(13) Search in a 2D matrix:

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 10 & 11 & 16 & 20 \\ 23 & 30 & 34 & 60 \end{bmatrix}$$

Brute force approach:
loop over every element & then find the req element

~~Better approach~~: (binary search)

[1 3 5 7] [10 11 16 20] [23 30 34 60]
↑ low ↑ high

```
while (low<=high)
{
  int middle = (low+high)/2;
  if (mat[middle/m][middle%m]<target)
        high = middle-1;
  if (mat[middle/m][middle%m]>target)
        low = middle+1;
  else
        return true;
}
```

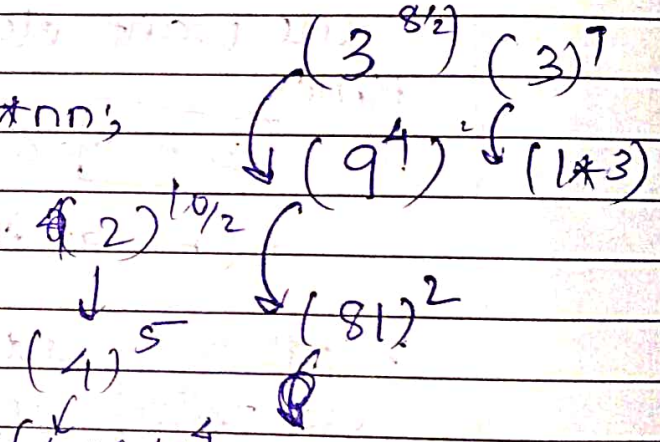
(16) Pow(x,n)

Brute force approach:

```
double ans = 1.0;
for(i=0 ---- n)
{
  ans = ans * x;
}
```

Binary Exponentiation:

```
double myPow(double x, int n)
{ double ans = 1.0;
  long long nn = n;
  if(nn < 0) nn = -1 * nn;
  while(nn){
   if(nn % 2){
   ans = ans * x;
    nn = nn/2;
   }else
   {
   x = x * x;
   nn = nn/2
   }
  }
  if(n<0) ans = double(1.0)/double(ans);
  return ans;
```



(14) Find majority element that occurs more than N/2

Brute force approach:

↳ Find freq of each element & compare with N/2
if > N/2 it is candidate element

→

Best Approach:
Moore's algorithm of candidate element:

```
int candidate, count=0;
while(i<n)
{
  if (count==0)
  {
    candidate = v[i];
    count++;
  }
  else if (candidate == v[i]) count++;
  else count--;
}
int count=0;
for(i=0—n)
if (v[i] == candidate) cnt++;
if((cnt > (n/2)) return el;
return -1;
}.
```

(15) Find elements that occurs more than N/3 times

Brute force:

→ Find freq of element & compare with N/3
If > N/3 add to vector (ans)

→

Best approach

~~extended Boye moore's Algo:~~

How many Integers at max be there in answer?

$$n \left\lfloor \frac{8}{3} \right\rfloor = 2$$

We want element > 2, means at minimum 3 times.

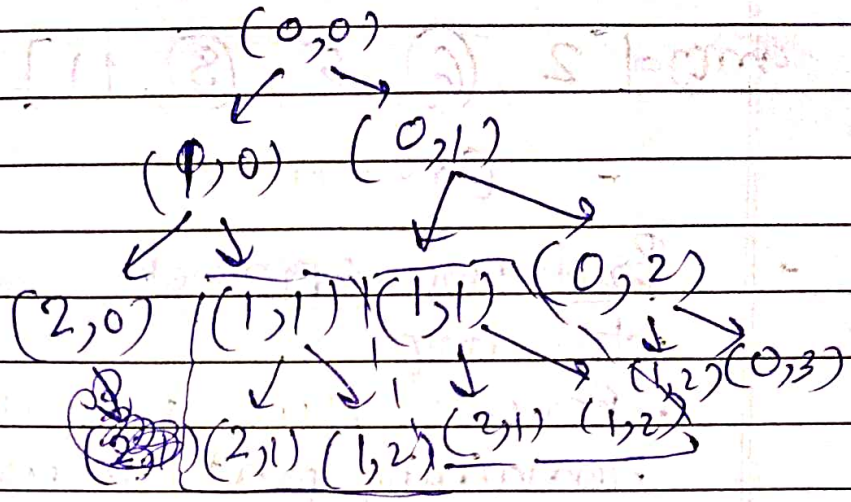
$$\boxed{3+3} = 6+3 = (9) > 8$$

So at max there will be 2 elements.

```
∴ cnt1 = 0 , cnt2 = 0
  ele1 = Int min  ele2 = Int min;
{ for (i=0 → n)
    if (cnt1 == 0 && ele2 != arr[i])
    { cnt1 = 1; ele1 = arr[i]; }
    else if ( cnt2 == 0 && ele1 != arr[i])
    { cnt2 = 1; ele2 = arr[i]; }
    else if (arr[i] == ele1) cnt1++;
    else if (arr[i] == ele2) cnt2++;
    else
    { cnt1--; cnt2--; }
}
```

(7) Grid Unique path:

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)



Recursive approach:

dp approach:

```
countpaths( )
{ if (i == (n-1) && j == (m-1)) return 1;
  if (i >= n || j >= m) return 0;
  if (dp[i][j] != 0) return dp[i][j];
  else return dp[i][j] = countpaths(i+1, j, n, m, dp);
                       + countpaths(i, j+1, n, m, dp);
}

uniquePaths( )
{
  vector<vector<int>> dp(m+1, vector<int>(n+1, -1));
  int num = countPaths (0, 0, m, n, dp);
  if (m == 1 && n == 1)
      return num;
  return dp[0][0];
}
```

(18) Reverse pairs:

[40, 25, 19, 12, 9, 6, 2]

i<j && a[i] > 2* a[j]

Brute force:

```
for(i→0→n)
{for(j=i+1→n+1)
  if(a[i]>2*a[j])
}}      cnt++;
```

Better approach: