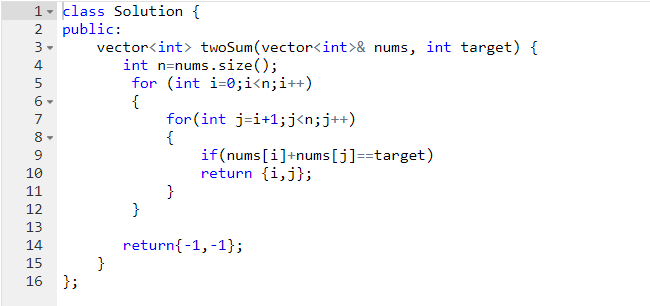
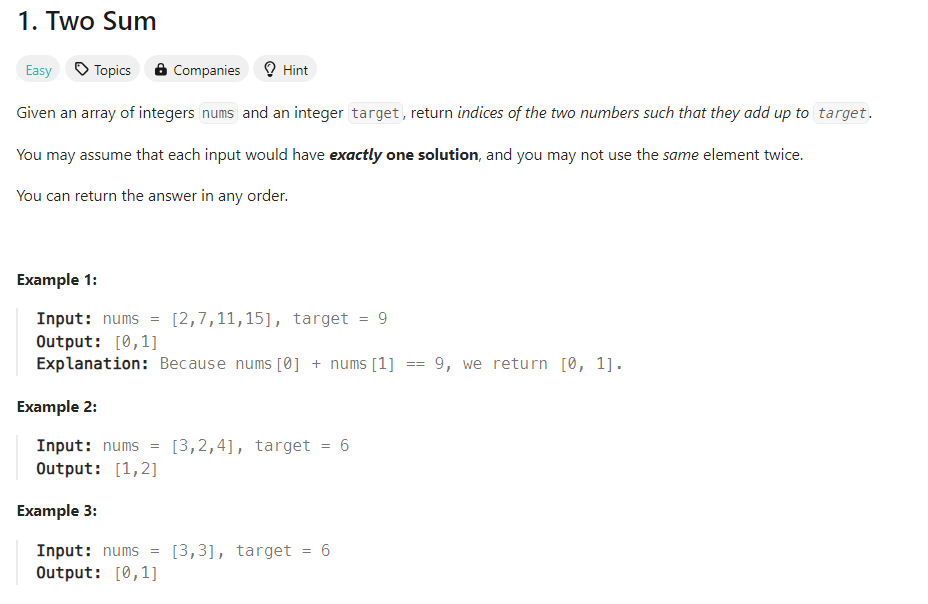
(1)

class Solution {

public:

vector<int> twoSum(vector<int>& nums, int target) {

int n=nums.size();

for (int i=0;i<n;i++)

{

for(int j=i+1;j<n;j++)

{

if(nums[i]+nums[j]==target)

return {i,j};

}

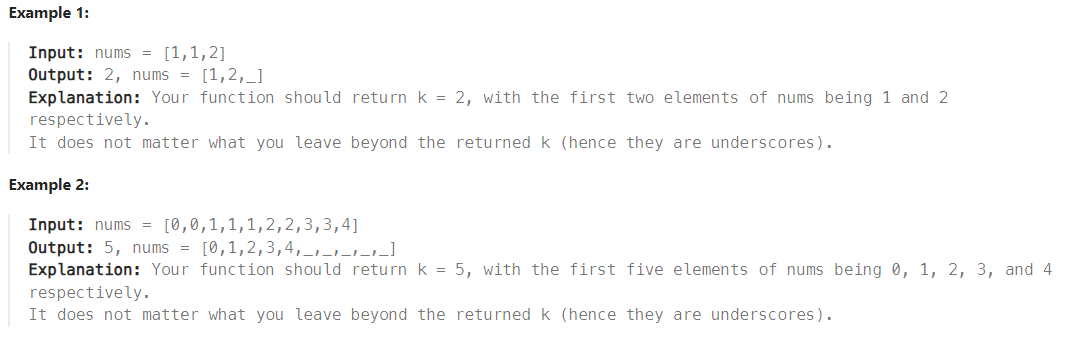
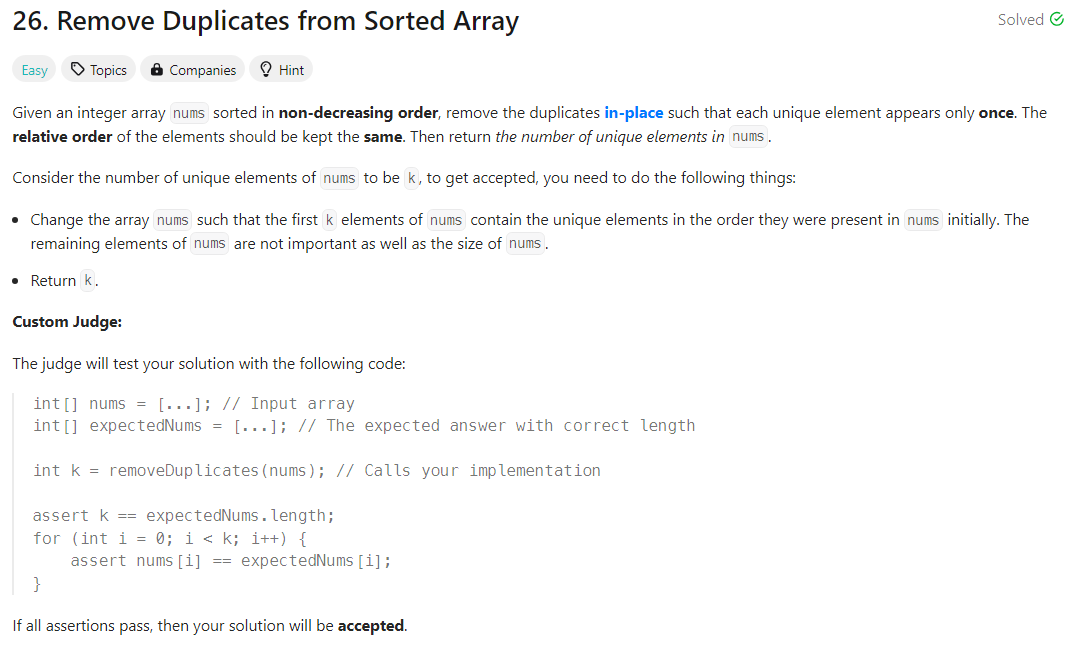
}

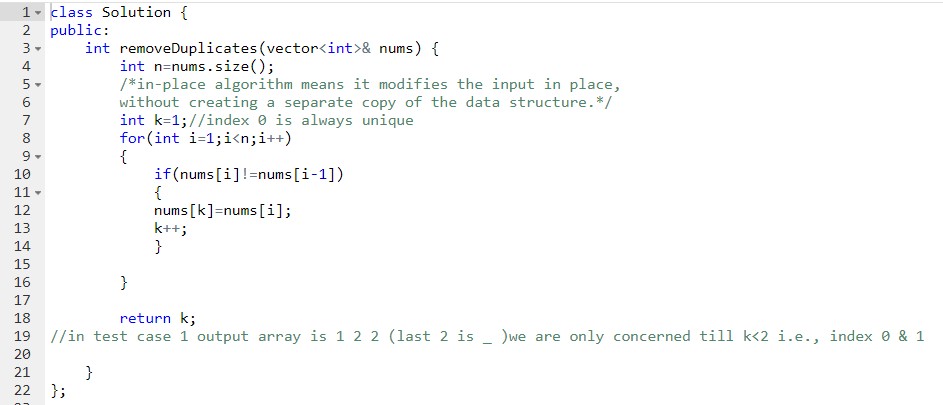
return{-1,-1};

}

};

(2)





class Solution {

public:

int removeDuplicates(vector<int>& nums) {

int n=nums.size();

/\*in-place algorithm means it modifies the input in place,

without creating a separate copy of the data structure.\*/

int k=1;//index 0 is always unique

for(int i=1;i<n;i++)

{

if(nums[i]!=nums[i-1])

{

nums[k]=nums[i];

k++;

