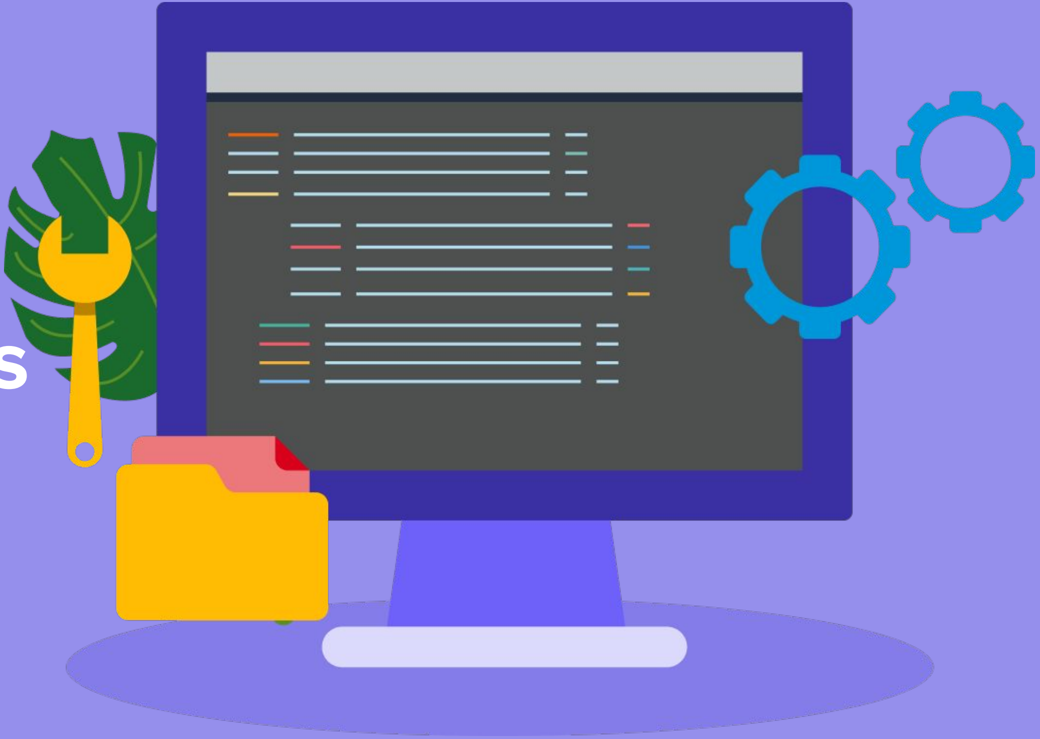
Basic Problem Solving: Arrays and Objects

Relevel
by Unacademy



Problem Solving

1. Sort array of 0 1
2. Sort array of 0 1 2
3. Target sum pair in an array
4. Target Sum Triplet in an array
5. Rain water trapping
6. Find element that occurs once in the array where rest of the element appear twice
7. Minimum Absolute Difference

Sort Array of 0, 1

Given an array having 0,1 as input. We need to write a function that sorts the array in ascending order without using inbuilt sort function

Expected time complexity :O(n)

Expected space complexity:O(1)

Example:

Input: {0, 1, 1,0, 0, 1}

Output: {0, 0, 0, 1, 1, 1}

Input: {0,0,1,1,0,1,0}

Output: {0,0,0,0,1,1,1}

Sort Array of 0, 1

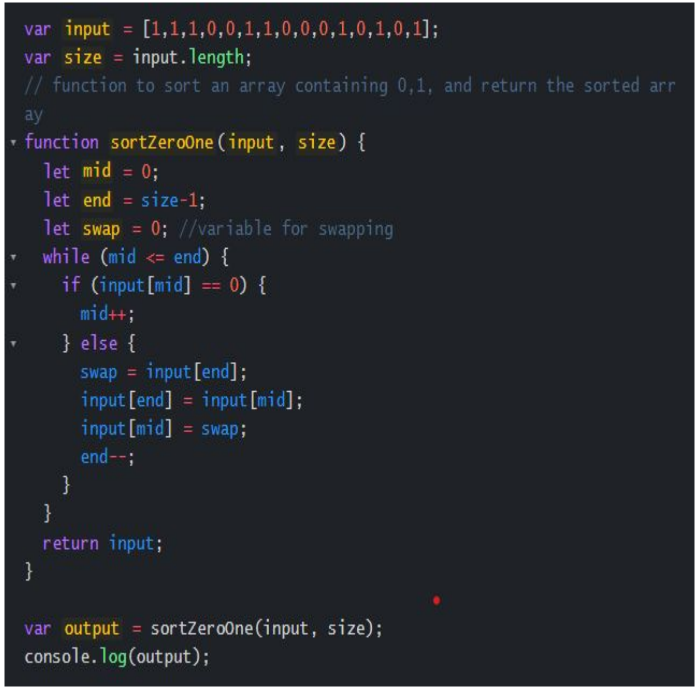
Approach:
We divide the array into three parts. Part one containing the zeros, second part containing the values which can be zero or one and the last part containing one. If the element in second part is zero will reduce the size of second part, if the element is one will move it to the third part and reduce the size of second part.

