Trapping Rain Water

In this problem, we just have to figure out the amount of water stored for all the indexes from 0 to n. For any index i, the amount of water stored is the minimum of maximal height on the left side and the maximal height on the right side minus the current height of the building. Please refer to this video for more information.

C++

```
class Solution {
public:
    int trap(vector<int>& height) {
        if (height.size() == 0) {
            return 0;
        }

        int maxleft[height.size()];
        int maxright[height.size()];

        maxleft[0] = 0;
        for (int i =1 ; i < height.size(); i++) {
            maxleft[i] = max(maxleft[i-1], height[i-1]);
        }

        maxright[height.size() - 1] = 0;
        for (int i = height.size() - 2; i>=0; i--) {
            maxright[i] = max(maxright[i+1], height[i+1]);
        }

        int res = 0;
        for (int i = 0; i < height.size(); i++) {
            int water_above = min(maxleft[i], maxright[i]) - height[i];
            if (water_above > 0) {
                res = res + water_above;
            }
        }
        return res;
    }
};
```

Java

```
class Solution {
    public int trap(int[] height) {

        int result =0;

        if(height.length>0){

            int n = height.length;

            int lmax[] = new int[n];
            int rmax[] = new int[n];

            lmax[0] = 0;
            for(int i=1;i<n;i++){
                lmax[i] = Math.max(height[i-1],lmax[i-1]);
            }

            rmax[n-1] = 0;
            for(int i=n-2;i>=0;i--){
                rmax[i] = Math.max(height[i+1],rmax[i+1]);
            }

            for(int i=0;i<n;i++){
                int water = (Math.min(lmax[i],rmax[i]) - height[i]);
                if (water > 0) {
                    result = result + (Math.min(lmax[i],rmax[i]) - height[i]);
                }
            }
        }
        return result;

    }
}
```

Python

```
class Solution:
    def trap(self, height: List[int]) -> int:
        maxleft = [0 for i in range(len(height))]
        maxright = [0 for i in range(len(height))]

        for i in range(1, len(height)):
            maxleft[i] = max(maxleft[i-1], height[i-1])

        for i in range(len(height) - 2, -1, -1):
            maxright[i] = max(maxright[i+1], height[i+1])

        res = 0
        for i in range(len(height)):
            water_above = min(maxleft[i], maxright[i])
            if water_above > height[i]:
                res += water_above - height[i]

        return res
```

C#

```
public class Solution {
    public int Trap(int[] height) {
        int n=height.Length;
        if(height == null||n==0)
        return 0;
        int[] left=new int[n];
        int[] right=new int[n];
        int maxwater=0;
        left[0]=height[0];
        for(int i=1;i<n;i++){
            if(height[i]>left[i-1])
            {
                left[i]=height[i];
            }
            else
            {
                left[i]=left[i-1];
            }
        }

        right[n-1]=height[n-1];
        for(int i=n-2;i>=0;i--){
            if(height[i]>right[i+1])
            {
                right[i]=height[i];
            }
            else
            {
                right[i]=right[i+1];
            }
        }

         for(int i=0;i<n;i++){
             maxwater+=Math.Min(left[i],right[i])-height[i];
         }
        return maxwater;
    }
}
```

Javascript

```
/**
 * @param {number[]} height
 * @return {number}
 */
var trap = function(height) {
  if (height.length < 3) return 0;
  const maxHeightLeft = height.slice();
  const maxHeightRight = height.slice();
  for (let i = 1; i < height.length; i++) {
    if (maxHeightLeft[i] < maxHeightLeft[i - 1])
      maxHeightLeft[i] = maxHeightLeft[i - 1];
  }
  for (let i = height.length - 2; i >= 0; i--) {
    if (maxHeightRight[i] < maxHeightRight[i + 1])
      maxHeightRight[i] = maxHeightRight[i + 1];
  }
  return height.reduce(
    (acc, cur, idx) =>
      acc + Math.min(maxHeightLeft[idx], maxHeightRight[idx]) - cur,
    0
  );
};
```

Maximum of absolute value expression

Here, it should be noted that we don’t have to worry about the absolute value of i - j, as if we swap i and j, we would get the same sum. So, we can assume that i >= j. We can make different cases such as when arr1[i] > arr1[j], arr2[i] > arr2[j] and so on. Please take a look at this video for more information.

