** In: Other | comment : 1

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:

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

Similarily, 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

READ PREVIOUS POST:
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...

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=**2**6**244** and 162*162*162 = **42**51**5**28 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, 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
360	0	All its representations have a 0 somewhere after the most significant digit: 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 consec utive 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 != $8^3 + 7^3$
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

‹ Previous Next ›

Related posts

Scubattokki.com checkout system and many vqmod conflicts removed successfully

2000th title of Packt Publishing are launching an exciting campaign until 26th Mar 2014

How Reward Points works in opencart 1.5+ eCommerce site

MUM test entrance exam solutions

Comments (1)

naruto

November 13 at

After controlling 9 tail why isn't Naruto turning into 9 tail when he uses his power!

Reply

Leave a Comment

Name*

Email*

Website

Submit Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.