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

public static void main()

{

largestPrimeFactor(10);

largestPrimeFactor(6936);

largestPrimeFactor(1);

}

public int largestPrimeFactor(int n)

{

int largestPF = 0;

ArrayList primes = getPrimeNumLessThan(n);

if (n > 1)

{

foreach (int prime in primes)

{

while (n % prime == 0)

{

n /= prime;

if (prime > largestPF)

largestPF = prime;

}

}

}

return largestPF;

}

ArrayList getPrimeNumLessThan(int n)

{

ArrayList primeNumbers = new ArrayList();

for (int i = 2; i < n; i++)

{

bool divisible = false;

foreach (int number in primeNumbers)

{

if (i % number == 0)

{

divisible = true;

break;

}

}

if (!divisible)

primeNumbers.Add(i);

}

return primeNumbers;

}

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 0.

Hint: proceed as follows:

compute the total number of prime factors including duplicates

allocate an array to hold the prime factors

populate the allocated array with the prime factors. The elements of the array when multiplied together should equal the number.

Do not hard-code the size of the returned array!!

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

public static void main()

{

encodeNumber(2);

encodeNumber(6);

encodeNumber(14);

encodeNumber(24);

encodeNumber(1200);

encodeNumber(1);

encodeNumber(-18);

}

public int[ ] encodeNumber(int n)

{

if (n <= 1)

return null;

else

{

int divisor = 2;

ArrayList arrPrimFactor = new ArrayList();

while (n > 1)

{

while (n % divisor == 0)

{

n /= divisor;

arrPrimFactor.Add(divisor);

//output += divisor + " " + (n > 1 ? "\* " : "");

}

++divisor;

}

int[] encodedNumber = new int[arrPrimFactor.Count];

for (int i = 0; i < arrPrimFactor.Count; i++)

encodedNumber[i] = int.Parse(arrPrimFactor[i].ToString());

return encodedNumber;

}

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 complete this code by replacing the '?'s with appropriate values.

matches = ?;

i = ?;

k = ?;

}

}

// 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 completing the code for the last else statement. Your answer just has to include this code, you do not have to write the entire function.

Examples

if a is and pattern is then function returns 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 and it does not

{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. Hint: make sure your code handles this situation correctly!

public static void main()

{

matchPattern(new int[] { 1, 1, 1, 1, 1 }, 5, new int[] { 1 }, 1);

matchPattern(new int[] { 1, 1, 2, 2, 2,2 }, 6, new int[] { 1,2 }, 2);

matchPattern(new int[] { 1, 1, 3 }, 3, new int[] { 1, 2 }, 2);

matchPattern(new int[] { 1, 2 }, 2, new int[] { 1, 2, 3 }, 3);

matchPattern(new int[] { 1, 1,2,2,2,2,3 }, 7, new int[] { 1, 3 }, 2);

matchPattern(new int[] { 1, 1,1,1 }, 4, new int[] { 1, 2 }, 2);

matchPattern(new int[] { 1, 1,1,1,2,2,3,3 }, 8, new int[] { 1, 2}, 2);

matchPattern(new int[] { 1, 1,10,4,4,3 }, 6, new int[] { 1, 4, 3 }, 3);

}

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 complete this code by replacing the '?'s with appropriate values.

matches = 0;

i = i - 1;

k = k + 1;

}

}

// 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;

}

//////////////////////////////////////////////////////////////////////////////////////////

4. private void Form3\_Load(object sender, EventArgs e)

{

MessageBox.Show(decodedArray(new int [] { }).ToString());

}

private int decodedArray(int[] a)

{

if (a.Length == 0)

return 0;

string decodedNo = "";

if(a[0] < 0)

decodedNo = "-" + decodedNo;

for (int i = 0; i < a.Length-1; i++)

{

decodedNo = decodedNo + Math.Abs(a[i] - a[i + 1]).ToString();

}

return int.Parse(decodedNo);

}

//////////////////////////////////////////////////////////////////////////////////////////

5. private int isPrimeHappy(int n)

{

if (n <= 2)

return 0;

else

{

ArrayList PrimeNumLessThan = getPrimeNumLessThan(n);

if (PrimeNumLessThan.Count > 0)

{

int sumofPrimes = 0;

for (int i = 0; i < PrimeNumLessThan.Count; i++)

{

sumofPrimes = sumofPrimes + int.Parse(PrimeNumLessThan[i].ToString());

}

if (sumofPrimes % n == 0)

return 1;

else

return 0;

}

else

return 0;

}

}

private ArrayList getPrimeNumLessThan(int n)

{

ArrayList primeNumbers = new ArrayList();

for (int i = 2; i < n; i++)

{

bool divisible = false;

foreach (int number in primeNumbers)

{

if (i % number == 0)

{

divisible = true;

break;

}

}

if (!divisible)

primeNumbers.Add(i);

}

return primeNumbers;

}

////////////////////////////////////////////////////////////////////////////////////////////////

6. private void Form1\_Load(object sender, EventArgs e)

{

label1.Text = isOnionArray(new int[] { 1, 2, 19, 4, 5 }) + " , ";

label1.Text = label1.Text + isOnionArray(new int[] { 1, 2, 3, 4, 15 }) + " , ";//;

label1.Text = label1.Text + isOnionArray(new int[] { 1, 3, 9, 8 }) + " , ";//;

label1.Text = label1.Text + isOnionArray(new int[] { 2 }) + " , ";//;

label1.Text = label1.Text + isOnionArray(new int[] { }) + " , ";//;

label1.Text = label1.Text + isOnionArray(new int[] { -2, 5, 0, 5, 12 }) + " , ";//;

}

int isOnionArray(int[] a)

{

int k = a.Length - 1;

while (k >= 0)

{

for (int j = 0; j < a.Length; j++)

{

if (j >= 0 && k >= 0 && j + k + 1 == a.Length && j != k)

if (a[j] + a[k] > 10)

return 0;

}

k--;

}

return 1;

}

int isOnion(int[] a)

{

ArrayList subArray = GetSubArray(a);

foreach(int[] sub in subArray)

{

int i = 0, j = sub.Length - 1;

if (i >= 0 && j >= 0 && i + j + 1 == sub.Length && i != j)

if (sub[i] + sub[j] > 10)

return 0;

}

return 1;

}

private ArrayList GetSubArray(int[] a)

{

ArrayList arr = new ArrayList();

if (a.Length < 2)

{

arr.Add(a);

return arr;

}

else

{

int i = 0, j = a.Length - 1;

while (j >= a.Length / 2)

{

int[] subArray = new int[] { a[i], a[j] };

i++;

j--;

arr.Add(subArray);

}

}

return arr;

}

//////////////////////////////////////////////////////////////////////////////////////////////

7 . private int[] doIntegerBasedRounding(int[] a, int n)

{

if (n > 0)

{

for (int i = 0; i < a.Length; i++)

{

if (a[i] > 0)

{

int smallFactor = (n/a[i])\* a[i];

if ((m[1] - a[i]) > (a[i] - m[0]))

a[i] = m[0];

else

a[i] = m[1];

}

}

}

return a;

}