Problem Solving on Sorting

Relevel
by Unacademy

Problem on Sorting

- K-Closest points to origin
- Sort array in wave form
- Meeting Rooms
- Max Chunks To Make Array Sorted
- Maximum Gap
- Minimum Number of Moves to Seat Everyone
- Minimum Increment to Make Array Unique

K-Closest points to origin

Given two array of points one representing the x-coordinate and one representing the y-coordinate and a integer k,return the k closest points to origin(0,0)
The distance between two points is Euclidean distance sqr_root((x1-x2)^2+(y1-y2)^2)
Answer can be written in any order

Example:

Input :x = [1,2,3];
y = [2,3,4];
K = 2;

Output:

Here
Distance between (1,2) and origin is sqrt(5)
Distance between (2,3) and origin is sqrt(13)
Distance between (3,4) and origin is sqrt(25)
As sqrt(5)<sqrt(13)<sqrt(25) so the closest points will be
(1,2),(2,3),(3,4).As we need closest 2 points the answer will be (1,2) and (2,3)

Intuition:
We can calculate the euclidean distance for every point from origin,for finding the k nearest points somehow we need to store this distance.If we store this in array and then we need to find the kth element in this array which can be found out if we sort the array.After sorting the array the element at kth index will be our required element.Now all the points having distance less than equal to k will belong to answer group.

Approach:
The approach behind this problem is find the euclidean distance of each point from the origin and store it in a separate array and sort the distance array and return the first k points corresponding to the distance array

Algorithm:
1)Create a distance array and store the euclidean distance of all points from origin.Sort the array .
2)Select the first k points and return the corresponding points for this distances.

Implementation

Code:https://jsfiddle.net/zx8qmbup/1/

Output:[[1,3],[3,1]]

```
//input

x = [1,2,3,4];
y = [3,4,1,0];
k = 2;


function k_closest(x,y,k){
  let output = [];
  let n = x.length;
  let dist = new Array(n);
  for(let i=0;i<n;i++){
    dist[i] = x[i]*x[i]+y[i]*y[i];
  }
  dist.sort((a,b)=>a-b);
  //finding the k_th distance;
  let k_dist = dist[k-1];
  for(let i=0;i<n;i++){
    let temp = x[i]*x[i]+y[i]*y[i];
    if(temp<=k_dist){
      output.push([x[i],y[i]]);
    }
  }
  return output;
}
console.log(k_closest(x,y,k));
```

Complexity Analysis:

Time Complexity:

For sorting the array time complexity is O(nlogn) and for iterating the array O(n).So overall complexity is O(nlogn)

Space Complexity:

As we are storing the distance in a newly created array ,so space complexity is O(n)

Sort array in wave form

Given an array which is unsorted ,we need to sort the array into wave like array where arr[0]>arr[1]<=arr[2]>=arr[3]<=arr[4]>=arr[5].....

Input:{9,8,7,6,5,21,2,4}
Output:{4,2,6,5,8,7,21,9}

Intuition:
Here,we can see we need to somehow make the element at odd indexes lesser than both the previous even and the next even index element
and element at even index should be greater than both prev and next odd index element.

Approach:
This problem can be solved easily if we sort the array.So after sorting input array becomes
{2,4,5,6,7,8,9,21} and now if we swap the adjacent elements the array has the wave like structure
{4,2,6,5,8,7,21,9}

Algorithm:
1)Sort the array
2)Swap the adjacent elements and return the array

Implementation:

Code:https://jsfiddle.net/zx8qmbup/2/

Output:[4, 2, 6, 5, 8, 7, 21, 9]

```
var input = [9,8,7,6,5,21,2,4];


function sortwave(input){
  input.sort((a,b)=>a-b);
  let n = input.length;
  if(n>1){
    for(let i=1;i<n;i+=2){
      let temp = input[i];
      input[i] = input[i-1];
      input[i-1] = temp;
    }
    return input;
  }
  else
  return input;
}
console.log(sortwave(input));
```

Complexity Analysis:

Time Complexity:
For sorting the array time complexity is O(nlogn) and for iterating over the array O(n).So overall time complexity is O(nlogn)

Space complexity:
No extra space is used:O(1)

Approach 2:
This problem can be solved in O(n) in time complexity if we just maintain the order at odd and even indexes.

Algorithm:
1)Traverse the array from start to second last element
2)if (index%2==0),{
if(arr[index]<arr[index+1]),then swap this elements as we need element at even index greater than odd index
if(arr[index]>=arr[index+1]),then continue as the element at even index is already greater
}
3)if(index%2==1){
if(arr[index]>arr[index+1]),then swap as we want the element at odd index be smaller than even index element
if(arr[index]<=arr[index+1]),then continue as the element at odd index is smaller than even index
}

Implementation

Code:https://jsfiddle.net/zx8qmbup/

```
var input = [9,8,7,6,5,21,2,4];

function wavesort(inp){
  let n = inp.length-1;
  let temp = 0;
  for(let i=0;i<n;i++){
    if(i%2==0){
      if(inp[i]<inp[i+1]){
        temp = inp[i];
        inp[i] = inp[i+1];
        inp[i+1] = temp;
      }
    }else{
      if(inp[i]>inp[i+1]){
        temp = inp[i];
        inp[i] = inp[i+1];
        inp[i+1] = temp;
      }
    }
  }
  return inp;
}

console.log(wavesort(input));
```

Complexity Analysis:

Time Complexity:
As we are traversing the array once so the time complexity is O(n)

Space Complexity:
O(1) as no extra space is used

Meeting Rooms

Given two arrays S and F where elements in S represents the starting time of the meeting and elements in F represent the ending time of meetings.We need to find maximum number of meetings that can be accomodated in a meeting room

Input:S:{1,3,0,6,8,4}
F:{2,4,6,7,9,9}
Output:4

Intuition: Here we need to choose maximum number of meetings without any violation.If we sort this interval on the basis of end time we would have input something like this

[{1,2},{3,4},{0,6},{6,7},{8,9},{4,9}]

Choosing end time for sorting so that we get the smallest end times earlier as it would help us to choose maximum intervals.Now we need to select intervals and for selection of intervals ,previous intervals ending time should be less than new intervals starting time.If this condition is met we increase the count of interval.

Approach: As we want maximum number of meetings to happen we need to add as many number of meetings whose ending time are less We create a array storing {si,fi} as element of array.We then sort this array on the basis of finishing time(fi) as we want to select maximum number of meetings so the elements having less finish time will be of advantage and set finish limit as finishing time of first meeting.Now iterate over the start array from index one and check if start time greater than finish limit update the meeting counter by one and finish limit by finish timing of current meeting

Algorithm:

1)Create a array containing both start and finish time

2)Sort this array in increasing order of finish time of each pair

3)Create a counter for keeping track of meeting and limit for finding the ending time of previous meeting

4)Increment the counter by one as we are keeping the first element of the sorted array and set limit to ending time of first meeting

5)Traverse from second element to last element of array and if the starting time is greater than limit increment counter by one and update limit to ending time of current element

Implementation:

Code:https://jsfiddle.net/zx8qmbup/3/

```
var start = [1,3,0,6,8,4];
var finish = [2,4,6,7,9,9]


function maxMeeting(s,f){
  var pair = [];
  let n = start.length;
  for(let i=0;i<n;i++){
    pair.push([s[i],f[i]])
  };
  pair.sort((a,b)=>a[1]-b[1]);
  console.log(pair);
  let counter = 1;//keeping the first element as it has the minimum en
ding time
  let limit = pair[0][1];//setting limit to first meetings ending time

  for(let i=1;i<n;i++){
    if(pair[i][0]>=limit){
      counter++;
      limit = pair[i][1];
    }
  }
  return counter;
}
console.log(maxMeeting(start,finish));
```

Complexity Analysis:

Time Complexity:
Sorting the array takens O(nlogn) and iterating over the array takes O(n) time.So overall complexity is O(nlogn)

Space Complexity:
As we are creating an array for sorting purpose ,space complexity is O(n)

Max Chunks To Make Array Sorted

Given an array of length n having elements in the range [0,n-1] in any order.We have to split the array into maximum number of chunks and sort each chunk and concatenate them which should return a complete sorted array.Find the maximum number of chunks that can be made

Example
Input:[1,0,4,3,2]
Output:2

If we split the array into individual elements the chunk size would be maximum but the array will not be sorted but if we divide into two chunks [1,0] and [4,3,2] and sort and concatenate it we get [0,1,2,3,4].

Intuition: The idea is to find the maximum number of splitting lines so that elements at the left side of this line are smaller than the elements at the right side of the line.You consider [1,0,4,3,2],1 original position is at index 1 after sorting and its the biggest element in the range(0,1) and if we create a partition here,all the elements before the partition are smaller than elements after the partition.