C++

```
class Solution {
public:

    int maxAbsValExpr(vector<int>& arr1, vector<int>& arr2) {
        vector<int> val1, val2, val3, val4;
        for (int i = 0; i < arr1.size(); i++){
            val1.push_back(i + arr1[i] + arr2[i]);
            val2.push_back(i + arr1[i] - arr2[i]);
            val3.push_back(i - arr1[i] + arr2[i]);
            val4.push_back(i - arr1[i] - arr2[i]);
        }
        int ans = 0;
        ans = max(ans, maxArray(val1) - minArray(val1));
        ans = max(ans, maxArray(val2) - minArray(val2));
        ans = max(ans, maxArray(val3) - minArray(val3));
        ans = max(ans, maxArray(val4) - minArray(val4));
        return ans;
    }
    int maxArray(vector<int>& arr) {
        int max = -99999;
        for (int i = 0; i < arr.size(); i++) {
            if (arr[i] > max) {
                max = arr[i];
            }
        }
        return max;
    }
    int minArray(vector<int>& arr) {
        int min = 99999;
        for (int i = 0; i < arr.size(); i++) {
            if (arr[i] < min) {
                min = arr[i];
            }
        }
        return min;
    }
};
```

Java

```
class Solution {
    public int maxAbsValExpr(int[] arr1, int[] arr2) {
        int[] val1 = new int[arr1.length];
        int[] val2 = new int[arr1.length];
        int[] val3 = new int[arr1.length];
        int[] val4 = new int[arr1.length];

        for (int i = 0; i < arr1.length; i++) {
            val1[i] = i + arr1[i] + arr2[i];
            val2[i] = i - arr1[i] + arr2[i];
            val3[i] = i + arr1[i] - arr2[i];
            val4[i] = i - arr1[i] - arr2[i];
        }

        int res = 0;
        res = Math.max(res, maxArray(val1) - minArray(val1));
        res = Math.max(res, maxArray(val2) - minArray(val2));
        res = Math.max(res, maxArray(val3) - minArray(val3));
        res = Math.max(res, maxArray(val4) - minArray(val4));
        return res;

    }
    public int maxArray(int[] arr) {
        int max = -99999;
        for (int i = 0; i < arr.length; i ++) {
            max = Math.max(max, arr[i]);
        }
        return max;
    }
    public int minArray(int[] arr) {
        int min = 99999;
        for (int i = 0; i < arr.length; i ++) {
            min = Math.min(min, arr[i]);
        }
        return min;
    }
}
```

Python

```
class Solution:
    def maxAbsValExpr(self, arr1: List[int], arr2: List[int]) -> int:
        val1, val2, val3, val4 = [], [], [], []
        for i in range(len(arr1)):
            val1.append(i+arr1[i]+arr2[i])
            val2.append(i+arr1[i]-arr2[i])
            val3.append(i-arr1[i]+arr2[i])
            val4.append(i-arr1[i]-arr2[i])

        res = 0
        res = max(res, max(val1) - min(val1))
        res = max(res, max(val2) - min(val2))
        res = max(res, max(val3) - min(val3))
        res = max(res, max(val4) - min(val4))
        return res
```