}

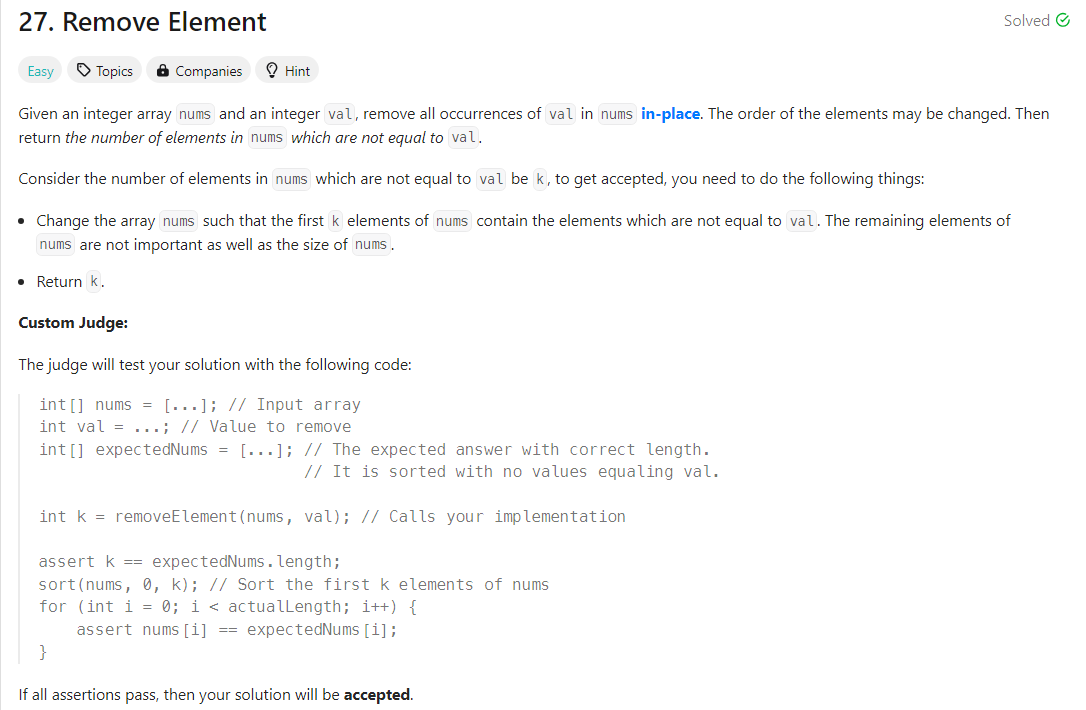
}

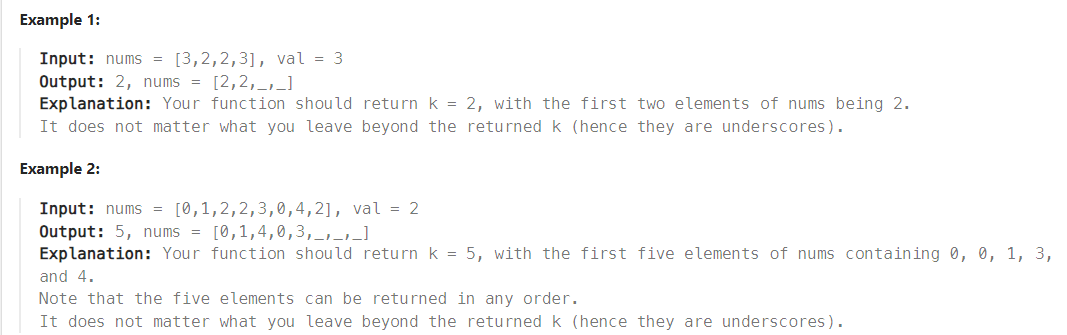
return k;

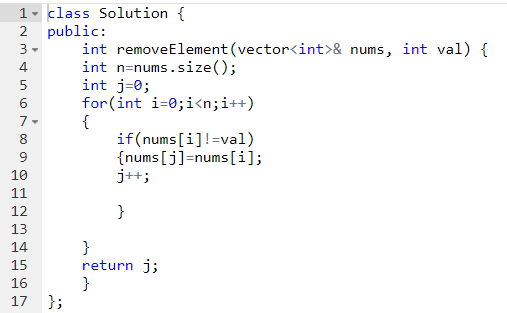
//in test case 1 output array is 1 2 2 (last 2 is \_ )we are only concerned till k<2 i.e., index 0 & 1

}

};

(3) 





class Solution {

public:

int removeElement(vector<int>& nums, int val) {

int n=nums.size();

int j=0;

for(int i=0;i<n;i++)

{

if(nums[i]!=val)

{nums[j]=nums[i];

j++;