Algorithm
1) We will have two indices mid=0,end=N and there are three parts
    (0,mid):the elements here will be 0
    (mid,end):the elements here can be 0,1
    (end,N):the elements here will be 1
    Where N = size of the array
2) We will be traversing the array from start to end until mid is less than end
3) If the mid element is 0,increment mid by one
4) If the mid element is 1,will swap with the element at index end and decrement the value of end by one
6) We will continue doing this till the mid value is less than the end.

Sort Array of 0, 1

Implementation:

Code: https://jsfiddle.net/041zoLwu/1/



```
var input = [1,1,1,0,0,1,1,0,0,0,1,0,1,0,1];
var size = input.length;
// function to sort an array containing 0,1, and return the sorted arr
ay
function sortZeroOne(input, size) {
  let mid = 0;
  let end = size-1;
  let swap = 0; //variable for swapping
  while (mid <= end) {
    if (input[mid] == 0) {
      mid++;
    } else {
      swap = input[end];
      input[end] = input[mid];
      input[mid] = swap;
      end--;
    }
  }
  return input;
}

var output = sortZeroOne(input, size);
console.log(output);
```

Sort Array of 0, 1

Complexity Analysis:

Time Complexity: O(n)
Only one iteration of the array is made

Space Complexity: O(1)
No extra space is required

Sort Array of 0, 1, 2

Given an array having 0,1,2. We need to write a function that sorts the array in ascending order without using inbuilt sort function

Expected time complexity :O(n)

Expected space complexity:O(1)

Example:

Input: {0, 1, 2, 2, 1, 2}

Output: {0, 1, 1, 2, 2, 2}

Input: {2,2,1,0,0,1,2,2}

Output: {0,0,1,1,2,2,2,2}

Sort Array of 0, 1, 2

Approach:

First will divide the array into four parts. Part one containing the zeros, second part containing the ones, third part containing the values which can be zero, one or two and the last part containing two. If the element in third part is zero will move it to the first part and reduce the size of third part, if the element is one will reduce only the size of third part and if the element is two will move it to the fourth part and reduce the size of third part.

Algorithm

1) We will have three indices start=0,mid=0,end=N and there are four parts

    (0,start):the elements here will be 0

    (start,mid):the elements here will be 1

    (mid,end):the elements here can be 0,1,2

    (end,N):the elements here will be 2

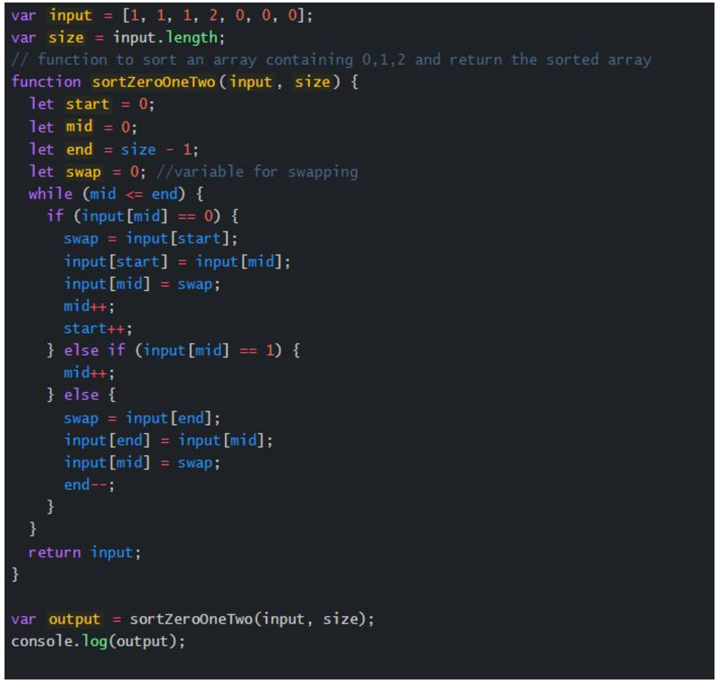
    Where N = size of the array

2) We will be traversing the array from start to end until mid is less than end
3) If the mid element is 0 will swap element at start index with mid and increment both mid and start by one
4) If the element is 1, increment the mid by one.
5) If the element is 2,will swap with the element at index end and decrement the value of end by one
6) We will continue doing this till the mid value is less than the end.

Sort Array of 0, 1

Implementation:

Code: https://jsfiddle.net/041zoLwu/



```
var input = [1, 1, 1, 2, 0, 0, 0];
var size = input.length;
// function to sort an array containing 0,1,2 and return the sorted array
function sortZeroOneTwo(input, size) {
  let start = 0;
  let mid = 0;
  let end = size - 1;
  let swap = 0; //variable for swapping
  while (mid <= end) {
    if (input[mid] == 0) {
      swap = input[start];
      input[start] = input[mid];
      input[mid] = swap;
      mid++;
      start++;
    } else if (input[mid] == 1) {
      mid++;
    } else {
      swap = input[end];
      input[end] = input[mid];
      input[mid] = swap;
      end--;
    }
  }
  return input;
}

var output = sortZeroOneTwo(input, size);
console.log(output);
```

Sort Array of 0, 1

Complexity Analysis:

Time Complexity: O(n)

Only one iteration of the array is made

Space Complexity: O(1)

No extra space is required

Target Sum Pair in an Array

Given an array A with a number x. We need to write a function which takes array A and x as argument and return a pair from the array A whose sum is equal to x. If no pair exists then return empty array.