C#

```
public class Solution
{
    public int MaxAbsValExpr(int[] arr1, int[] arr2)
    {
        int ans=0;
        var sign = new int[4, 3] {{1,1,1}, {1,1,-1}, {1,-1,1}, {1,-1,-1}};
        for(int s = 0; s < sign.GetLength(0); s++)
        {
            int mini=int.MaxValue, maxi= int.MinValue;
            for(int i = 0; i < arr1.Length; i++)
            {
                mini=Math.Min(mini, sign[s, 0] * arr1[i] + sign[s, 1] * arr2[i] +
sign[s, 2] * i);
                maxi=Math.Max(maxi, sign[s, 0] * arr1[i] + sign[s, 1] * arr2[i] +
sign[s, 2] * i);
            }

            ans = Math.Max(ans, maxi-mini);
        }

        return ans;
    }
}
```

Javascript

```
/**
 * @param {number[]} arr1
 * @param {number[]} arr2
 * @return {number}
 */
var maxAbsValExpr = function(arr1, arr2) {
    const n = arr1.length;
    const bucket1 = new Array(n);
    const bucket2 = new Array(n);
    const bucket3 = new Array(n);
    const bucket4 = new Array(n);

    for(let i = 0; i < n; i++)
    {
        bucket1[i] = arr1[i] + arr2[i] + i;
        bucket2[i] = arr1[i] + arr2[i] - i;
        bucket3[i] = arr1[i] - arr2[i] + i;
        bucket4[i] = arr1[i] - arr2[i] - i;
    }

    return Math.max(
        getMinMaxDiff(bucket1),
        getMinMaxDiff(bucket2),
        getMinMaxDiff(bucket3),
        getMinMaxDiff(bucket4),
    );
};

function getMinMaxDiff(arr) {
    let max = Number.NEGATIVE_INFINITY;
    let min = Number.POSITIVE_INFINITY;

    for (let i = 0; i < arr.length; i++) {
        max = Math.max(max, arr[i]);
        min = Math.min(min, arr[i]);
    }

    return max - min;
}
```

Spiral Matrix II

Here the approach is to use four variables top = 0, bottom = n - 1, left = 0, and right = n - 1. We have to fill all the columns of the top row from left to right and then increment the top. Then we have to fill the rightmost column from top to bottom. Then the last row is filled from right to left and then the first column is printed. Please refer to this video.

C++

```
class Solution {
public:

    vector<vector<int>> generateMatrix(int n) {

        vector<vector<int>> result (n, vector<int>(n));
        int cnt = 1;
        for (int layer = 0; layer < (n + 1) / 2; layer++) {
            // direction 1 - traverse from left to right
            for (int ptr = layer; ptr < n - layer; ptr++) {
                result[layer][ptr] = cnt++;
            }
            // direction 2 - traverse from top to bottom
            for (int ptr = layer + 1; ptr < n - layer; ptr++) {
                result[ptr][n - layer - 1] = cnt++;
            }
            // direction 3 - traverse from right to left
            for (int ptr = n - layer - 2; ptr >= layer; ptr--) {
                result[n - layer - 1][ptr] = cnt++;
            }
            // direction 4 - traverse from bottom to top
            for (int ptr = n - layer - 2; ptr > layer; ptr--) {
                result[ptr][layer] = cnt++;
            }
        }

        return result;
    }
};
```

Java

```
class Solution {
    public int[][] generateMatrix(int n) {
        int[][] result = new int[n][n];
        int cnt = 1;
        for (int layer = 0; layer < (n + 1) / 2; layer++) {
            // direction 1 - traverse from left to right
            for (int ptr = layer; ptr < n - layer; ptr++) {
                result[layer][ptr] = cnt++;
            }
            // direction 2 - traverse from top to bottom
            for (int ptr = layer + 1; ptr < n - layer; ptr++) {
                result[ptr][n - layer - 1] = cnt++;
            }
            // direction 3 - traverse from right to left
            for (int ptr = layer + 1; ptr < n - layer; ptr++) {
                result[n - layer - 1][n - ptr - 1] = cnt++;
            }
            // direction 4 - traverse from bottom to top
            for (int ptr = layer + 1; ptr < n - layer - 1; ptr++) {
                result[n - ptr - 1][layer] = cnt++;
            }
        }
        return result;
    }
}
```

