SOURCE CODE (a)

```
#include <iostream>
#include <cstring>

using namespace std;

bool primes[1001]; // array will store all the prime numbers from 0 to 1000. Primes will be marked as true
// seive of eratosthenes to find all primes les sthan 1000
void primeSieve()
{
    // initialize all entries of 'primes' as true
    memset(primes, true, sizeof(primes));

    // traverse all numbers and mark their multiples as false
    // do not traverse false numbers
    for (int p = 2; p * p < 1001; p++)
    {
        if (primes[p])
        {
            for (int i = p * 2; i < 1001; i = i + p)
                primes[i] = false;
        }
    }
}

// function to return true if number is sphenic
bool isSphenic(int number)
{
    int arr1[8] = {0}; // array to store 8 divisors
    int count = 0;    // track the number of divisors
    int j = 0;
    for (int i = 1; i <= number; i++)
    {
        if (number % i == 0 && count < 9)
        {
            count++;
            arr1[j++] = i;
        }
    }
```

```
    // if there are 8 divisors and the divisors are primes, the number is sphenic
    if (count == 8 && (primes[arr1[1]] && primes[arr1[2]] && primes[arr1[3]]))
        return true;
    return false;
}


// main function
int main()
{
    int n;
    cout << "Enter n : ";
    cin >> n;
    // generate all prime numbers
    primeSieve();
    cout << "The sphenic numbers between 1 and " << n << " are:\n";
    for (int i = 1; i <= n; i++)
    {
        if(isSphenic(i))
            cout << i << "\n";
    }


}
```

OUTPUT

```
PS D:\projects and code\c++\chegg> g++ .\main.cpp
PS D:\projects and code\c++\chegg> ./a
Enter n : 50
The sphenic numbers between 1 and 50 are:
30
PS D:\projects and code\c++\chegg>
```

PSEUDOCODE (b)

1. array = {12, 13, 11, 9, 21, 45}
2. max = -99
3. temp = 0
4. for i from 0 to array.length
5.   findMax(temp, array[i])
6.   if(max < temp)
7.   max = temp
8. print("Max number is ", max)

SOURCE CODE (b)

```
#include <iostream>
```

```
using namespace std;


// function definition completed

void findMax(int &max, int a)

{

    if(a > max)

        max = a;

}


// implementing the pseudocode

int main()

{

    int array[6] = {12, 13, 11, 9, 21, 45};

    int max = -999;

    int temp;


    for (int i = 0; i < 6; i++)

    {

        findMax(temp, array[i]);

        if(max < temp)

            max = temp;

    }


    // print the output

    cout << "The max from the given sequence is: " << max;

    return 0;

}
```

OUTPUT

```
PS D:\projects and code\c++\chegg> g++ .\main.cpp
PS D:\projects and code\c++\chegg> ./a
The max from the given sequence is: 45
PS D:\projects and code\c++\chegg>
```

COMPLEXITY

The complexity will be O(n) where n is the length of the sequence. This is because the **findMax()** function takes constant time, and traversing the sequence and finding the max form there takes **n** iterations, n is the size of the squence.

PSEUDOCODE (c)

```
1. sum = 1
2. for i from 2 to root(n)
3.         if(number % i == 0)
4.                 if(i * i != number)
5.                         sum += (i + number/i)
6.                 else
7.                         sum += i
8.         if(sum == number)
9.                 print("Number is perfect")
```

SOURCE CODE TO FIND PERFECT NUMBERS FROM I TO N

```
#include <iostream>
using namespace std;

// sets flag to true if number is perfect
void isPerfect(int number, bool &flag)
{
    flag = false;
    int sum = 1;

    // Find all divisors of number
    for (int i = 2; i * i <= number; i++)
    {
        if (number % i == 0)
        {
            if (i * i != number)
                sum = sum + i + number / i;
            else
                sum = sum + i;
        }
    }
    // If sum is equal to number, set flag as true
    if (sum == number && number != 1)
        flag = true;
}

// Driver program
int main()
{
    int n;
    cout << "Enter n: ";
    cin >> n;
    cout << "Below are all perfect numbers from 1 till " << n << "\n";
    for (int i = 1; i <= n; i++)
    {
        bool flag = false;
        isPerfect(i, flag);
        if (flag)
            cout << i << "\n";
    }
```

```
    return 0;

}
```

OUTPUT

```
PS D:\projects and code\c++\chegg> g++ .\main.cpp
PS D:\projects and code\c++\chegg> ./a
Enter n: 100
Below are all perfect numbers from 1 till 100
6
28
PS D:\projects and code\c++\chegg>
```

COMPLEXITY

In this program, finding the divisors of a number **n** takes squareRoot(n) amount of iterations, hence finding if a number is a perfect number or not takes squareRoot(n) time. FInding the first n natural numbers which are perfect numbers hence takes **squareRoot(1) + squareRoot(2) + ... + squareRoot(n)** amount of time. Upon computation, we find that it results in $\frac{2}{3}\cdot\left((n-2)\sqrt{n+1} \;-\; 2\sqrt{2} \right) + 1$

Hence, for large values of n, the complexity of this program is **O(n*squareRoot(n))**