}

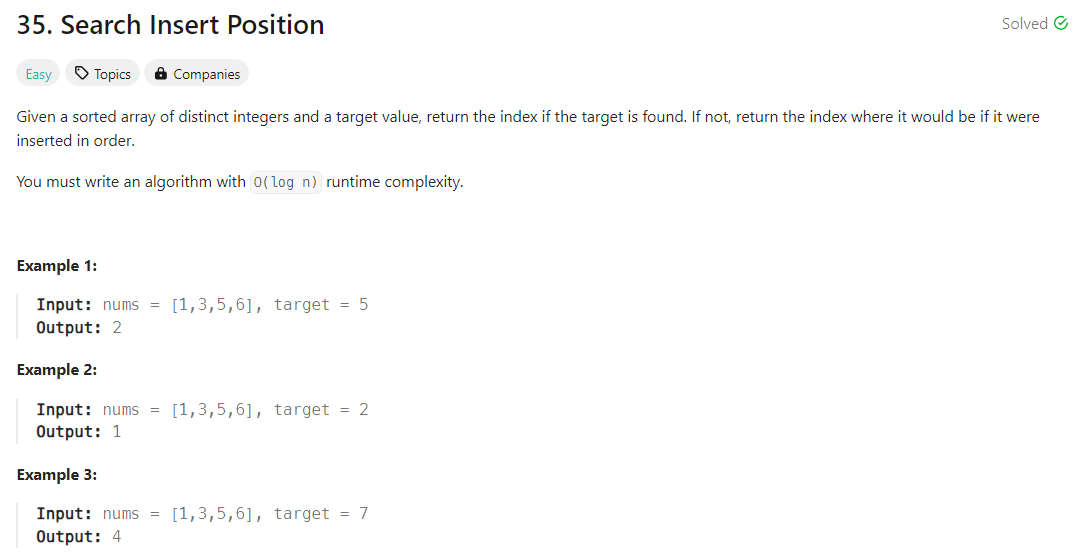
}

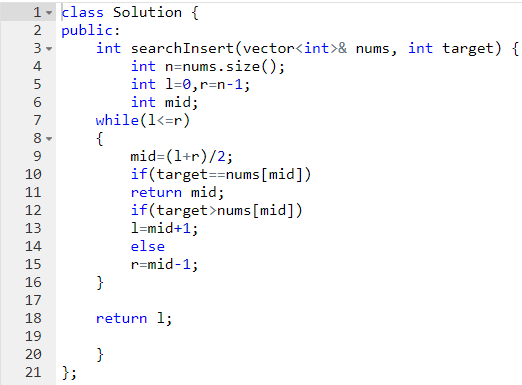
return j;

}

};

(4)





class Solution {

public:

int searchInsert(vector<int>& nums, int target) {

int n=nums.size();

int l=0,r=n-1;

int mid;

while(l<=r)

{

mid=(l+r)/2;

if(target==nums[mid])

return mid;

if(target>nums[mid])

l=mid+1;

else

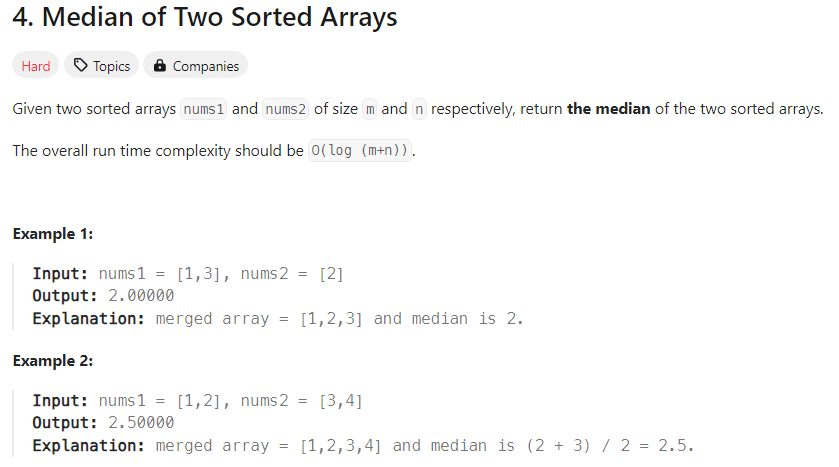
r=mid-1;