Example :
Input: {0,1,2,-2,4,5,6}
Sum = 5
Output: {1,4}
There are many approaches to solve this problem

Target Sum Pair in an Array

Approach 1:

This approach will include sorting and two pointer technique. First we will sort the array and then will two pointer technique. We will take two pointers and put at the begin and the end of the array. Now will iterate and if the sum of the two pointers is greater then x will decrement the right pointer and if the sum is less than x will increment the left pointer. We will continue doing this until left pointer is less than right pointer.

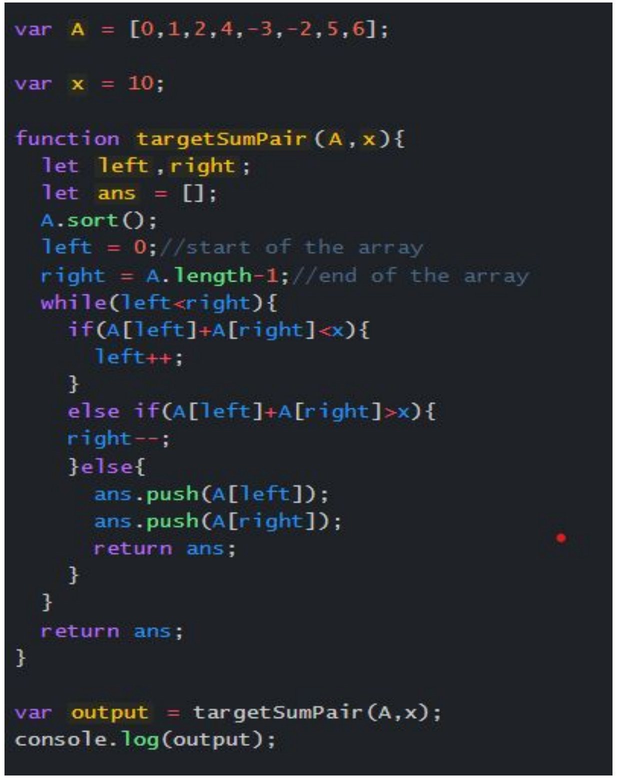
Algorithm:

```
1) Sort the array A in ascending order
2) Initialize two pointers
        1) Leftmost index:left = 0,
        2) Rightmost index:right = A.length -1;
3) Now wil iterate till l<r and three conditions can be there
        1) if A[left]+A[right] less than x,increment left by one.
        2) if A[left]+A[right] greater than x,decrement right by one
        3) if A[left]+A[right]==x return the pair as an array
4) If no pair exists then return an empty array.
```

Target Sum Pair in an Array

Implementation:

Code: https://jsfiddle.net/v6u0Lyaw/



```
var A = [0,1,2,4,-3,-2,5,6];

var x = 10;

function targetSumPair(A,x){
  let left,right;
  let ans = [];
  A.sort();
  left = 0;//start of the array
  right = A.length-1;//end of the array
  while(left<right){
    if(A[left]+A[right]<x){
      left++;
    }
    else if(A[left]+A[right]>x){
    right--;
    }else{
      ans.push(A[left]);
      ans.push(A[right]);
      return ans;
    }
  }
  return ans;
}

var output = targetSumPair(A,x);
console.log(output);
```

Target Sum Pair in an Array

Complexity Analysis:

Time Complexity:

Depends on which sorting algo used:

1) Quick Sort: O(N^2) worst case scenario
2) Merge Sort: O(NLOGN) worst case scenario

Space Complexity:

Depends on which sorting algo used:

1) Merge Sort: O(N)
2) Quick Sort: O(logN)

Target Sum Pair in an Array

Approach 2:

This approach involves hashing concept.We will create a hash table for storing elements in array. We will loop the array through a pointer curr and search for x- A[curr] in the map and if its present will return this values.

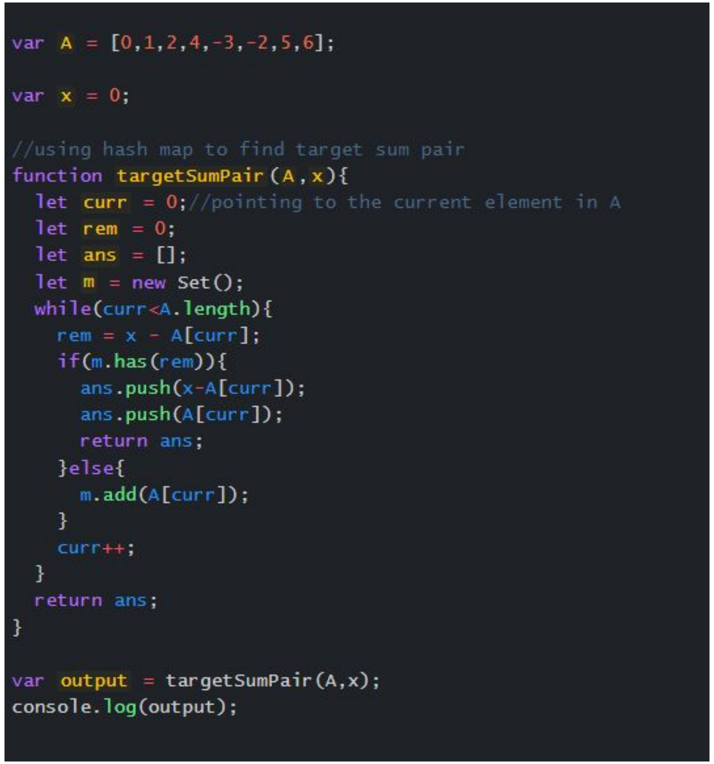
Algorithm:

```
1) Create an empty hash table m;
2) Create a pointer curr pointing to the start of the array and iterate over the array
        1) if(m[x-A[curr]) is set in hash,then return an array [x - A[curr], A[curr] ]
        2) If(m[x-A[curr]) is not set in hash,insert A[curr] in hash.
```