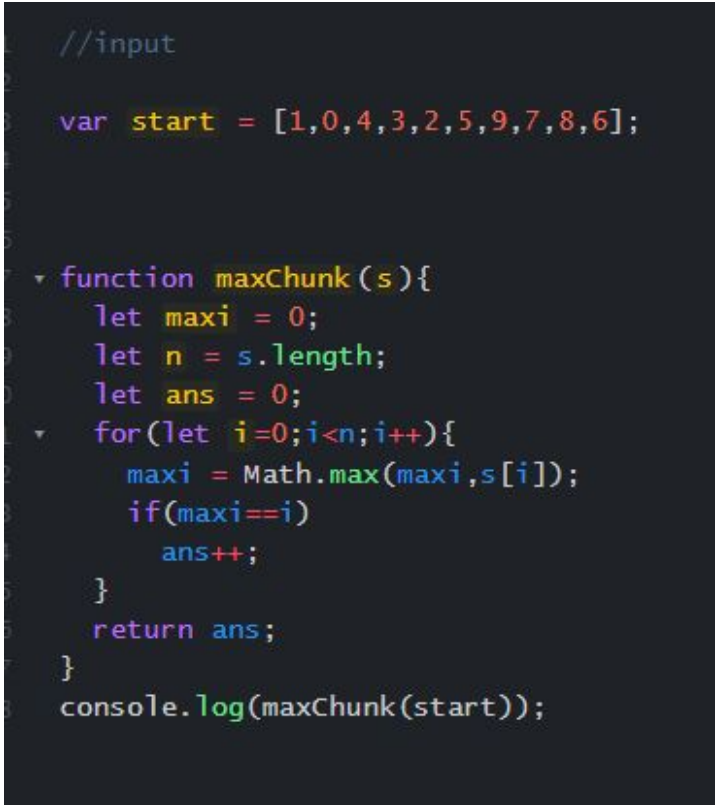
Approach: Here the idea is to check if an element at i is max of prefix input[0..i],we can create a partition ending with index i because at i,prefix(input[0..i]) == i and all elements before this point is smaller than or equal to i and all the elements after this are greater than index i.

Lets take an example and understand this
prefixmax = 0;
[1,0,4,3,2]
At index 0,prefixmax = max(prefixmax,arr[0])=1
At index 1,prefixmax = max(1,arr[1])=1
Here prefixmax == index,at this point all elements before this index are smaller than equal to index and after this point is greater than index,so we create a partition here
At index 2,prefixmax = max(1,arr[2])=4
At index 3,prefixmax = max(4,arr[3])=4
At index 4,prefixmax = 4
Here also prefixmax==index,now between last partition to this index,all the elements are smaller than or equal to index,so a partition can be created here

Algorithm:
1)Create a variable to store max in prefix input[0,i]
2)If max is equal to index create a partition

Implementation:

Output:4
Code:https://jsfiddle.net/zx8qmbup/4/



```
//input

var start = [1,0,4,3,2,5,9,7,8,6];



function maxChunk(s){
  let maxi = 0;
  let n = s.length;
  let ans = 0;
  for(let i=0;i<n;i++){
    maxi = Math.max(maxi,s[i]);
    if(maxi==i)
      ans++;
  }
  return ans;
}
console.log(maxChunk(start));
```

Complexity Analysis:

Time Complxity:

As we are iterating over the array once so the time complexity is O(n)

Space Complexity:

O(1) as no extra space is used

Maximum Gap

Given an array of integers we need to find the maximum difference between two consecutive elements when the array is sorted in ascending order

Example:
Input:[2,4,7,9,1,15]
Output:6
If the array is sorted it would have elements as
[1,2,4,7,9,15] and the maximum difference is (15-9) which is 6.

Approach 1:
The simplest approach for this problems is to sort the array and find the difference between two consecutive elements of the array and update the maximum storing variable

Algorithm:
1)Sort the array in ascending order
2)Initialize a variable maxi to store maximum difference between two consecutive elements
3)Iterate over the array and update maxi to difference between two elements if its greater than maxi value

Implementation:

Output:6
Code:https://jsfiddle.net/edpkc9mx/1/

```
var input = [2,4,7,9,1,15];

function maxdiff(input){
  input.sort((a,b)=>a-b);
  let maxi = 0;
  for(let i=1;i<input.length;i++){
    let diff = input[i]-input[i-1];
    maxi = diff>maxi?diff:maxi;
  }
  return maxi;
}

console.log(maxdiff(input));
```

