2D-ARRAY

Q1)Print in using vectors

```
#include <bits/stdc++.h>
using namespace std;
void print(vector<vector<int>> arr){
    for(int i = 0; i < arr.size(); i++){
        for(int   j = 0; j < arr[i].size(); j++)
            cout << arr[i][j] << " ";
    }
}

int main(){
    int m = 3, n = 2;
    vector<vector<int>> arr;
    for(int i = 0; i < m; i++){
        vector<int> v;
        for(int j = 0; j < n; j++){
            v.push_back(j);
        }
        arr.push_back(v);
    }
    print(arr);
    return 0;
}
```

Q2)snake pattern

```
void printSnake(int mat[R][C]){
    for(int i = 0; i < R; i++){
        if(i % 2 == 0){
            for(int j = 0; j < C; j++)
                cout << mat[i][j] << " ";
        }
        else{
            for(int j = C - 1; j >= 0; j--)
                cout << mat[i][j] << " ";
        }

    }
}
```

Q3)Boundary traversal

```
const int R = 4, C = 4;
void bTraversal(int mat[R][C])
{
    if(R == 1){//corner case
        for(int i = 0; i < C; i++)
            cout << mat[0][i] << " ";
    }
    else if(C == 1){//corner case
        for(int i = 0; i < R; i++)
            cout << mat[i][0] << " ";
    }
    else{
        for(int i = 0; i < C; i++)
            cout << mat[0][i] << " ";//first row traversl
        for(int i = 1; i < R; i++)
            cout << mat[i][C - 1] << " ";//last column traversal
        for(int i = C - 2; i >= 0; i--)
```

```
                                    cout << mat[R - 1][i]
<< " ";
                        for(int i = R - 2; i >= 1; i--)
                                    cout << mat[i][0] << "
";
            }

}
```

Q4) transpose

```
int rows = arr.size();
   int cols = arr[0].size();
   vector<vector<int>> transposed(cols,
vector<int>(rows));

   for(int i = 0; i < rows; i++) {
      for(int j = 0; j < cols; j++) {
         transposed[j][i] = arr[i][j];
      }
   }

   for(int i = 0; i < transposed.size(); i++) {
      for(int j = 0; j < transposed[i].size(); j++) {
         cout << transposed[i][j] << " ";
      }
      cout << endl;
   }
}
```

Q5) rotate by 90(right)

```
class Solution {
public:
   void rotate(vector<vector<int>>& matrix) {
      int n = matrix.size();
      // Step 1: Transpose the matrix
      for (int i = 0; i < n; i++) {
         for (int j = i + 1; j < n; j++) {
            swap(matrix[i][j], matrix[j][i]);
         }
      }

      // Step 2: Reverse each row
      for (int i = 0; i < n; i++) {
         reverse(matrix[i].begin(),
matrix[i].end());
      }
   }
};
```

Q6)Rotate left by 90(left)
same transpose but reverse the whole matrix upside down

Q7)SPIRAL TRAVERSAL

```
class Solution {
public:
   vector<int>
spiralOrder(vector<vector<int>>& matrix) {
      vector<int> result;
      if (matrix.empty() || matrix[0].empty()) {
         return result;
      }

      int rows = matrix.size(), cols =
matrix[0].size();
      int left = 0, right = cols-1, top = 0, bottom
= rows-1;
```

```
        while (left <= right && top <= bottom) {
            for (int i = left; i <= right; i++) {
                result.push_back(matrix[top][i]);
            }
            top++;

            for (int i = top; i <= bottom; i++) {
                result.push_back(matrix[i][right]);
            }
            right--;

            if (top <= bottom) {
                for (int i = right; i >= left; i--) {
result.push_back(matrix[bottom][i]);
                }
                bottom--;
            }

            if (left <= right) {
                for (int i = bottom; i >= top; i--) {
                    result.push_back(matrix[i][left]);
                }
                left++;
            }
        }

        return result;
    }
};
```

BINARY SEARCH

Q8)

https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/

```
class Solution {
public:
  int findPivot(vector<int>& nums) {
    int mid;
    int l = 0, r = nums.size()-1;
    while(l < r) {
      mid = l + (r-l)/2;


      if(nums[mid] > nums[r]) { //something wrong on right side. Possibly my answer lies in right side.
        l = mid+1;          //move right (Discarding mid, because it's greater than nums[r], so it can't be the minimum element)
      } else {
        r = mid;
      }
    }
    return nums[l];
  }

  int findMin(vector<int>& nums) {
    return findPivot(nums);
  }
};
```

