# NUMBER THEORY

# Lesson -6

## Topics to be Covered:

1. Prime factors of LCM of array elements
2. Maximum Gcd Pair
3. Program to find last digit of nth Fibonnaci Number
4. Last non-zero digit of a factorial (tricks)
5. Kadane's Algorithm
6. Inclusion - Exclusion Principle
7. The PigeonHole Principle

# 1. Prime factors of LCM of array elements

**Given an array of elements find all the prime factors of the LCM of the given numbers.**

### Example 1:

**Input:**

N = 8

Arr = {1 , 2, 3, 4, 5, 6, 7, 8}

**Output:** 2 3 5 7

**Explanation:** LCM of N elements is 840

and 840 = 2*2*2*3*5*7.so prime factors

would be 2, 3, 5, 7.

### Example 2:

**Input:**

N = 4

Arr = {20, 10, 15, 60}

**Output:** 2 3 5

**Explanation:** LCM of N elements is 60

and 60 = 2*2*3*5.so prime factors

would be 2, 3, 5.

Code:

```
vector<int> primeFactorLcm(  int arr[], int n ){
        bool factors[500001];
        memset(factors,false,sizeof(factors));
        vector<int> ans;

        for(int i=0;i<n;i++)
        {
            for(int j=2;j*j<=arr[i];j++)
            {
                    if(arr[i]%j==0)
                    {
                        factors[j]=true;
                        while(arr[i]%j==0)
                        {
                            arr[i]/=j;
                        }
                    }
                }
                 if(arr[i]>1&&arr[i]<500000)
                    {factors[arr[i]]=true;
                    }
          }
          if(factors[2])
          ans.push_back(2);
          for(int i=3;i<=500000;i+=2)
          {
              if(factors[i])
              ans.push_back(i);
          }
          return ans;
     }
```

## 2. Maximum GCD Pair

**Given an array of integers, find the pair in the array with maximum GCD. Find the maximum possible GCD.**

**Example 1:**

```
Input: N = 5, a[] = {1, 2, 3, 4, 5}
Output: 2
Explanation: Maximum gcd is GCD(2, 4)
= 2.
```

**Example 2:**

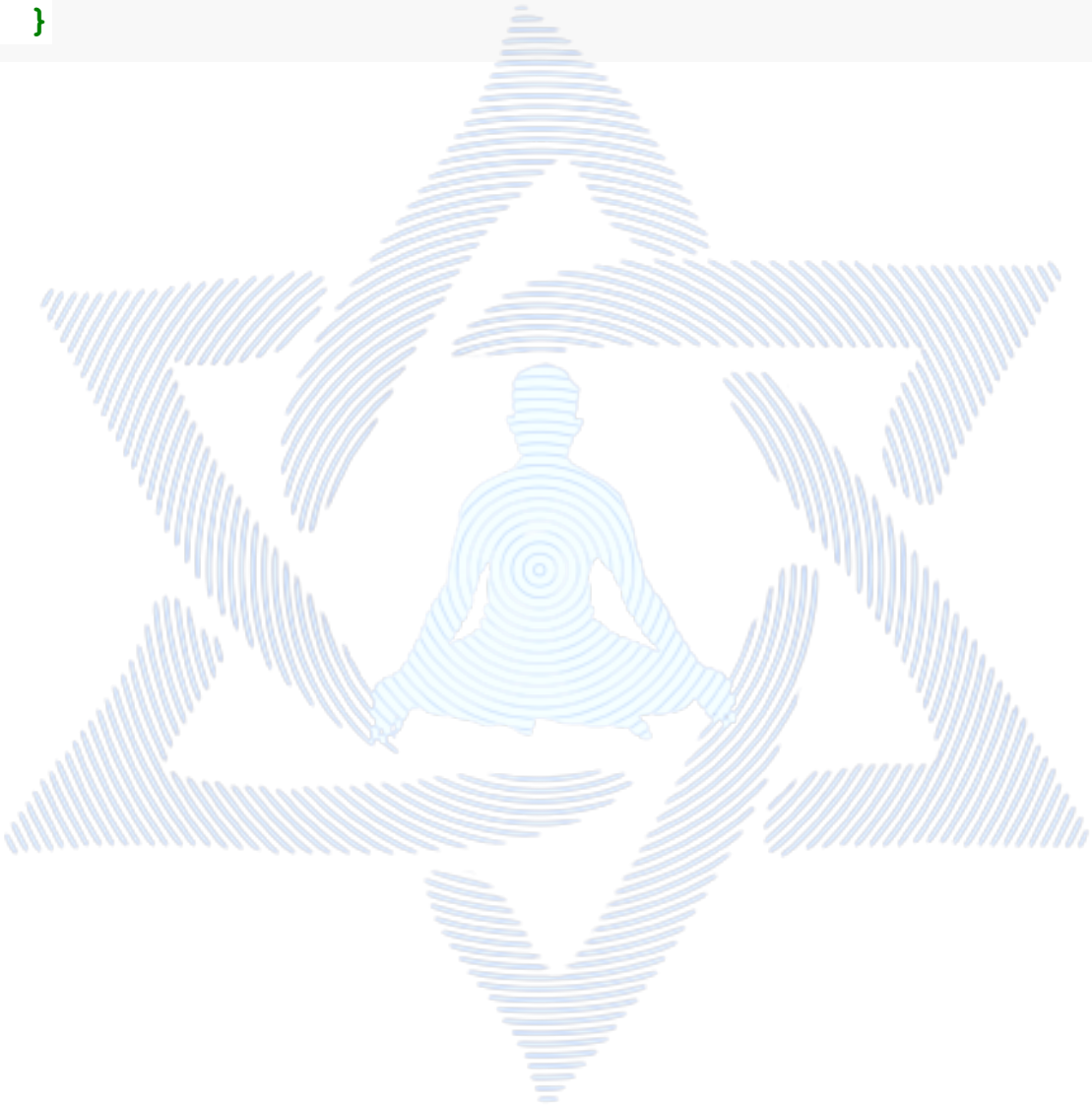
```
Input: N = 3, a[] = {3, 5, 2}
Output: 1
Explanation: Maximum gcd is 1.
```