Python

```
class Solution:
    def generateMatrix(self, n: int) -> List[List[int]]:
        result = [[0 for i in range(n)] for j in range(n)]
        cnt = 1
        for layer in range(0, (n+1)//2) :
            for ptr in range(layer, n - layer):
                result[layer][ptr] = cnt
                cnt += 1
            for ptr in range(layer + 1, n - layer):
                result[ptr][n - layer - 1] = cnt
                cnt += 1
            for ptr in range(layer + 1, n - layer):
                result[n - layer - 1][n - ptr - 1] = cnt
                cnt += 1
            for ptr in range(layer + 1, n - layer - 1):
                result[n - ptr - 1][layer] = cnt
                cnt += 1

        return result
```

Javascript

```
var generateMatrix = function(n) {
    if (n == 0) return [];
    if (n == 1) return[[1]];
    let result = [], num = 1;
    for (let i = 0; i < n; i++) {
        result.push([]);
    }
    let rowStart = 0, rowEnd = n - 1, colStart = 0, colEnd = n - 1;
    while (rowStart <= rowEnd && colStart <= colEnd) {
        // to right
        for (let i = colStart; i <= colEnd; i++) {
            result[rowStart][i] = num;
            num++;
        }
        rowStart++;

        // downwards
        for (let i = rowStart; i <= rowEnd; i++) {
            result[i][colEnd] = num;
            num++;
        }
        colEnd--;

        // to left
        for (let i = colEnd; i >= colStart; i--) {
            result[rowEnd][i] = num;
            num++;
        }
        rowEnd--;

        // upwards
        for (let i = rowEnd; i >= rowStart; i--) {
            result[i][colStart] = num;
            num++;
        }
        colStart++;
    }
    return result;
};
```

Maximum Gap

We will use the pigeonhole principle to solve this problem. We will find the average interval from the minimum and maximum value present in the vector. We will divide the interval into (n - 1) buckets of equal size. Please take a look at this video

C++

```
class Solution {
public:
    class Bucket {
public:
    bool used = false;
    int minval = numeric_limits<int>::max();        // same as INT_MAX
    int maxval = numeric_limits<int>::min();        // same as INT_MIN
};

int maximumGap(vector<int>& nums)
{
    if (nums.empty() || nums.size() < 2)
        return 0;

    int mini = *min_element(nums.begin(), nums.end()),
        maxi = *max_element(nums.begin(), nums.end());

    int bucketSize = max(1, (maxi - mini) / ((int)nums.size() - 1));        // bucket
size or capacity
    int bucketNum = (maxi - mini) / bucketSize + 1;                         // number of
buckets
    vector<Bucket> buckets(bucketNum);

    for (auto&& num : nums) {
        int bucketIdx = (num - mini) / bucketSize;                          // locating
correct bucket
        buckets[bucketIdx].used = true;
        buckets[bucketIdx].minval = min(num, buckets[bucketIdx].minval);
        buckets[bucketIdx].maxval = max(num, buckets[bucketIdx].maxval);
    }

    int prevBucketMax = mini, maxGap = 0;
    for (auto&& bucket : buckets) {
        if (!bucket.used)
            continue;

        maxGap = max(maxGap, bucket.minval - prevBucketMax);
        prevBucketMax = bucket.maxval;
    }

    return maxGap;
}
};
```

Java

```
class Solution {
    public int maximumGap(int[] nums) {
        if(nums.length < 2) return 0;
        int min = nums[0], max = nums[0];
        for(int i : nums){
            min = Math.min(min, i);
            max = Math.max(max, i);
        }
        int n = nums.length;
        int bucketSize = (max-min)/(n-1);
        if(bucketSize == 0) bucketSize++;
        int totalBuckets = (max-min)/bucketSize + 1;

        int[] minBucket = new int[totalBuckets];
        int[] maxBucket = new int[totalBuckets];
        Arrays.fill(minBucket, Integer.MAX_VALUE);

        for(int i = 0; i < n; i++){
            int index = (nums[i]-min)/bucketSize;
            minBucket[index] = Math.min(minBucket[index], nums[i]);
            maxBucket[index] = Math.max(maxBucket[index], nums[i]);
        }
        int prevMax = maxBucket[0], result = 0;
        for(int i = 1; i< totalBuckets; i++){
            if(minBucket[i] == Integer.MAX_VALUE)   continue;
            result = Math.max(result, minBucket[i]-prevMax);
            prevMax = maxBucket[i];
        }
        return result;
    }
}
```