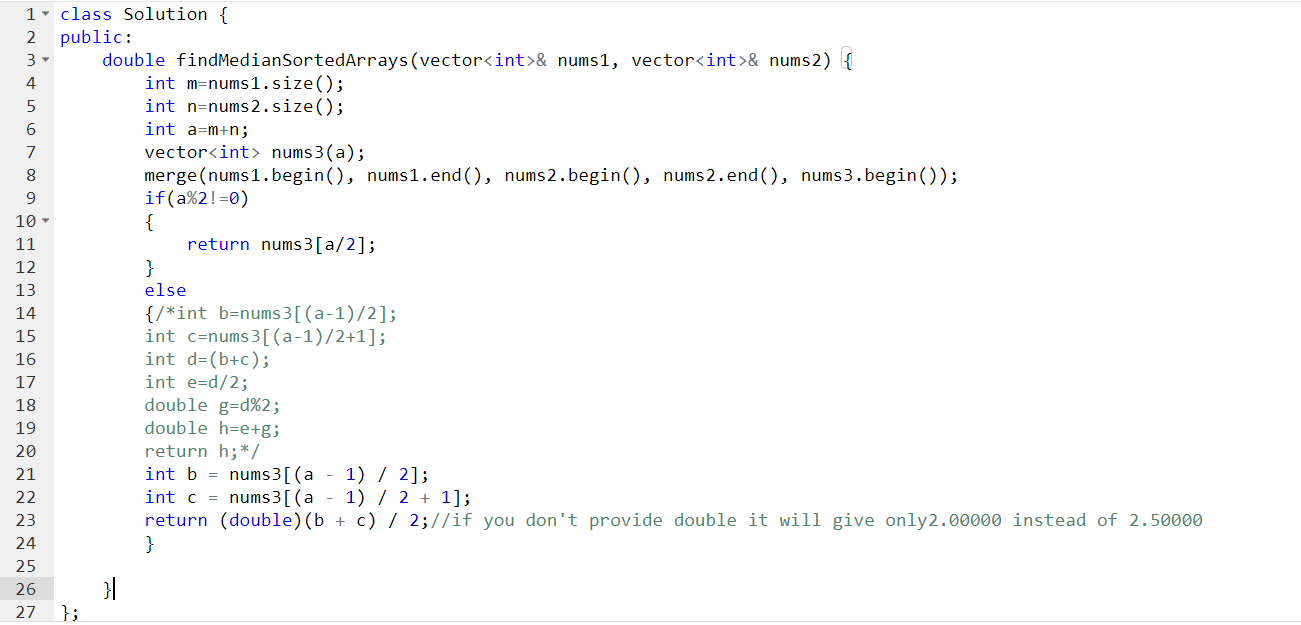
}

return l;

}

};

(5) 



class Solution {

public:

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

int m=nums1.size();

int n=nums2.size();

int a=m+n;

vector<int> nums3(a);

merge(nums1.begin(), nums1.end(), nums2.begin(), nums2.end(), nums3.begin());

if(a%2!=0)

{

return nums3[a/2];

}

else

{/\*int b=nums3[(a-1)/2];

int c=nums3[(a-1)/2+1];

int d=(b+c);

int e=d/2;

double g=d%2;

double h=e+g;

return h;\*/

int b = nums3[(a - 1) / 2];

int c = nums3[(a - 1) / 2 + 1];

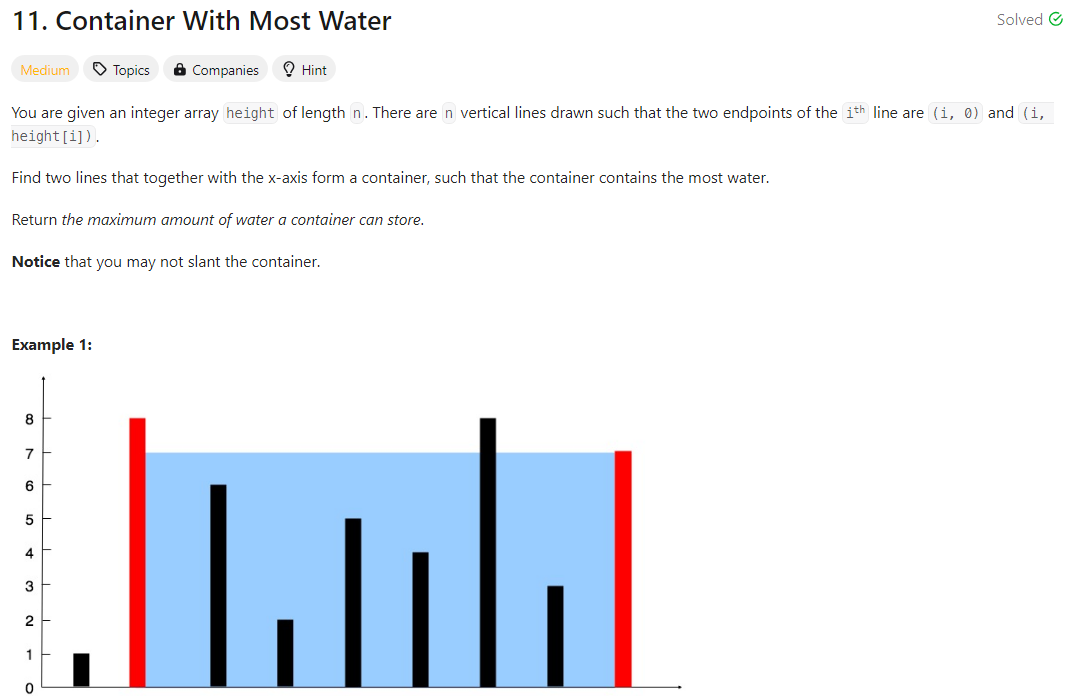
return (double)(b + c) / 2;//if you don't provide double it will give only2.00000 instead of 2.50000