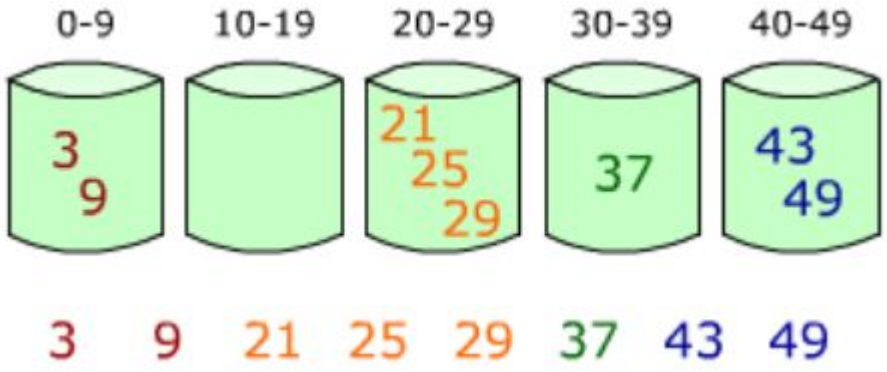
Complexity Analysis:

Time Complexity:
For sorting the array time complexity is O(nlogn) and for iterating over the array time complexity is O(n).So overall complexity is O(nlogn)

Space Complexity:
As constant space is used,so space complexity is O(1)

Intuition:

The main idea behind this approach is to separate the elements in some buckets and we try to find the maximum gap by finding difference between min in one bucket and max in another bucket.



Now the main problem is how we define the bucket size.We have to define the bucket size such that maximum gap will be surely larger than a single bucket.For achieving a right bucket size,we iterate through the array to find the total range

(max(arr)-min(arr))

and use this figure for finding the

smallest gap value(max(arr)-min(arr))/(arr.length-1).

If we make sure that bucket size is smaller than this gap value,then the numbers forming the maximum gap will have to be in separate buckets.If there are two numbers,there will be one gap,if there are three numbers there will be two gaps.For sorted array of n elements there will be n-1 gaps.So if we take n-1 gaps and evenly spread the total distance between them we get the gap size to be (max(arr)-min(arr))/(arr.length-1).
If any gap becomes smaller than this number,then some other gap must have become larger to make up for the difference in total gap

Approach 2:

If there are N elements in the array, we find the the min and max value in the array.Then the maximum gap >= max(arr)-min(arr)/N-1

We take the gap = ceiling[(max - min) / (N - 1)]. We divide elements in the array into n-1 buckets, where the elements in buckets are [min,min+gap],[min+gap,mn+2*gap].....

So the kth bucket would contains elements [min + (k-1)gap, min + k*gap).

We have N-2 numbers which are not equal min or max and we have n-1 buckets, at least one of the buckets will be empty. We will be creating two arrays interval_min,interval_max to store the min and max value in each bucket and the max gap can be found by iterating over the array using interval_min[i+1] -interval_max[i].The main logic lies in finding the number of buckets.

Algorithm:

1. Find the max and min element in array
2. Find the value of interval length(gap) = max(arr)-min(arr)/arr.length-1
3. Initialize two new array of size arr.length-1 for storing max and min in each interval
4. Iterate over the array and put max and min value for each interval in the newly created arrays
5. Initialize a variable previous with min(arr)
6. Initialize a variable maxGap = 0;
7. Run a loop from 0 to N-1 to find the difference between min of next bucket and max of current bucket.

Implementation:

Output:6

Code:https://jsfiddle.net/edpkc9mx/2/

```
var input = [2,4,7,9,1,15];

function maxdiff(input){
  let min = input[0];
  let max = input[0];
  for(let i=0;i<input.length;i++){
    min = Math.min(input[i],min)
    max = Math.max(input[i],max);
  }
  let n = input.length;
  let gap = Math.ceil((max-min)/(n-1));
  let min_interval = new Array(n-1).fill(Number.MAX_SAFE_INTEGER);
  let max_interval = new Array(n-1).fill(Number.MIN_SAFE_INTEGER);
  for(let i=0;i<n;i++){
    if(input[i]==min||input[i]==max)
      continue;
    let idx = Math.ceil((input[i]-min)/gap);
    min_interval[idx] = Math.min(input[i],min_interval[idx]);
    max_interval[idx] = Math.max(input[i],max_interval[idx]);
  }
  let maxGap = 0;
  let previous = min;
  for(let i=0;i<n-1;i++){
    if(min_interval[i]===Number.MAX_SAFE_INTEGER && max_interval[i]===Number.MIN_SAFE_INTEGER)
      continue;
    console.log(min_interval[i]+" "+max_interval[i]);
    maxGap = Math.max(maxGap,min_interval[i]-previous);
    previous = max_interval[i];
  }
  maxGap = Math.max(maxGap,max-previous);
  return maxGap;
}

console.log(maxdiff(input));
```