Target Sum Pair in an Array

Implementation:

Code: https://jsfiddle.net/v6u0Lyaw/1/


```
var A = [0,1,2,4,-3,-2,5,6];

var x = 0;

//using hash map to find target sum pair
function targetSumPair(A,x){
  let curr = 0;//pointing to the current element in A
  let rem = 0;
  let ans = [];
  let m = new Set();
  while(curr<A.length){
    rem = x - A[curr];
    if(m.has(rem)){
      ans.push(x-A[curr]);
      ans.push(A[curr]);
      return ans;
    }else{
      m.add(A[curr]);
    }
    curr++;
  }
  return ans;
}

var output = targetSumPair(A,x);
console.log(output);
```

Target Sum Pair in an Array

Complexity Analysis:

Time Complexity: O(N)

We are traversing the complete array

Space Complexity: O(N)

As all the elements are stored in hash table

Target Sum Triplet in an Array

Given an array A with a number x. We need to write a function which takes array A and x as argument and return a triplet from the array A whose sum is equal to x. If no pair exists then return empty array.

Example :
Input: {0,1,2,-2,4,5,6}
Sum = 7
Output: {0,1,6}
There are many approaches to solve this problem

Target Sum Triplet in an Array

Approach 1:
This approach uses sorting and two pointer technique.We will be traversing through the array by a pointer i which fix the first element of triplet. Then will use two pointer technique to find a pair in the remaining array whose sum is x - A[i]. Here the starting pointer will be i+1 and end will the right most index of array

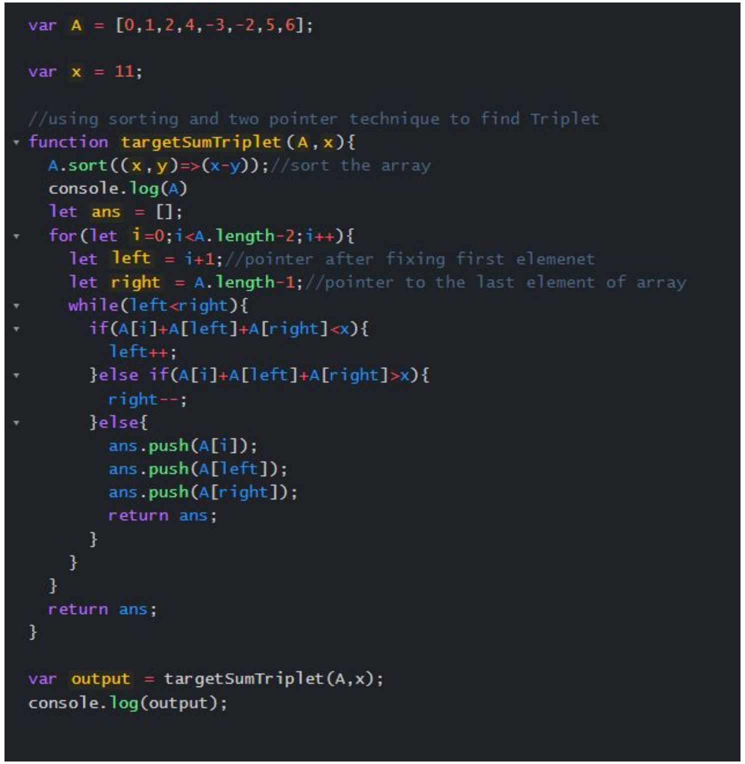
Algorithm:

1) Sort the array A in ascending order
2) Will loop over the array and choose the first element of the triplet, A[i]
3) We will then create two pointers one pointing to i+1 and the second one at A.length-1
    a) If the sum of two pointers is less than x-A[i], increment the left pointer
    b) If the sum of two pointers is greater than x-A[i], decrement the right pointer
    c) If the sum of two pointers is equal to x, return the triplet as an array

Target Sum Triplet in an Array

Implementation:

Code: https://jsfiddle.net/v6u0Lyaw/2/



```javascript
var A = [0,1,2,4,-3,-2,5,6];

var x = 11;

//using sorting and two pointer technique to find Triplet
function targetSumTriplet(A,x){
  A.sort((x,y)=>(x-y));//sort the array
  console.log(A)
  let ans = [];
  for(let i=0;i<A.length-2;i++){
    let left = i+1;//pointer after fixing first elemenet
    let right = A.length-1;//pointer to the last element of array
    while(left<right){
      if(A[i]+A[left]+A[right]<x){
        left++;
      }else if(A[i]+A[left]+A[right]>x){
        right--;
      }else{
        ans.push(A[i]);
        ans.push(A[left]);
        ans.push(A[right]);
        return ans;
      }
    }
  }
  return ans;
}

var output = targetSumTriplet(A,x);
console.log(output);
```

Target Sum Triplet in an Array

Complexity Analysis:

Time Complexity: O(N^2)

As the worst case complexity for sorting is O(N^2) and we are also running two nested loops one for choosing the first element O(N) and two pointer traversal for choosing the other elements giving complexity O(N) which gives overall complexity O(N^2).

