ALGORITHMS

HASHING

eg You will be given array from int 0-9
Find frequency of each number in the array

Input = [1,2,9,1,4,1,3,1,5,7,8,8]

Hashing solution

initialise array with all values as 0.
arr = [0,0,0,0,0,0,0,0,0,0]
0th index indicates count of 0
1st index indicates count of 1
:
9th index indicates count of 9

```
ALGORITHM
int array[n];
for i range (n):
      Scanf ("%d",&arr[i])
int count[10];
// setting all values to 0
memset (count, 0, sizeof(count))
for i in range (n):
     count[arr[i]] ++
// printing the frequency
for i in range (10):
     print ('count of', i,'is', count[i])
```

Q. WAP to find no of pairs in array having sum = x, given array has distinct elements

```
x = input (" enter x : ")
n = input (" no of elements ")
a = eval (input (" enter list "))
int hash [1000000] = {0} // array containing
for i in range (n):          the frequency of i at
      hash [a[i]] ++          i index
int ans = 0    // variable holding pairs
for i in range (n): // x=100, a[i]=20
      int t = x - a[i]   t = 100 - 20 = 80
      if ((t > 0) && hash[t])
            ans ++      if hash[80] exist
                        then ans ++
ans = ans >> 1
print ('no of pairs is', ans)
```

MODULAR ARITHMETIC

cannot be applied to floating point number

Modular addition

$(a+b)\%m = (a\%m + b\%m)\%m$

Modular subtraction

$(a-b)\%m = (a\%m - b\%m + m)\%m$

Modular Multiplication

$(a*b)\%m = (a\%m * b\%m)\%m$

Modular Divisions

$(a/b)\%m = (a\%m * b^{-1}\%m)\%m$

Why is expansion of modulo equations required?
Before modulo operator is applied above expression will lead to INT OVERFLOW

GREATEST COMMON DIVISOR

GCD(A,B) = GCD(B, A%B)
until A%B == 0

```
CODE
int gcd (int a, int b){
    if (b == 0)
        return a
    return gcd (b, a%b)}
int main (){
   a, b = input (" Enter the no ")
   print (gcd (a,b))
```

PRIME NUMBERS

Naive Approach

We traverse through numbers from 2 to sqrt(N) and check if it is divisible

Time complexity = sqrt(N)

Seive Of Eratosthenes

The basic idea is that at each iteration we pick one prime number and eliminate all multiples of the prime number. After elimination process ends, we are left with PRIME NUMBERS

```c
#include<stdio.h>
#include <stdbool.h>
bool is_prime(int n)   //checks if a number is prime
{
    for (int i=2;i*i<=n;i++){
    if (n%i)
        return false;
    }
    return true;
}

void seive(int N){
    bool newp[N+1];
    for (int i=0;i<=N;i++){
        newp[i]=true;
//adding all the elements in th e array and marking it a pr
ime number
    }
    newp[0]=false;newp[1]=false;
//since 0 and 1 are not prime we are marking them false
    for (int i=2;i*i<=N;i++){
        if (is_prime(i)==true){
//checking if the number is prime
            for (int j=i*i;j<=N;j+=i){
//we take j=i*i because the smaller number has already been
covered in the previous iterations
                newp[j]=false;
//if prime then marking no as false
            }
        }
    }
}
```

Time Complexity

Inner loop runs for each element

if i = 2 , inner loop runs N/2 times

if i = 3 , inner loop runs N/3 times

if i = 5 , inner loop runs N/5 times

So total complexity => $N * (\frac{1}{2} + \frac{1}{3} + \dots)$

$= O(N \log \log N)$