~~ceil(float(5)/float~~

✓ ceil((float)5, (float)3) O/P → 2

modulo

① $(-a)\%b = b-(a\%b)$
② $(-a)\%(-b) = -(a\%b)$

Some more in Binary Search (If you stuck at finding search space take l=0, h=1e9.)

Search Space:
① l = v[0], h = v[n-1]
✓② l = 0, h = n-1
③ l = 0, h = 1e9
④ l = 1, h = n (when n is given)
⑤ l = *min_element(v.begin(), v.end())
h = accumulate(v.begin(), v.end(), 0);

→ search space को हम खुद कम करते जाते है ∴ always work

⑥

→ Making the visited element negative:

```
for(int i=0; i<n; i++)
{
        if( a[abs(a[i])] >0)
            a[abs(a[i])] = -a[abs(a[i])];
}
```

a[i] <= n

Modifying Array to get SC → O(1)

	0	1	2	3	4	5
I/P →	2	4	3	2	1	3
Modified Array →	(2)	-4	-3	-2	-1	(3)

duplicate element

→ Modifying Array for O(1) SC

n=5

	0	1	2	3	4	5
I/P:	2	4	3	1	3	2
⇓	2	9	8 → 13	6 → 11	8	2

```
for(int i=0; i<n; i++)
    a[a[i]%n] = a[a[i]%n] +n;
```

duplicate element ∴ index 2 & 3 ans ← 13/5 > 1, 11/5 > 1 ← final array

2	9	(13	11)	8	2
0	1	2	3	4	5

(XIV) N^{th} Magical Number

→ A positive integer is magical if it is divisible by either a or b

→ return n^{th} magical number.

```
int nthmagical(int n, int a, int b)
{
    long l = min(a,b), h = (long)n * min(a,b);
    long mod = 1e9+7, A = (long)a, B = (long)Bb;
    while(A!=B)
    {
        If(A>B)
            A = A-B;
        else
            B = B-A;
    }
    long long int lcm = (a*b)/A;      (a*b = lcm * gcd)
    while(l<=h)
    {
        long m = (l+h)/2;
        if((m/a) + (m/b) - (m/lcm) < n)
            l = m+1;
        else
            h = m-1;
    }
    return l%mod;
}
```

gcd program

We are checking can mid be answer

I/P → n=4, a=2, b=3

O/P → 6

Say a=2, b=3 & m=10

mid

$$\frac{m}{a} + \frac{m}{b} - \frac{m}{lcm}$$

$$\frac{10}{2} + \frac{10}{3} - \frac{10}{6} = 5+3-1 = 7$$

7 no. ≤ 10 are either divisible by 2 or 3

they are 2 3 4 6 8 9 10 → 7 no's

(XV) Find peak element:

- A peak element is an element that is strictly greater than its neighbours.
- If array contain multiple peak return index of any pe.

```
int findPeak(vector<int> &v)
{
    int l=0, h=v.size()-1;
    while(l<=h)
    {
        int m=(l+h)/2;
        if((m==0 || v[m]>=v[m-1]) &&
           (m==v.size()-1 || v[m]>=v[m+1]))
            return m;

        else if( m>0 && v[m-1]>=v[m])
            h=m-1;
        else
            l=m+1;
    }
    return 1; → no use
}
```

pe

[5 4 3 2]

[3 4 5 6 7]

pe

** end points case

I/P → [1,2,1,3,5,6,4]

O/P → 5 (or 1)

(XVI) Sqrt(x)

x=8 $\sqrt{8}$ = 2.82842 return 2

```
int f(int x)
{   double l=1, h=x;
    while((h-l)>0.000001)
    {   double m=(l+h)/2;
        if( m*m<x)
            l=m;
        else h=m;
    }
    return h;
```

`int len = sizeof(arr)/sizeof(arr[0])`

↑ to calculate length of arr

classmate Date Page

Points in Binary Search

① ~~If start & end are all approx INT_MAX~~, then

② $(l+h)/2 \Rightarrow l+(h-l)/2$

② $(l<=h) \Rightarrow (h-l)>1$

} If overflow occur.

- use long long if overflow occurs
- also use if the array index got negative
- check for bounday conditions.