## Code:

```
1.  int MaxGcd(int n, int arr[]) {
2.      // Calculating MAX in array
3.      int high = 0;
4.      for (int i = 0; i < n; i++)
5.          high = max(high, arr[i]);
6.
7.      // Maintaining count array
8.      int count[high + 1] = {0};
9.      for (int i = 0; i < n; i++)
```

```
10.         count[arr[i]]++;
11.
12.     // Variable to store the
13.     // multiples of a number
14.     int counter = 0;
15.
16.
17.     for (int i = high; i >= 1; i--)
18.     {
19.         int j = i;
20.        counter = 0;
21.         while (j <= high)
22.         {
23.             // A multiple found
24.
25.             if(count[j] >=2)
26.                 return j;
27.
28.            else if (count[j] == 1)
29.                 counter++;
30.
31.             // Incrementing potential
32.             // GCD by itself
33.             // To check i, 2i, 3i....
34.             j += i;
35.
36.             // 2 multiples found,
37.             // max GCD found
```

```
38.             if (counter == 2)
39.                 return i;
40.         }
41.     }
42.
43. }
```



# 3. Program to find last digit of nth Fibonnaci Number

**Given a number 'n', write a function that prints the last digit of n'th ('n' can also be a large number) Fibonacci number.**

### Examples :

**Input :** n = 0

**Output :** 0

**Input:** n = 2

**Output :** 1

**Input :** n = 7

**Output :** 3

### Look at the final digit in each Fibonacci number – the units digit:

**0, 1, 1, 2, 3, 5, 8,** 1**3,** 2**1,** 3**4,** 5**5,** 8**9,** 14**4,** 23**3,** 37**7,** 61**0,** 98**7,** ...

### Is there a pattern in the final digits?

0, 1, 1, 2, 3, 5, 8, 3, 1, 4, 5, 9, 4, 3, 7, 0, 7, ...

Yes!

