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] exists
                        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 => $N * (\frac{1}{2} + \frac{1}{3} + \dots)$

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

BINARY SEARCH

Finds the position of a target value within a sorted array

CODE

Iterative binary search

```
int begin = 0;
int last = array.Length - 1;
int mid = 0;
```

Part #1 Initialize pointers

```
while (begin <= last) {
    mid = (begin + last) / 2;
    if (array[mid] < x) {
        begin = mid + 1;
    }
    else if (array[mid] > x) {
        last = mid - 1;
    }
    else {
        return mid;
    }
}

return -1;
```

Part #2 Search

ADVANCED BINARY SEARCH

Problem Statement: You are given an array a[1 ... N]. There is some k for which, a[i] = 0 for all i in the range 1 <= i <= k, and a[i] = 1 for all i in the range k + 1 <= i <= N. We want to find this position k which splits the groups of 0s and 1s

Consider this example first:

1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	1	1	1	1	1	1	1	1

k = 5

Naive approach: Iterate from right to left and return the value of k at which 0 is seen for the first time. This approach has a time-complexity of O(N) but can we do better?
Yes, we can.

Code

```
beg = 0, end = n-1, ans = -1
while (beg <= end):
    mid = (beg + end) / 2
    if arr[mid] == 0:
        ans = mid
        beg = mid + 1
    else
        end = mid - 1
if (ans != -1):
    print(ans)
```

Question - Aggressive Cow

- Farmer John has built a new long barn, with N (2 <= N <= 100,000) stalls. The stalls are located along a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000).
- His C (2 <= C <= N) cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, FJ wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

PSEUDO CODE

1. make a funct^n isPossible(int x) that check if a distance x is possible between each of the cows

beg = lowest value
end = highest value
mid = $\frac{beg + end}{2}$

so from the left we should try to place C no of cows

if the no of cows placed != C
return false
and end = mid - 1

If you placed all cows with minimum dist mid, this mean we can try further increasing this distance by beg = mid+1
But if the cows could not be placed then we should end = mid - 1

eg if we have [1,2,4,8,9] and cow = 3

1. beg = 1, end = 8, mid = 4
we can only place 2 cows with mid = 4
so end = 3
beg = 1, end = 3, mid = 2
we can succesfully place 3 cows with mid = 2

So we know 1 possible ans is 2. Let try increasing this dist
beg = mid + 1 = 3, end = 3, mid = 3
we can succesfully place 3 cows with mid = 3

So we know another possible ans is 3. Let try increasing this dist
beg = mid + 1 = 4, end = 3
since beg > end not possible
ans = 3

CODE

```cpp
#include <bits/stdc++.h>
using namespace std;
int N, C;
long long A[100000];
int main()
{
  int T;
  cin >> T;
  while (T--)
  {
    cin >> N >> C;
    for (int i = 0; i < N; i++)
      cin >> A[i];
    sort(A, A + N);

    // binary search
    long long low = 0, high = 1000000000, mid, pos = 0;
    while (high >= low)
    {
      mid = (high + low) / 2;
      if (chk(mid))
      {
        low = mid + 1;
        pos = mid;
      }
      else
      {
        high = mid - 1;
      }
    }
    cout << pos << endl;
  }
  return 0;
}
```

```cpp
// check if a distance of x is possible b/w each cow
bool chk(int x)
{
  // greedy approach, put each cow in the first place you can
  int cows_placed = 1, last_pos = A[0];
  for (int i = 1; i < N; i++)
  {
    if ((A[i] - last_pos) >= x)
    {
      if (++cows_placed == C)
        return true;
      last_pos = A[i];
    }
  }
  return false;
}
```

STEPS TO WRITE CODE IN C++

Use file name extension .cpp
Use only 1 header file stdc++.h
Include extra command using namespace std; after header file
Use g++ instead of gcc
int n; int t; } taking input
cin >> n >> t
cout << "Value of n:" << n << "& t is:" << t } to print out put

Vector:

A vector is an array like container that improves on the C++ array types. In particular it is not necessary to know how big you want the vector to be when you declare it, you can add new elements to the end of a vector using the ***push_back*** function. (In fact the ***insert*** function allows you insert new elements at any position of the vector, but this is a very inefficient operation -- if you need to do this often consider using a list instead).

Declaration:

1) vector <data type> vector_name : In this declaration a vector of 0 length is initialized.
2) vector <data type> vector_name(len) : In this declaration a vector of length len is initialized;
3) vector <data type> vector_name(len,val) : In this declaration the vector is of length len is initialized where elements are initialized with a value val.

Access: You can access the elements of a vector by using operator[] i.e. we can access the ith element of vector v as v[i].

Functions of vector:

1. push_back(): It push the elements into a vector from the back
2. empty(): Returns whether the container is empty.
3. size(): Returns the number of elements in the vector.
4. clear(): It is used to remove all the elements of the vector container
5. erase(): It is used to remove elements from a container from the specified position or range
6. insert(): It inserts new elements before the element at the specified position
7. front(): Returns a reference to the first element in the vector.
8. back(): Returns a reference to the last element in the vector