Q2) https://leetcode.com/problems/search-a-2d-matrix/

```
class Solution {
public:
  bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int m = matrix.size();
    int n = matrix[0].size();

    int start = 0;
    int end   = m*n-1;

    while(start <= end) {
      int mid = start + (end-start)/2;

      int row = mid/n;
      int col = mid%n;

      if(matrix[row][col] > target) {
        end = mid-1;
      } else if(matrix[row][col] < target) {
        start = mid+1;
      } else {
        return true;
      }
    }

    return false;

  }
};
```

Q3)

https://leetcode.com/problems/single-element-in-a-sorted-array/description/

```
class Solution {
public:
  int singleNonDuplicate(vector<int>& nums) {
    int n = nums.size();
    int l = 0, r = n-1;
    int mid;

    while(l < r) {
      mid = l + (r-l)/2;
      bool isEven = (r-mid)%2==0;

      if(nums[mid] == nums[mid+1]) {
        if(isEven) {
          l = mid+2;//go right
        } else {
          r = mid-1;//go left
        }
      } else if(nums[mid] == nums[mid-1]) {
        if(isEven) {
          r = mid-2;
        } else {
          l = mid+1;
        }
      } else {
        return nums[mid];
      }
    }

    return nums[l]; //or, nums[r]
  }
};
```

Q4) https://leetcode.com/problems/single-element-in-a-sorted-array/description/

```
class Solution{
public:
       int findKRotation(int arr[], int n) {
         int l = 0, r = n-1;
         while(l < r) {
           int mid = l + (r-l)/2;
           if(arr[mid] < arr[r]) {
             r = mid;
           } else {
             l = mid+1;
           }
         }
         return r;
       }

};
```

Q5)

https://leetcode.com/problems/search-in-rotated-sorted-array/description/

```
class Solution {
public:
  int find_pivot(vector<int>& nums, int l, int r) {
    while(l < r) {
      int mid = l + (r-l)/2;

      if(nums[mid] > nums[r]) {
        l = mid+1;
      } else {
        r = mid;
      }
    }
    return r;
  }

  int binary_search(vector<int>& nums, int l, int r, int target) {
    while(l<=r) {
      int mid = l + (r-l)/2;
      if(nums[mid] == target)
        return mid;
      if(nums[mid] < target)
        l = mid+1;
      else
        r = mid-1;
    }

    return -1;
  }

  int search(vector<int>& nums, int target) {
    int n = nums.size();
    int pivot = find_pivot(nums, 0, n-1);

    if(nums[pivot] == target)
      return pivot;

    int idx = -1;
    idx = binary_search(nums, pivot+1, n-1, target);
    if(idx != -1)
      return idx;
    idx = binary_search(nums, 0, pivot-1, target);
    return idx;
  }
};
```

Q6)

https://leetcode.com/problems/search-in-rotated-sorted-array-ii/description/

```
class Solution {
public:
  int pivot(vector<int>& nums, int l, int r) {
    while(l < r) {

      while(l < r && nums[l] == nums[l+1])
        l++;

      while(r < l && nums[r] == nums[r-1])
        r--;

      /*
        You need to do what I did above because you'll fail in case like
        [1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1]
        2
        Here, the nums[mid] <= nums[r] and
        and we will cut down the right half but our pivot lies there
        So, make it a RULE, whenever there are duplicate elements and you need to to something
        like Binary Search, you need to ignore duplicates like done above
        Similar Qn : "Smallest element in a rotated sorted array with duplicates"
```

```
            */
            int mid = l + (r-l)/2;

            if(nums[mid] <= nums[r]) { //sorted part
                r = mid; //possibly my pivot
            } else {
                l = mid+1;
            }
        }

        return r;
    }

    bool binarySearch(vector<int>& nums, int l, int r, int& target) {
        while(l <= r) {
            int mid = l + (r-l)/2;

            if(nums[mid] == target)
                return true;

            if(nums[mid] < target)
                l = mid+1;
            else
                r = mid-1;
        }

        return false;
    }

    bool search(vector<int>& nums, int target) {
        int n = nums.size();

        int p = pivot(nums, 0 , n-1);
        cout <<"p = " << p << endl;
        if(binarySearch(nums, 0, p-1, target)) {
            return true;
        }

        return binarySearch(nums, p, n-1, target);

    }
};
```