Space Complexity: O(N)

Depends totally on the sorting algorithm used

1) Merge Sort: O(N)

2) Quick Sort: O(logN)

Target Sum Triplet in an Array

Approach 2:

This approach uses hashing based algo.We will be traversing through the array by a pointer i fixing our first element and then will run a loop from i+1 to A.length by a pointer j.I n the second loop we will create a hashmap to store elements from i+1 to j-1. and search for x-A[i]-A[j]. If it’s there then will return the array with the triplet

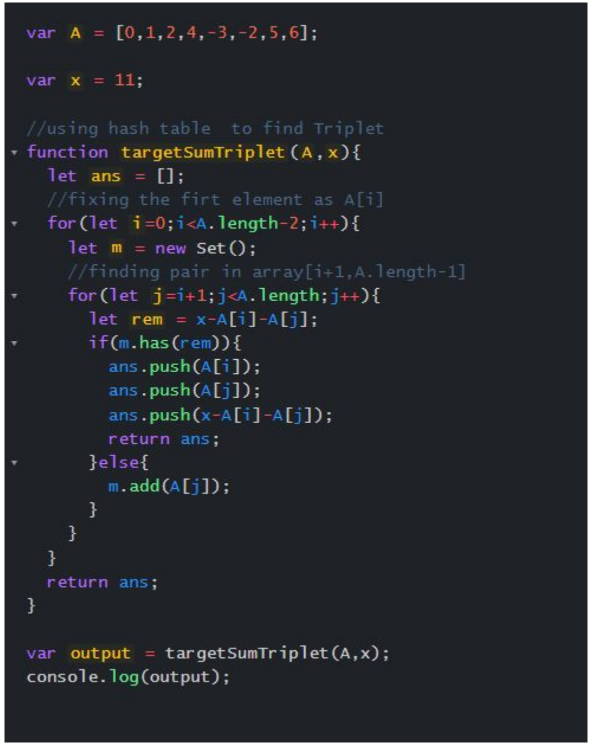
Algorithm:

```
1) Loop the array from start to end by counter i
2) Create a hashmap to store elements
3) Run second loop from i+1 to A.length
        1) If the value x-A[i]-A[j] is set then will return an array with elements A[i], A[j], x-A[i]-A[j]
        2) Insert the A[j] element in set
```

Target Sum Triplet in an Array

Implementation:

Code: https://jsfiddle.net/v6u0Lyaw/3/



```
var A = [0,1,2,4,-3,-2,5,6];

var x = 11;

//using hash table  to find Triplet
function targetSumTriplet(A,x){
  let ans = [];
  //fixing the firt element as A[i]
  for(let i=0;i<A.length-2;i++){
    let m = new Set();
    //finding pair in array[i+1,A.length-1]
    for(let j=i+1;j<A.length;j++){
      let rem = x-A[i]-A[j];
      if(m.has(rem)){
        ans.push(A[i]);
        ans.push(A[j]);
        ans.push(x-A[i]-A[j]);
        return ans;
      }else{
        m.add(A[j]);
      }
    }
  }
  return ans;
}

var output = targetSumTriplet(A,x);
console.log(output);
```

Target Sum Triplet in an Array

Complexity Analysis:

Time Complexity: O(N^2)

As two loops are involved, in worst case the array is sorted it can take O(N^2) time.

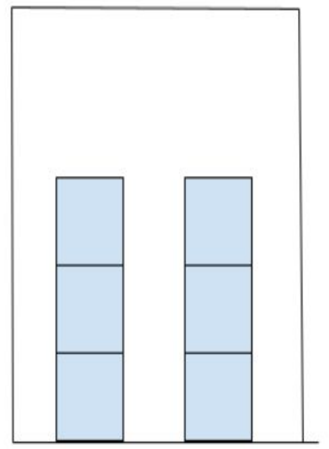
Space Complexity: O(N)

As hash table is used to store the values present in the array.

Rain water trapping

Given an array of non negative integers representing the height of each block where the width of each block is 1, we need to find out maximum rain water that can be trapped using blocks.

Example:
Input: {3,0,3}
Output: 3 as three units of water can be stored between this two blocks.
Approach: There are many approaches for this problem. Will discuss each one of them one by one.



Rain water trapping

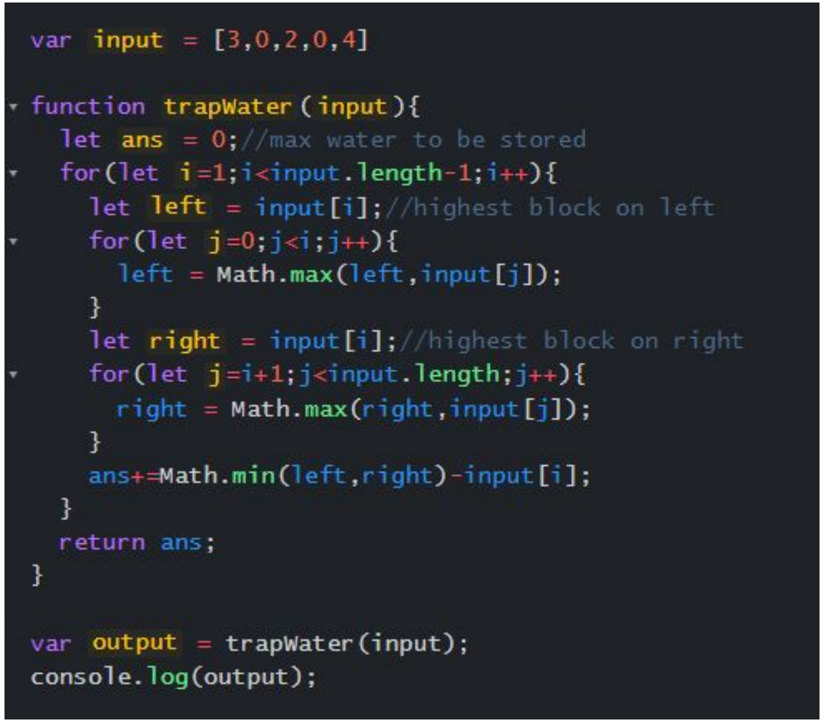
Approach 1:
We will traverse the array from start to end and for each element will find the highest bars on left of this element and right of this element. So the amount of water it can store will be the minimum of left and right height blocks-height of the current block