Complexity Analysis:

Time Complexity:
As we iterating over the array constant times so the time complexity will be O(n)

Space Complexity:
As we have create two separate array for storing the max and min in each bucket so the space complexity will be O(n)

Minimum Number of Moves to Seat Everyone

Given there are N seats and N students in a room.An array seat of length is given where seat[i] is the position of ith seat and also an array of students of length n is given where stud[i] is the position of ith student.
We can perform the following move any number of times.
Increase or decrease the position of ith student(moving position of ith student from position j to j+1 or j-1)
We need to find the minimum number of moves required to move each student to a seat so that no two students share same seat.

Example: seat = [2,1,5]
stud = [2,7,4]
Output:5

Intuition:
We have to arrange in a way where the student having the smallest positioned chair will acquire the smallest position seat and the next student will acquire the next smallest position seat

Approach:
We can sort the students and seat array in ascending order and will find the minimum move the student has to move in order to occupy the positioned seat.

Algorithm:
1)Sort the two array
2)Initialize a variable counter to store the min moves made by each student to occupy the seat
3)Iterate over the array and increase the value of
counter by abs(seat[i]-stud[i]) for each index.

Implementation

Code:https://jsfiddle.net/k7v91hxm/1/
Output:5

```
let seat = [2,1,5];
let stud = [2,7,4];


function arrange(seat,stud){
  seat.sort((a,b)=>a-b);
  stud.sort((a,b)=>a-b);
  let counter = 0;
  for(let i=0;i<seat.length;i++){
    counter+=Math.abs(seat[i]-stud[i]);
  }
  return counter;
}

console.log(arrange(seat,stud));
```

Complexity Analysis:

Time Complexity:
O(nlogn) as we are sorting the seat and stud array

Space complexity:
Additional space is required for sorting how much space it depends on the type of sorting

Minimum Increment to Make Array Unique

Given an array of integers.We need to find the minimum number of moves where move is you can choose any element of array and increment it by 1 to make every element in the array unique

Input:[1,3,3]

Output:After incrementing one of the 3 to four ,all the elements in the array will be unique,so output is 1

Intuition:

We need to make all the duplicate elements present in array to be unique,we need to increment those values.

Approach:

Lets take an example:

[1,1,1,4,6]:Here we will not be processing all the increments of extra 1's.We will store it and continue processing until we find a value which is not present in array like 2,3 from which we get the increments to be 2-1,3-1.

Algorithm:

1)Store the count of every element of array

2)We will iterate from 0 to largest value in array +array.length and for each value of x

a)If the count of x>=2,save the duplicates to increment later

b)If there are 0 values of x,then last save duplicate value gets incremented to x.

Implementation

Output:6
Code:https://jsfiddle.net/k7v91hxm/2/

```
let nums = [3,2,1,2,1,7]

function makeUnique(nums){
let maxVal = 0;
let map1 = {};
for(let i=0;i<nums.length;i++){
  maxVal = Math.max(maxVal,nums[i]);
  if(!map1[nums[i]]){
    map1[nums[i]]=0;
  }
  map1[nums[i]]++;
}
console.log(map1);
let move = 0;
let taken = 0;
for(let x=0;x<nums.length+maxVal;x++){
  if(map1[x]>=2){
    taken+=(map1[x]-1);
    move -=x*(map1[x]-1);
  }else if(taken>0 && !map1[x]){
      taken--;
      move+=x;
  }
}
return move;
}
console.log(makeUnique(nums));
```

Complexity Analysis:

Time Complexity:

Let N be the size of array and M be the maximum value in array.We are iterating from 0 to N+M so the time complexity is O(N+M)

Space Complexity:

As we are using object for storing the count of each element in array so the space complexity is O(N)

Practice Questions

1. Given a list of time intervals where time intervals[i] = [starti, endi],we need to find the least number of time intervals to be removed in order to make the list of time intervals non overlapping
2. Given an array of integers with length n and we need to find element that occurs more than n/2 times in the array

Thank you!