It takes a while before it is noticeable. In fact, the series is just 60 numbers long and then it repeats the same sequence again and again all the way through the Fibonacci series – forever. **The series of final digits repeats with a cycle length of 60.**

```cpp
// Optimized Program to find last
// digit of nth Fibonacci number
#include<bits/stdc++.h>
using namespace std;

typedef long long int ll;

// Finds nth fibonacci number
ll fib(ll f[], ll n)
{
    // 0th and 1st number of
    // the series are 0 and 1
    f[0] = 0;
    f[1] = 1;

    // Add the previous 2 numbers
    // in the series and store
    // last digit of result
    for (ll i = 2; i <= n; i++)
        f[i] = (f[i - 1] + f[i - 2]) % 10;

    return f[n];
}

// Returns last digit of n'th Fibonacci Number
int findLastDigit(int n)
{
    ll f[60] = {0};

    // Precomputing units digit of
    // first 60 Fibonacci numbers
    fib(f, 60);
```

```
33.
34.     return f[n % 60];
35. }
36.
37. // Driver code
38. int main ()
39. {
40.     ll n = 1;
41.     cout << findLastDigit(n) << endl;
42.     n = 61;
43.     cout << findLastDigit(n) << endl;
44.     n = 7;
45.     cout << findLastDigit(n) << endl;
46.     n = 67;
47.     cout << findLastDigit(n) << endl;
48.     return 0;
49. }
50.
```

# 4. Last non-zero digit of a factorial

**Interesting fact: After 5 last digit of every factorial is zero.**

Given a number n, find the last non-zero digit in n!.

Examples:

**Input : n = 5**

**Output : 2**

**5! = 5 * 4 * 3 * 2 * 1 = 120**

**Last non-zero digit in 120 is 2.**

**Input : n = 33**

**Output : 4**

The **Solution** is based on below recursive formula

```
Let D(n) be the last non-zero digit in n!
If tens digit (or second last digit) of n is odd
    D(n) = 4 * D(floor(n/5)) * D(Unit digit of n)
If tens digit (or second last digit) of n is even
    D(n) = 6 * D(floor(n/5)) * D(Unit digit of n)

D(1) to D(9) are assumed to be precomputed.

Example 1: n = 27 [Second last digit is even]:
D(27) = 6 * D(floor(27/5)) * D(7)
      = 6 * D(5) * D(7)
      = 6 * 2 * 4
      = 48
Last non-zero digit is  8
Example 2: n = 33 [Second last digit is odd]:
```

```
D(33) = 4 * D(floor(33/5)) * D(3)
      = 4 * D(6) * 6
      = 4 * 2 * 6
      = 48
Last non-zero digit is  8
```

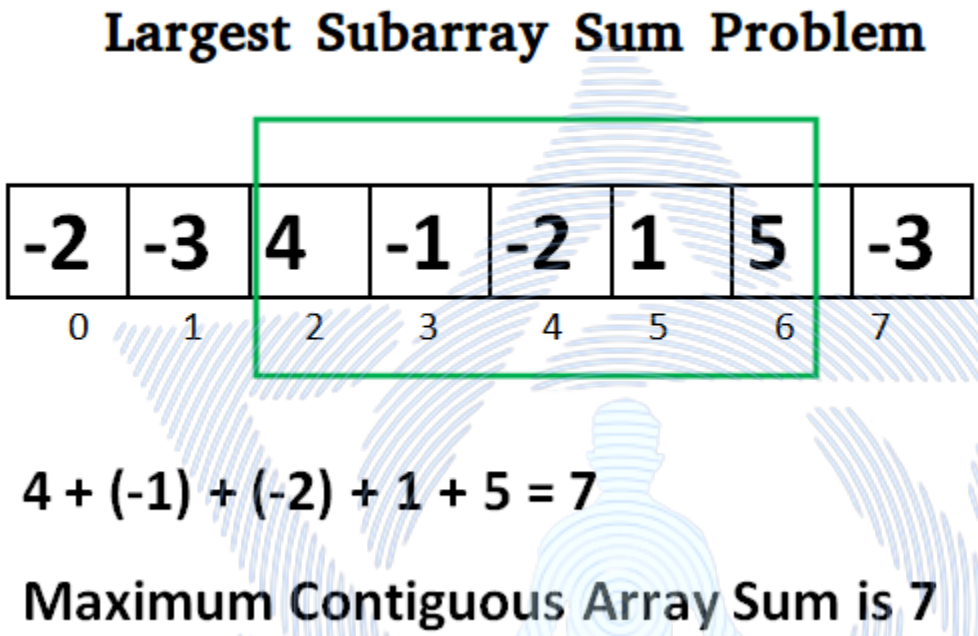
```cpp
// C++ program to find last non-zero digit in n!
#include<bits/stdc++.h>
using namespace std;

// Initialize values of last non-zero digit of
// numbers from 0 to 9
int dig[] = {1, 1, 2, 6, 4, 2, 2, 4, 2, 8};

int lastNon0Digit(int n)
  {
    if (n < 10)
        return dig[n]
    // Check whether tens (or second last) digit
    // is odd or even
    // If n = 375, So n/10 = 37 and (n/10)%10 = 7
    // Applying formula for even and odd cases.
    if (((n/10)%10)%2 == 0)
        return (6*lastNon0Digit(n/5)*dig[n%10]) % 10;
    else
        return (4*lastNon0Digit(n/5)*dig[n%10]) % 10;
  }
  // Driver code
  int main()
  {
    int n = 14;
    cout << lastNon0Digit(n);
    return 0;
  }
```

