Search A 2-D Matrix

Step 1:Start from the top right corner of the matrix(matrix[0][n-1])

Step 2:If element is less than the target move down

else if the element is greater than the target move left.

If you have trouble understanding watch this:https://youtu.be/ZYpYur0znng

C++

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m,n,i,j;
        m=matrix.size(),n=matrix[0].size();
        i=0,j=n-1;
        while(i>=0 and j>=0 and i<m and j<n)
        {
            if(matrix[i][j]==target)return true;
            //if target is less than current element
            if(target<matrix[i][j])
                j--;
            else
                i++;
        }
        return false;
    }
};
```

Java

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int m,n,i,j;
        m=matrix.length;n=matrix[0].length;
        i=0;j=n-1;
        while(i>=0 && j>=0 && i<m && j<n)
        {
            if(matrix[i][j]==target)return true;
            //if target is less than current element
            if(target<matrix[i][j])
                j--;
            else
                i++;
        }
        return false;
    }
}
```

Python

```
class Solution(object):
    def searchMatrix(self, matrix, target):
        m,n=len(matrix),len(matrix[0]);
        i,j=0,n-1

        while(i>=0 and j>=0 and i<m and j<n):
            if matrix[i][j]==target:
                return True
            #If element is greater than target
            elif matrix[i][j]>target:
                j=j-1
            else:
                i=i+1

        return False
```

Magnetic Force Between Two Balls

Here we are asked to find the minimum magnetic force so that each ball can be placed in the basket.
How magnetic force is calculated?
The difference between the two positions is the magnetic force between them.
So here minimum magnetic force can be 0 and maximum magnetic force can be max(position)
So our magnetic force lies somewhere between 0 and max(position)
We apply binary search to find minimum magnetic force which can be applied and would be maximum among all the cases.
Since the array is not sorted so we will sort the array so that position remains in sorted form.

Read more detail solution

C++

```
 class Solution {
public:
    bool get(vector<int>&position,int mid,int m)
    {
        int i,cnt=1,prev=position[0];
        //here prev denote the last position where ball was placed
       //because the magentic force is distance between two distances
        for(i=1;i<position.size();i++)
        {
            if((position[i]-prev)>=mid)
            {
                cnt++;
                prev=position[i];
                if(cnt==m)return true;
            }
        }
        if(cnt<m)return false;
        return true;
    }
    int maxDistance(vector<int>& position, int m) {
        int lo,hi;
     sort(position.begin(),position.end());
        lo=0,hi=position[position.size()-1];
            int ans=0;
        while(lo<=hi)
        {
            int mid=lo+(hi-lo)/2;
       //checking that the current magnetic force is possible
            //if yes then we should increase the magnetic force to
            //maximise it
            if(get(position,mid,m))
            {ans=mid;lo=mid+1;}
      //magnetic force is very large we can not place all the balls
            //we are reducing some force now so that all balls can be placed
            else
                hi=mid-1;
        }
        return ans;
    }
};
```

Java

```
class Solution {
    boolean get(int[] position,int mid,int m)
    {
        int i,prev,cnt=1;
        prev=position[0];
        //here prev denote the last position where ball was placed
        //because the magentic force is distance between two distances
        for(i=1;i<position.length;i++)
        {
            if(position[i]-prev>=mid)
            {
                cnt++;prev=position[i];
                if(cnt==m)return true;
            }
        }
        if(cnt<m)return false;
        return true;
    }
      public int maxDistance(int[] position, int m) {

        Arrays.sort(position);
        int lo=0,hi=position[position.length-1];
        int ans=0;
        while(lo<=hi)
        {
            int mid=lo+(hi-lo)/2;
            boolean isForcePossible = get(position,mid,m);
            //checking that the current magnetic force is possible
            //if yes then we should increase the magnetic force to
            //maximise it
            if(isForcePossible)
            {
                ans=mid;
                lo=mid+1;
            }
            //magnetic force is very large we can not place all the balls
            //we are reducing some force now so that all balls can be placed
            else
                hi=mid-1;
        }
          return ans;
    }
}
```

Python

```
class Solution:
    def maxDistance(self, position: List[int], m: int) -> int:
        n = len(position)
        position.sort()

        def count(d):
            ans, curr = 1, position[0]
            for i in range(1, n):
                if position[i] - curr >= d:
                    ans += 1
                    curr = position[i]
                    if(ans==m):
                        return ans
            return ans

        l, r = 0, max(position)
        ans=0
        while l <= r:
            mid = l+(r - l) // 2
            if count(mid) == m:
                ans=mid
                l = mid+1
            elif count(mid)>m:
                lo=mid+1
            else:
                r = mid - 1
        return ans
```

Capacity To Ship Packages Within D Days

The minimum capacity we can load on the ship would be max(capacity) and the maximum capacity we can load on the ship is the sum of total weight.
Now the constraint here is we have to load all the weight in D days so we would apply binary search to find them, minimum weight to ship within D days.