Modulo can't be performed on double & float datatype
To perform on d & float we use fmod() function

```
double a=9.7, b=2.3;
cout<< fmod(a,b);
```

o/p → 0.5

Array Remainings

→ Given array find all triplets a[i]+a[j]+a[k] ==0

(33) 3 Sum: (Sum 0)

(I) Brute force: → I/P → [-1,0,1,2,-1,-4]
three for loop O/P → [[-1,-1,2],[-1,0,1]]

```
for(i=0 to n-1)
  for(j=i+1 to n-1)
    for(k=j+1 to n-1)
      if(a[i]+a[j]+a[k]==0)
```

TC → O(n³ log m) (log m → to insert in set; m → no. of triplets)

store it in set to avoid duplicates

SC → O(m)

(II) Using Hashing: we will create a loop on a & b and to find c we will be using Hashing.

a+b+c=0

c=-(a+b)

→ if -(a+b) exist in Hash map then we can say a,b,-(a+b) is our triplet

(III) Using 2-pointer Approach:

→ first sort array

[-2,-2,-1,-1,-1,0,0,0,2,2,2]

i → -2, l → 0, h → 2

(-2+0+2==0) Any

I → remember to avoid duplicates we increase l & decrease h until new values are assigned to v[l] & v[h]

II → And also to avoid same value to i we increase value to i until new value is assigned to v[i].

```
vector<vector<int>> threeSum(vector<int> &v)
{
    vector<vector<int>> ans;
    sort(v.begin(), v.end());
    for(int i=0; i<v.size(); i++)
    {
        int sum = -v[i], s = i+1, e = v.size()-1;
        while(s<e)
        {
            if(v[s]+v[e]<sum)
                s++;
            else if(v[s]+v[e]>sum)
                e--;
            else
            {   vector<int> ds = {v[i], v[s], v[e]};
                ans.push_back(ds);
                while(s<e && v[s]==ds[1]) s++;   I
                while(s<e && v[e]==ds[2]) e--;
            }
        }
        while(i<v.size()-1 && v[i+1]==v[i])   II
            i++;
    }
    return ans;
}
```

TC → $O(N^2)$

SC → $O(1)$ no extra space other than returning vector.

(34) 4 Sum:

- Given array, & target find $a[i]+a[j]+a[k]+a[l] = target$.

I/P → [1,0,-1,0,-2,2] t = 0

O/P → [[-2,-1,1,2], [-2,0,0,2], [-1,0,0,1]]

(I) ~~3 Sum approach + BS~~

3 pointer (i,j,k) + BS : first sort() array

4 3 3 4 7 8 4 3

i j k

3 3 3 4 4 4 7 8 → sorted

t = 16

now we will find (t - v[i] - v[j] - v[k]) using BS from k+1 to v.size()-1

TC → $O(N^3 \log N)$

→ Similar to previous approach

(II)
```
vector<vector<int>> fourSum(vector<int>& v, int t)
{
    vector<vector<int>> ans;
    sort(v.begin(), v.end());
    for(int i=0; i<v.size(); i++)
    {
        for(int j=i+1; j<v.size; j++)
        {
            int l=j+1, h=v.size()-1, sum = t-v[i]-v[j];
            while(l<h)
            {
                if(v[l]+v[h] > sum)
                    h--;
                else if(v[l]+v[h] < sum)
                    l++;
                else
```

```
                { vector<int> ds = {v[i], v[j], v[l], v[h]};
                  ans.push_back(ds);
                  while(l<h && ds[2]==v[l]) l++;
                  while(l<h && ds[3]==v[h]) h--;
                }
            }
            while(j<v.size()-1 && v[j]==v[j+1]) j++;
        }
        while(i<v.size()-1 && v[i]==v[i+1]) i++;
    }
    return ans;
}
```

TC → $O(n^3)$
SC → $O(1)$

(35) 3 Sum Closest:

- Given array & target, find three numbers in array such that the sum is closest to equal to target.

I/P → a[-1, 2, 1, -4] t=1
O/P → 2 (-1+2+1)

```
int threeSum(vector<int>& v, int t)
{
    int ans = v[0]+v[1]+v[2];
    sort(v.begin(), v.end());
    for(int i=0; i<v.size()-2; i++)
    {
        if(i>0 && v[i]==v[i-1]) continue;
        int l=i+1, h=v.size()-1;
```

```
while (l<h)
{
    if(v[l]+v[h]+v[i]==t)
        return t;
    if( abs(v[i]+v[l]+v[h]-t) < abs(ans-t))
        ans = v[i]+v[l]+v[h];
    if(v[l]+v[h]+v[i]>t)
        h--;
    else
        l++;
}
}
return ans;
}
```