## 5. Kadane's Algorithm

**Write an efficient program to find the sum of contiguous subarray within a one-dimensional array of numbers that has the largest sum.**




**Kadane's Algorithm:**

**Initialize:**

```
max_so_far = INT_MIN
max_ending_here = 0
```

**Loop for each element of the array**

```
(a) max_ending_here = max_ending_here + a[i]
(b) if(max_so_far < max_ending_here)
        max_so_far = max_ending_here
(c) if(max_ending_here < 0)
        max_ending_here = 0
```

return max_so_far

Code:

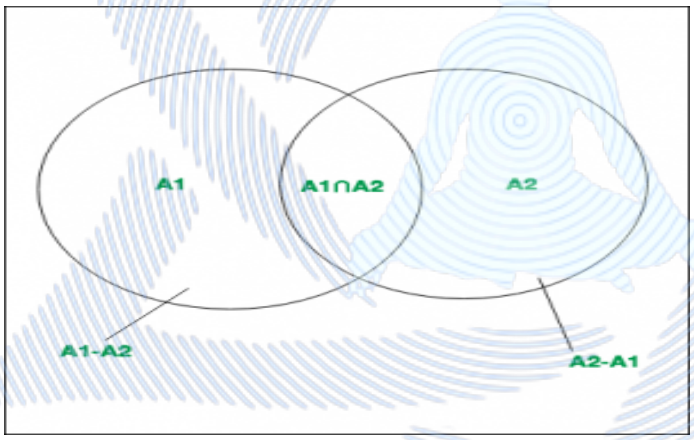
```
1.  #include<iostream>
2.  #include<climits>
3.  using namespace std;
4.
5.  int maxSubArraySum(int a[], int size)
6.  {
7.     int max_so_far = INT_MIN, max_ending_here = 0;
8.
9.     for (int i = 0; i < size; i++)
10.       {
11.          max_ending_here = max_ending_here + a[i];
12.          if (max_so_far < max_ending_here)
13.             max_so_far = max_ending_here;
14.
15.          if (max_ending_here < 0)
16.             max_ending_here = 0;
17.       }
18.       return max_so_far;
19.    }
20.    int main()
21.    {
22.       int a[] = {-2, -3, 4, -1, -2, 1, 5, -3};
23.       int n = sizeof(a)/sizeof(a[0]);
24.       int max_sum = maxSubArraySum(a, n);
```

```
25.     cout << "Maximum contiguous sum is " << max_sum;
26.     return 0;
27. }
```

## 6. Inclusion-Exclusion and its various Applications

In the field of Combinatorics, it is a counting method used to compute the cardinality of the union set. According to basic Inclusion-Exclusion principle:

- For 2 finite sets $A_1$ and $A_2$ , which are subsets of Universal set, then ($A_1$-$A_2$) , ($A_2$- $A_1$) and $(A_1 \bigcap A_2)$ are disjoint sets.




Here it can be said that:

$$|(A_1 - A_2) \bigcup (A_2 - A_1) \bigcup (A_1 \bigcap A_2)| = |A_1| - |A_1 \bigcap A_2| + |A_2| - |A_1 \bigcap A_2| + |A_1 \bigcap A_2|$$

$$|A_1 \bigcup A_2| = |A_1| + |A_2| - |A_1 \bigcap A_2|$$

Similarly, in general:

$$|A_1 \bigcup A_2 \bigcup A_3 .. \bigcup A_i| = \sum_{1 \le k \le i} |A_k| + (-1) \sum_{1 \le k_1 < k_2 \le i} |A_{k_1} \bigcap A_{k_2}| + (-1)^2 \sum_{1 \le k_1 < k_2 < k_3 \le i} |A_{k_1} \bigcap A_{k_2} \bigcap A_{k_3}| .. + (-1)^{i+1} \sum_{1 \le k_1 < k_2 < k_3 < .. k_i \le i} |A_{k_1} \bigcap A_{k_2} \bigcap A_{k_3} .. \bigcap A_{k_i}|$$

## Properties :

1. Computes the total number of elements that satisfy at least one of several properties.
2. It prevents the problem of double counting.