Q7)

https://leetcode.com/problems/kth-missing-positive-number/

```
class Solution {
public:
    int findKthPositive(vector<int>& arr, int k) {
        int n = arr.size();

        int l = 0, r = n-1;

        while(l <= r) {
            int mid = l + (r-l)/2;

            if(arr[mid] - (mid+1) < k) { //A[mid]-(mid+1)   --> This gives umber of missing number before m'th index
                l = mid+1;
            } else {
                r = mid-1;
            }

        }



        return l + k; //see my youtube video above for the explanation of this line

    }

};
```

Q8)

https://leetcode.com/problems/minimum-time-to-complete-trips/description/

```
class Solution {

public:

    bool possible(vector<int>& time, long long givenTime, int totalTrips) {

        long long actualTrips = 0;


        for(int &t : time){

            actualTrips += givenTime/t;

        }


        return actualTrips >= totalTrips;

    }


    long long minimumTime(vector<int>& time, int totalTrips) {

        int n = time.size();


        long long l = 1;

        long long r = (long long) *min_element(begin(time), end(time)) * totalTrips;

        while(l < r) {


            long long mid_time = l + (r - l)/2;


            if(possible(time, mid_time, totalTrips)) {

                r = mid_time;

            } else {

                l = mid_time + 1;

            }


        }


        return l;//note what to return


    }

};
```

Q9)

https://leetcode.com/problems/koko-eating-bananas/description/

```
class Solution {

public:


    bool canEatAll(vector<int>& piles, int givenHour, int h) {

        int actualHour = 0;


        for(int &x : piles) {

            actualHour += x/givenHour;


            if(x%givenHour != 0)
```

```
                actualHour++;

            }

            return actualHour <= h;
        }

        int minEatingSpeed(vector<int>& piles, int h) {
            int n = piles.size();

            int l = 1, r = *max_element(begin(piles), end(piles));

            while(l < r) {
                int mid = l + (r-l)/2;

                if(canEatAll(piles, mid, h)) {
                    r = mid;
                } else {
                    l = mid+1;
                }

            }

            return l;
        }
};
```

Q10)

https://leetcode.com/problems/successful-pairs-of-spells-and-potions/description/

```
//interviwer method
class Solution {
public:
    int BinarySearch(int l, int r, vector<int>& potions, int target){
        //we have to find the index of first element greater than or equal to target (minPotion)
        int index = -1;
        int mid = 0;
        while(l <= r){
            mid = l + (r-l)/2;
            if(potions[mid] >= target) {
                index = mid;
                r = mid-1;
            } else {
                l = mid+1;
            }
        }
        return index;
    }

    vector<int> successfulPairs(vector<int>& spells, vector<int>& potions, long long success) {
        int m = spells.size();
        int n = potions.size();
        sort(begin(potions), end(potions));
        int maxPotion = potions[n-1];
        vector<int> answer;
        for(int i = 0; i<m; i++){
            int spell = spells[i];
            long long minPotion = ceil((1.0*success)/spell);
```

```
      if(minPotion > maxPotion) {
        answer.push_back(0);
        continue;
      }
      int index = BinarySearch(0, n-1, potions, minPotion);//lower_bound(begin(potions), end(potions), minPotion) - begin(potions);
      answer.push_back(n-index);
    }
    return answer;
  }
};

//stl method
class Solution {
public:
  vector<int> successfulPairs(vector<int>& spells, vector<int>& potions, long long success) {
    int m = spells.size();
    int n = potions.size();
    sort(begin(potions), end(potions));
    int maxPotion = potions[n-1];

    vector<int> answer;
    for(int i = 0; i<m; i++) {

      int spell = spells[i];

      long long minPotion = ceil((1.0*success)/spell);

      if(minPotion > maxPotion) {
        answer.push_back(0);
        continue;
      }

      int index = lower_bound(begin(potions), end(potions), minPotion) - begin(potions);

      answer.push_back(n-index);

    }

    return answer;
  }
};
```