Algorithm:
1) Traverse the array from beginning to end
2) For each element iterate the array from start to present element and find the highest block in left,similarly start from current element index to end and find the highest block in right
3) The amount of water that can be current stored for current block is min(left,right)-current block height
4) We will have a variable to store the sum of water for all the elements of array

Rain water trapping

Implementation

Code: https://jsfiddle.net/p85ehvyx/



```
var input = [3,0,2,0,4]

function trapWater(input){
  let ans = 0;//max water to be stored
  for(let i=1;i<input.length-1;i++){
    let left = input[i];//highest block on left
    for(let j=0;j<i;j++){
      left = Math.max(left,input[j]);
    }
    let right = input[i];//highest block on right
    for(let j=i+1;j<input.length;j++){
      right = Math.max(right,input[j]);
    }
    ans+=Math.min(left,right)-input[i];
  }
  return ans;
}

var output = trapWater(input);
console.log(output);
```

Rain water trapping

Complexity Analysis:

Time Complexity: O(N^2)

As there is nested loop

Space Complexity: O(1)

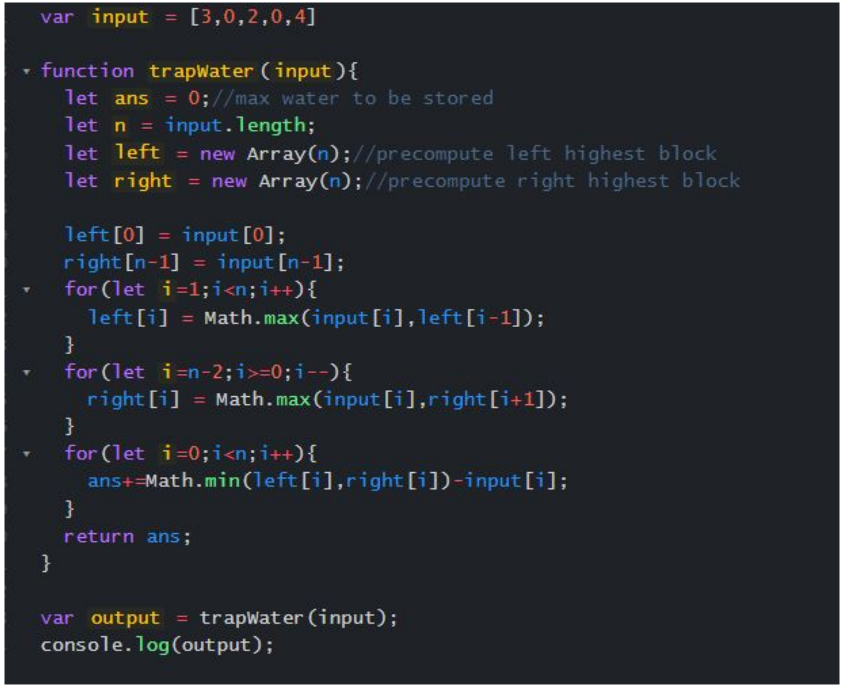
Rain water trapping

Approach 2:
Upper mentioned approach’s time complexity can be reduced if we can precompute the left and right element for every element of the array which will require extra space but time complexity will be reduced. We will create two array for storing the left and right values and use it for computation.

Rain water trapping

Implementation:

Code: https://jsfiddle.net/p85ehvyx/1/



```
var input = [3,0,2,0,4]

function trapWater(input){
  let ans = 0;//max water to be stored
  let n = input.length;
  let left = new Array(n);//precompute left highest block
  let right = new Array(n);//precompute right highest block

  left[0] = input[0];
  right[n-1] = input[n-1];
  for(let i=1;i<n;i++){
    left[i] = Math.max(input[i],left[i-1]);
  }
  for(let i=n-2;i>=0;i--){
    right[i] = Math.max(input[i],right[i+1]);
  }
  for(let i=0;i<n;i++){
    ans+=Math.min(left[i],right[i])-input[i];
  }
  return ans;
}

var output = trapWater(input);
console.log(output);
```

Rain water trapping

Complexity Analysis:

Time Complexity: O(N)

As we are traversing the array constant times

Space Complexity: O(N)

As we are having left and right array for finding highest block on left and right side

Rain water trapping

Approach 3:
We can reduce the space complexity by using two variables to store the left maximum and right maximum upto that point