Read more about this question here

C++

```
class Solution {
public:
       //checking the no of days in which shipping is possible
      int get(vector<int>&weights,int mid)
      {
            int i,j;
            int d=1,sum=0;
            for(i=0;i<weights.size();i++)
            {
                  sum+=weights[i];
                  if(sum>mid)
                  {
                        d++;
                        sum=weights[i];
                  }
            }
            return d;
      }
    int shipWithinDays(vector<int>& weights, int D) {
            int lo,hi,n;
            n=weights.size();
            int i,j;
                  //lo is minimum maximum capacity you can ship and high is maximum
                   //capacity you can ship
            lo=INT_MIN,hi=0;
            for(i=0;i<n;i++)
            {
                  hi+=weights[i];
                  lo=max(lo,weights[i]);
            }
            while(lo<=hi)
            {
                  int mid=lo+(hi-lo)/2;
                  int days=get(weights,mid);
                  if(days<=D)
                  hi=mid-1;
                  else
                        lo=mid+1;
            }
            return lo;
        }
};
```

Java

```
class Solution {
    //checking the no of days in which shipping is possible
      public int get(int[]weights,int mid)
      {
            int i,j;
            int d=1,sum=0;
            for(i=0;i<weights.length;i++)
            {
                  sum+=weights[i];
                  if(sum>mid)
                  {
                        d++;
                        sum=weights[i];
                  }
            }
            return d;
      }
   public int shipWithinDays(int[] weights, int D) {
            int lo,hi,n;
            n=weights.length;
             int i,j;
            //lo is minimum maximum capacity you can ship and high is maximum
            //capacity you can ship
            lo=Integer.MIN_VALUE;hi=0;
            for(i=0;i<n;i++)
            {
                  hi+=weights[i];
                  lo=Math.max(lo,weights[i]);
            }
            while(lo<=hi)
            {
                  int mid=lo+(hi-lo)/2;
                  int days=get(weights,mid);
                  if(days<=D)
                  hi=mid-1;
                  else
                        lo=mid+1;
            }
            return lo;
        }
}
```

Python

```
class Solution:
   def shipWithinDays(self, weights: List[int], D: int) -> int:
        #Checking for if is possible to ship in D days
        def cannot_split(weights, D, max_wgt):
            s = 0
            days = 1
            for w in weights:
                s += w
                if s > max_wgt:
                    s = w
                    days += 1
            return days > D

        low, high = max(weights), sum(weights)
        while low <= high:
            mid = low + (high - low) // 2
            if cannot_split(weights, D, mid):
                low = mid + 1
            else:
                high = mid-1
        return low
```

[Smallest Good Base]

C++

```
class Solution {
public:
    string smallestGoodBase(string n) {
        long long  num = stoll(n);

        for(int m = 61; m > 2; m--){
            long long lo = 2, hi = num-1;


            while(lo <= hi){
                long long  mid =  (lo + hi) / 2;
                int flag = 0;
                long long val = 1, j = 1, res = 1;
                while(j < m){
                    if(val  > (num - res) / mid){
                        flag = 1;
                        break;
                    }
                    j++;
                    val *= mid;
                    res += val;
                }

                if(flag || res > num){
                    hi = mid-1;
                }
                else if(res < num)
                    lo = mid + 1;
                else
                    return to_string(mid);
            }
        }
        return to_string(num-1);
    }
};
```

Java

```
class Solution {
    public String smallestGoodBase(String n) {
        long num = Long.valueOf(n);

        for(int m = 61; m > 2; m--){
            long lo = 2, hi = num-1;


            while(lo <= hi){
                 long  mid =  (lo + hi) / 2;
                int flag = 0;
                long val = 1, j = 1, res = 1;
                while(j < m){
                    if(val  > (num - res) / mid){
                        flag = 1;
                        break;
                    }
                    j++;
                    val *= mid;
                    res += val;
                }

                if(flag==1 || res > num){
                    hi = mid-1;
                }
                else if(res < num)
                    lo = mid + 1;
                else
                     return String.valueOf(mid);
            }
        }
          return String.valueOf(num - 1);
    }

}
```

Python

```
import math
class Solution(object):
    def smallestGoodBase(self, n):

        for i in range(64, 1, -1):
            # i is the number of bits
            low, high = 2, 10000000000000000000
            while(low <= high):
                mid = low + (high - low)//2
                value = 0
                for j in range(0, i):
                    value = value + mid**j
                if (value == int(n)):
                    return str(mid)
                elif (value > int(n)) :
                    high = mid - 1
                else:
                    low = mid + 1

        return -1
```