Q11)

https://leetcode.com/problems/minimize-maximum-of-array/description/

```
class Solution {
public:

  bool isValid(vector<int> &nums, int mid_max, int n) {

    vector<long long> arr(begin(nums), end(nums));//copy of the nums
    for(int i = 0; i<n-1; i++) {
      if(arr[i] > mid_max)
        return false;
      long long buffer = mid_max - arr[i];
      arr[i+1] = arr[i+1] - buffer;
    }
```

```
        return arr[n-1] <= mid_max;

    }

    int minimizeArrayValue(vector<int>& nums) {
        int n = nums.size();
        int maxL = 0;
        int maxR = *max_element(begin(nums), end(nums));
        int result;
        while(maxL <= maxR) {

            int mid_max = maxL + (maxR-maxL)/2;

            if(isValid(nums, mid_max, n)) {
                result = mid_max;
                maxR   = mid_max-1;
            } else {
                maxL = mid_max+1;
            }

        }

        return result;
    }
};
```

Q12)

https://leetcode.com/problems/minimize-maximum-of-array/description/

```
class Solution {
public:
    typedef long long ll;

    ll getSumElements(ll count, ll val) {

        return val*count - (count*(count+1))/2;

    }

    int maxValue(int n, int index, int maxSum) {

        ll left  = 1;
        ll right = maxSum;

        ll mid_val;
        int result = 0;


        while(left <= right) {

            mid_val = left + (right - left)/2;

            ll left_count = min((ll)index, mid_val-1);//formula we need this much

            ll left_sum  = getSumElements(left_count, mid_val);

            left_sum += max((ll)0, index - mid_val+1);//number of 1 we need to add
            ll right_count = min((ll)n-index-1, mid_val-1);


            ll right_sum  = getSumElements(right_count, mid_val);


            right_sum += max((ll)0, n-index-1 - mid_val+1);




            if(left_sum + right_sum + mid_val <= maxSum) {
                result = max((ll)result, mid_val);

                left = mid_val+1;
            } else {
                right = mid_val-1;
            }

        }


        return result;

    }
};
```

Q13)

https://leetcode.com/problems/most-profit-assigning-work/description/

```
class Solution {
public:
    int maxProfitAssignment(vector<int>& difficulty, vector<int>& profit, vector<int>& worker) {
        int n = difficulty.size();
        int m = worker.size();


        vector<pair<int, int>> vec;
        for(int i = 0; i < n; i++) {
            vec.push_back({difficulty[i], profit[i]});
        }


        sort(begin(vec), end(vec));


        //Pre-processing to find the maximum profit till index i at constant time
        for(int i = 1; i < vec.size(); i++) {
            vec[i].second = max(vec[i].second, vec[i-1].second);
        }



        int totalProfit = 0;
        for(int i = 0; i < m; i++) {

            int workerDiffLevel = worker[i];


            //apply b.search on vec
            int l = 0, r = vec.size()-1;
            int maxProfit = 0;
            while(l <= r) {
                int mid = l + (r-l)/2;
                if(vec[mid].first <= workerDiffLevel) {
```

```
          maxProfit = max(maxProfit, vec[mid].second);
          l = mid+1;
        } else {
          r = mid-1;
        }
      }

      totalProfit += maxProfit;

    }

    return totalProfit;
  }
};
```

Q14)

https://leetcode.com/problems/minimum-cost-to-make-array-equal/description/

```
class Solution {
public:
  typedef long long ll;

  ll getCost(vector<int>& nums, vector<int>& cost, int target) {
    ll result = 0;
    for (int i = 0; i < nums.size(); ++i) {
      result += (ll) abs(nums[i] - target) * cost[i];
    }
    return result;
  }

  long long minCost(vector<int>& nums, vector<int>& cost) {

    ll answer = INT_MAX;

    int left = *min_element(nums.begin(), nums.end());  //1
    int right = *max_element(nums.begin(), nums.end()); //5

    while (left <= right) {
      int mid = left + (right-left)/2;

      ll cost1 = getCost(nums, cost, mid);//check on left
      ll cost2 = getCost(nums, cost, mid + 1);//check on right

      answer = min(cost1, cost2);
      if (cost1 > cost2)
        left = mid + 1;
      else
        right = mid-1;
    }
    return answer == INT_MAX ? 0 : answer;
  }
};
```