Python

```
class Solution:

    def maximumGap(self, nums: List[int]) -> int:

        if len(nums) <= 1:
            return 0
        mini = min(nums)
        maxi = max(nums)
        G = int(math.ceil((maxi - mini) // (len(nums) - 1)))
        # print (G)
        if G < 1:
            G = 1
        bucket_num = int((maxi - mini) / G + 1)
        # print (bucket_num)
        buckets = [[False, 999999999, -999999999] for i in range(bucket_num)]
        # if G < 1:
        #     G = 1
        for ele in nums:
            # print (ele)
            bucket_index = (ele - mini ) // G
            # print (bucket_index)
            buckets[bucket_index][0] = True
            buckets[bucket_index][1] = min(ele, buckets[bucket_index][1])
            buckets[bucket_index][2] = max(ele, buckets[bucket_index][2])

        prev_max = mini
        max_gap = 0
        for bucket in buckets:
            if (not bucket[0]):
                 continue
            max_gap = max(max_gap, bucket[1] - prev_max)
            prev_max = bucket[2]
        return max_gap
```

C#

```
public class Solution
    {
        public int MaximumGap(int[] nums)
        {
            if (nums.Length < 2)
            {
                return 0;
            }

            if (nums.Length == 2)
            {
                return Math.Abs(nums[0] - nums[1]);
            }

            int min = nums.Min();
            int max = nums.Max();

            int bucketSize = Math.Max(1, (max - min) / (nums.Length - 1));
            int bucketCount = (max - min) / bucketSize + 1;

            (int? min, int? max)[] buckets = new (int? min, int? max)[bucketCount];

            foreach (var num in nums)
            {
                var idx = (num - min) / bucketSize;
                var current = buckets[idx];

                current.min = Math.Min(current.min.GetValueOrDefault(int.MaxValue), num);
                current.max = Math.Max(current.max.GetValueOrDefault(int.MinValue), num);
                buckets[idx] = current;
            }

            int res = 0;

            int prevGapIdx = -1;
            for (int i = 0; i < bucketCount; i++)
            {
                if (!buckets[i].min.HasValue)
                {
                    continue;
                }

                var prev = prevGapIdx >= 0 ? buckets[prevGapIdx].max.Value : min;
                res = Math.Max(res, buckets[i].min.Value - prev);
                prevGapIdx = i;
            }

            return res;
        }
    }
```

Javascript

```
/**
 * @param {number[]} nums
 * @return {number}
 */

var maximumGap = function(nums) {
    let max = Math.max(...nums);
    let min = Math.min(...nums);
    let bucketNum = nums.length;
    let bucketSize = (max - min) / (bucketNum - 1);

    let bucketMax = new Array(bucketNum).fill(-Infinity);
    let bucketMin = new Array(bucketNum).fill(Infinity);

    for (let i = 0; i < bucketNum; ++i) {
        let index = Math.floor((nums[i] - min) / bucketSize);
        bucketMax[index] = Math.max(bucketMax[index], nums[i]);
        bucketMin[index] = Math.min(bucketMin[index], nums[i]);
    }
    let preMax = min;
    let res = 0;
    for (let i = 0; i < bucketNum; ++i) {
        if (bucketMax[i] === -Infinity || bucketMin[i] === Infinity) continue;
        res = Math.max(res, bucketMin[i] - preMax);
        preMax = bucketMax[i];
    }
    return res;
};
```