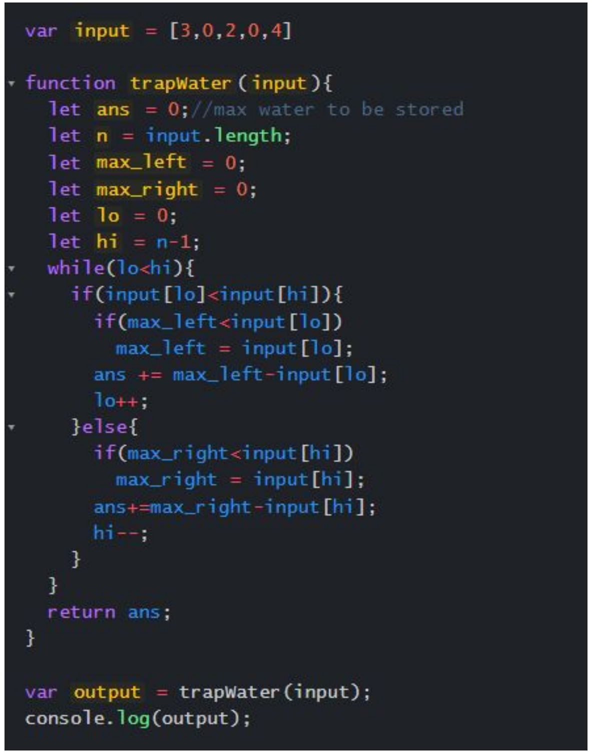
Algorithm:

```
1) Take two pointers lo and hi,lo pointing to the start of array and hi pointing to the end of array
2) Create two variables max_left and max_right for storing max left and max right block.
3) While lo<hi
        1) if input[lo]<input[hi] update max_left,res and increment lo by one
        2) if input[lo]>=input[hi] update max_right,res and decrement hi by one
4) Return the result
```

Rain water trapping

Implementation:

Code: https://jsfiddle.net/p85ehvyx/2/



```
var input = [3,0,2,0,4]

function trapWater(input){
  let ans = 0;//max water to be stored
  let n = input.length;
  let max_left = 0;
  let max_right = 0;
  let lo = 0;
  let hi = n-1;
  while(lo<hi){
    if(input[lo]<input[hi]){
      if(max_left<input[lo])
        max_left = input[lo];
      ans += max_left-input[lo];
      lo++;
    }else{
      if(max_right<input[hi])
        max_right = input[hi];
      ans+=max_right-input[hi];
      hi--;
    }
  }
  return ans;
}

var output = trapWater(input);
console.log(output);
```

Rain water trapping

Complexity Analysis:

Time Complexity: O(N)

As we are traversing the array constant times

Space Complexity: O(1)

As we are using two pointers to store the left max and right max

Find element that occurs once in the array where rest of the element appear twice

Given an array where every element appears twice leaving one of the element.We need to find the element which appears once in the array.
Example:
Input: {4,3,5,6,7,7,6,5,3}
Output: 4

Approach: There are many approaches .We will discuss some of them and implement the one with least time complexity and space complexity

First approach: will be two pick one element from the array and search for occurrence of that element leaving the index of the picked element and if the picked element has a single occurrence then return the picked element.
Time Complexity: O(N^2)
As we will be traversing the array twice ,first for picking the element and second time for searching the same element leaving the picked element.
Space Complexity: O(1) No extra space is required.

Find element that occurs once in the array where rest of the element appear twice

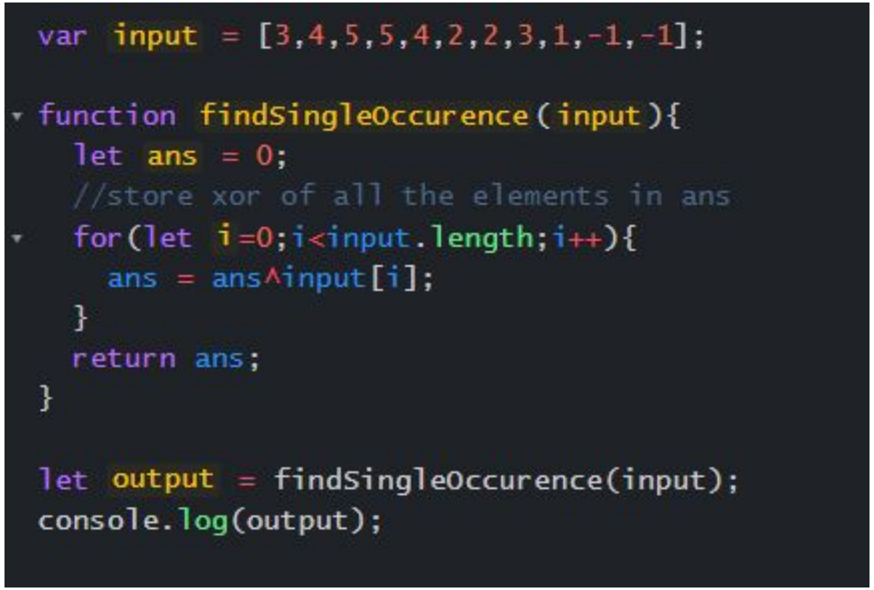
Second approach: will be to create a hash table to store the array elements as key and count of the element as its value.Traverse the array again and return the element with count one
Time Complexity:O(N)
As we are traversing the array twice,once for storing in hash table and second time for checking the element with count one,so the complexity will be O(N)
Space Complexity:O(N)
As we are storing array elements in hash table so the space complexity will be O(N)

Best approach: will be to use some properties of XOR which are as follows.
1) XOR of a number with itself returns 0
    Eg: 3 XOR 3 = 0
2) XOR of a number with 0 return the number itself
    Eg: 3 XOR 0 = 3
If there are elements in pairs in the array leaving one element, if we do the XOR of all the elements present in the array, pairwise elements will return 0 and the XOR of a single occurrence element with 0 will return the single occurrence element.