Q15)

https://leetcode.com/problems/peak-index-in-a-mountain-array/description/

```
class Solution {
public:
    int
peakIndexInMountainArray(vector<int>& arr)
{
        int n = arr.size();

        int l = 0;
        int r = n-1;

        while(l < r) {

            int mid = l + (r-l)/2;

            if(arr[mid] < arr[mid+1])
                l = mid+1;
            else
                r = mid;

        }

        return l;
    }
};
```

Q16)

https://leetcode.com/problems/minimum-speed-to-arrive-on-time/description/

```
class Solution {
public:

    double possible(vector<int>& dist, int
speed) {

        double time = 0.0;
        int n = dist.size();

        for(int i = 0; i < n - 1; i++) {

            double t =
(double)dist[i]/(double)speed;

            time += ceil(t);

        }

        time += (double)dist[n-1]/(double)speed;

        return time;

    }

    int minSpeedOnTime(vector<int>& dist,
double hour) {
        int l = 1;
        int r = 1e7;

        int minSpeed = -1;

        while(l <= r) {

            int mid = l + (r-l)/2;

            if(possible(dist, mid) <= hour) {
                minSpeed = mid;
                r = mid-1;
            } else {
                l = mid+1; //need to speed up
            }

        }

        return minSpeed;
    }
};
```

Q17)

https://leetcode.com/problems/maximum-running-time-of-n-computers/description/

```
class Solution {
public:
    typedef long long ll;

    bool possible(vector<int>& batteries, ll mid_time, int n) {

        ll target = n*mid_time; //each computer will run mid_time minutes

        ll sum = 0;

        for(int i = 0; i<batteries.size(); i++) {

            target -= min((ll)batteries[i], mid_time);

            if(target <= 0)
                return true;

        }
        return target <= 0;
    }

    long long maxRunTime(int n, vector<int>& batteries) {

        ll l = *min_element(begin(batteries), end(batteries));

        long long sum_total_minutes = 0;

        for(auto &mints : batteries){
            sum_total_minutes += mints;
        }

        ll r = sum_total_minutes/n;

        ll result = 0;

        while(l <= r) {

            ll mid_time = l + (r-l)/2;

            if(possible(batteries, mid_time, n)) {
                result = mid_time;
                l = mid_time+1;
            } else {
                r = mid_time-1;
            }
        }

        return result;
    }
};
```

Q18)

https://leetcode.com/problems/minimize-the-maximum-difference-of-pairs/description/

```
class Solution {
public:
  int n;

  bool isValid(vector<int>& nums, int mid, int p) {

    int i = 0;
    int pairs = 0;
    while(i<n-1){
      if(nums[i+1] - nums[i] <= mid) {
        pairs++;
        i+=2;
      } else {
        i++;
      }
    }

    return pairs >= p;

  }

  int minimizeMax(vector<int>& nums, int p) {
    n = nums.size();

    sort(begin(nums), end(nums));

    int l = 0;
    int r = nums[n-1] - nums[0];

    int result = INT_MAX;

    while(l <= r) {

      int mid = l + (r-l)/2;

      if(isValid(nums, mid, p)) {
        result = mid;
        r = mid-1;
      } else {
        l = mid+1;
      }

    }

    return result;

  }
};
```

Q19)

https://leetcode.com/problems/median-of-two-sorted-arrays/

```
class Solution {
public:
  double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
    if(nums1.size() > nums2.size())
```

```
            return findMedianSortedArrays(nums2, nums1);

        int m = nums1.size();
        int n = nums2.size();

        int low = 0, high = m;
        while(low <= high) {

            int Px = low + (high-low)/2;
            int Py = (m+n+1)/2 - Px;

            int x1  = (Px == 0) ? INT_MIN : nums1[Px-1];
            int x3 = (Px == m) ? INT_MAX : nums1[Px];

            int x2  = (Py == 0) ? INT_MIN : nums2[Py-1];
            int x4 = (Py == n) ? INT_MAX : nums2[Py];

            if(x1 <= x4 && x2 <= x3) {
                if((m+n)%2 == 0)
                    return (max(x1, x2) + min(x3, x4))/2.0;

                return max(x1, x2);
            } else if(x1 > x4) {
                high = Px-1;
            } else {
                low = Px+1;
            }
        }

        return -1;
    }
};
```