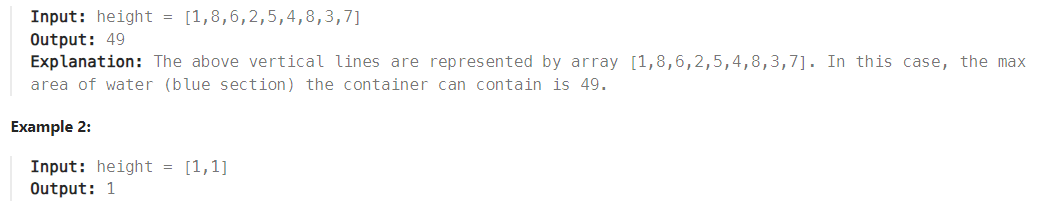
}

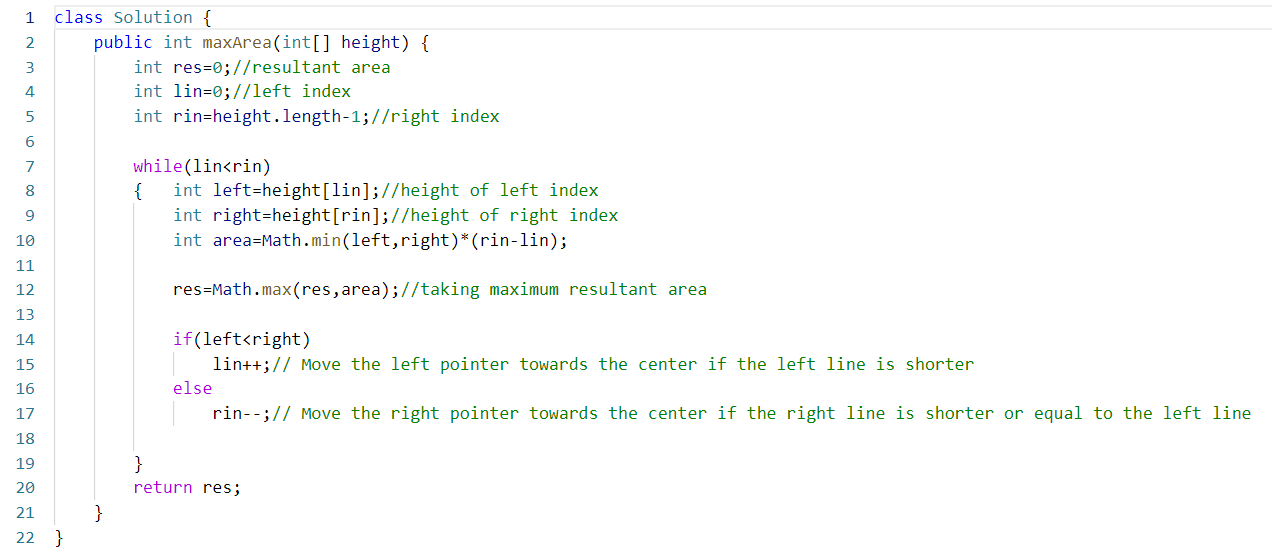
}

};

(6)







class Solution {

    public int maxArea(int[] height) {

        int res=0;//resultant area

        int lin=0;//left index

        int rin=height.length-1;//right index

        while(lin<rin)

        {   int left=height[lin];//height of left index

            int right=height[rin];//height of right index

            int area=Math.min(left,right)\*(rin-lin);

            res=Math.max(res,area);//taking maximum resultant area

            if(left<right)

                lin++;// Move the left pointer towards the centre if the left line is shorter

            else

                rin--;// Move the right pointer towards the centre if the right line is shorter or equal to the left line

        }

        return res;

    }

}