Find element that occurs once in the array where rest of the element appear twice

Implementation:

Code:https://jsfiddle.net/1os7x09a/



```
var input = [3,4,5,5,4,2,2,3,1,-1,-1];

function findSingleOccurence(input){
  let ans = 0;
  //store xor of all the elements in ans
  for(let i=0;i<input.length;i++){
    ans = ans^input[i];
  }
  return ans;
}

let output = findSingleOccurence(input);
console.log(output);
```

Find element that occurs once in the array where rest of the element appear twice

Complexity Analysis:

Time Complexity: O(N)

We are traversing the array once

Space Complexity: O(1)

We are having only one variable to store the result of XOR.

Minimum Absolute Difference

Given an array of integers and we need to find all the pairs in the array which have the minimum absolute difference and return the array

Example:
Input: [3,4,5,5,4,2,2,3,1,-1,-1];
Output: [[-1,-1], [2,2], [3,3], [4,4], [5,5]]

Minimum Absolute Difference

Approach:

First we need to find the minimum absolute difference and then using that we can get the element pairs. If we sort the array in ascending order minimum absolute difference can be taken by comparing the difference between the two consecutive elements. We need to traverse the sorted array and check if the absolute difference between them is equal to minimum absolute one and if it's then enter this pair of elements into the result array.

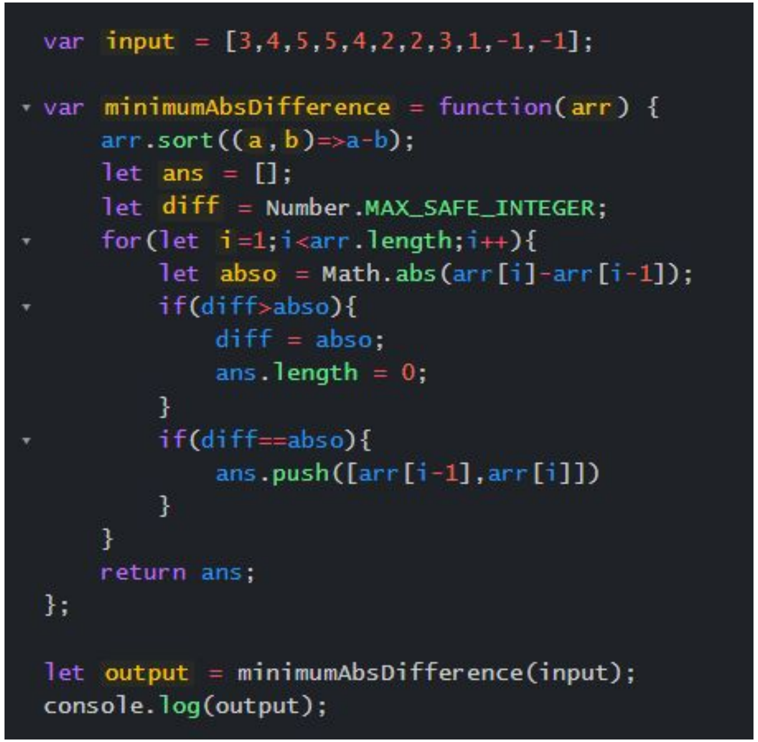
Algorithm:

1) Sort the array A in ascending order and create a variable diff to store minimum absolute difference with max integer value
2) Traverse the array starting from index position one
    1) If (diff>absolute(A[i]-A[i-1]) then change the value of diff to the latest consecutive difference and remove all the elements from the answer array.
    2) if(diff=absolute(A[i]-A[i-1]) then insert A[i], A[i-1] in the answer array

Minimum Absolute Difference

Implementation:

Code:https://jsfiddle.net/1os7x09a/1/



```
var input = [3,4,5,5,4,2,2,3,1,-1,-1];

var minimumAbsDifference = function(arr) {
    arr.sort((a,b)=>a-b);
    let ans = [];
    let diff = Number.MAX_SAFE_INTEGER;
    for(let i=1;i<arr.length;i++){
        let abso = Math.abs(arr[i]-arr[i-1]);
        if(diff>abso){
            diff = abso;
            ans.length = 0;
        }
        if(diff==abso){
            ans.push([arr[i-1],arr[i]])
        }
    }
    return ans;
};

let output = minimumAbsDifference(input);
console.log(output);
```

Minimum Absolute Difference

Complexity Analysis:
Time complexity:
As we are sorting the array depending on the type of sort it can vary
Merge Sort: O(NLOGN)
Quick Sort: O(N^2)

Space Complexity:
Merge Sort: O(N)
Quick Sort: O(LOGN)

Practice Questions

1) Write code for creating a new sorted array from two sorted arrays in O(n+m) time complexity where n,m are the size of the unsorted arrays.

2) Write code for finding the smallest element in an array

Thank You