## Applications :

### Derangements

To determine the number of derangements( or permutations) of n objects such that no object is in its original position (like Hat-check problem).

As an example we can consider the derangements of the number in the following cases:

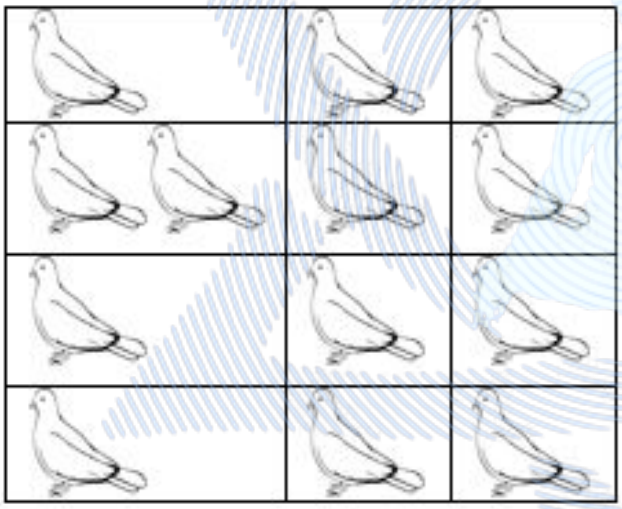
For i = 1, the total number of derangements is 0.

For i = 2, the total number of derangements is 1. This is 21 .

For i = 3, the total number of derangements is 2. These are 231 and 3 1 2.

# 7. The Pigeonhole Principle

Suppose that a flock of 20 pigeons flies into a set of 19 pigeonholes to roost. Because there are 20 pigeons but only 19 pigeonholes, a least one of these 19 pigeonholes must have at least two pigeons in it. To see why this is true, note that if each pigeonhole had at most one pigeon in it, at most 19 pigeons, one per hole, could be accommodated. This illustrates a general principle called the pigeonhole principle, which states that if there are more pigeons than pigeonholes, then there must be at least one pigeonhole with at least two pigeons in it.



Theorem –

I) If "A" is the average number of pigeons per hole, where A is not an integer then

- At least one pigeon hole contains ceil[A] (smallest integer greater than or equal to A) pigeons

- Remaining pigeon holes contains at most floor[A] (largest integer less than or equal to A) pigeons

Or

II) We can say as, if n + 1 objects are put into n boxes, then at least one box contains two or more objects.

The abstract formulation of the principle: Let X and Y be finite sets and let $f : A \to B$ be a function.

- If X has more elements than Y, then f is not one-to-one.
- If X and Y have the same number of elements and f is onto, then f is one-to-one.
- If X and Y have the same number of elements and f is one-to-one, then f is onto.

- Example – 1: In a computer science department, a student club can be formed with either 10 members from first year or 8 members from second year or 6 from third year or 4 from final year. What is the minimum no. of students we have to choose randomly from department to ensure that a student club is formed?

  Solution: we can directly apply from the above formula where,

  q1 =10, q2 =8, q3 =6, q4 =4 and n=4

  Therefore the minimum number of students required to ensure department club to be formed is

  10 + 8 + 6 + 4 – 4 + 1 = 25

- **Example – 2: A box contains 6 red, 8 green, 10 blue, 12 yellow and 15 white balls. What is the minimum no. of balls we have to choose randomly from the box to ensure that we get 9 balls of same color?**

  **Solution: Here in this we cannot blindly apply pigeon principle. First we will see what happens if we apply above formula directly.**

  **From the above formula we have get answer 47 because 6 + 8 + 10 + 12 + 15- 5 + 1 = 47**

  **But it is not correct. In order to get the correct answer we need to include only blue, yellow and white balls because red and green balls are less than 9. But we are picking randomly so we include after we apply pigeon principle.**

  **i.e., 9 blue + 9 yellow + 9 white – 3 + 1 = 25**

  **Since we are picking randomly so we can get all the red and green balls before the above 25 balls. Therefore we add 6 red + 8 green + 25 = 39**

  **We can conclude that in order to pick 9 balls of the same color randomly, one has to pick 39 balls from a box.**