Min Steps in Infinite Grid

Asked in: Directi

Suggest Edit | Bookmark

You are in an infinite 2D grid where you can move in any of the 8 directions :

```
(x,y) to
   (x+1, y),
   (x - 1, y),
   (x, y+1),
   (x, y-1),
   (x-1, y-1),
   (x+1,y+1),
   (x-1,y+1),
   (x+1,y-1)
```

You are given a sequence of points and **the order in which you need to cover the points**. Give the minimum number of steps in which you can achieve it. You start from the first point.

```
public class Solution {
    public int coverPoints(ArrayList<Integer> A, ArrayList<Integer> B) {
        int x=A.get(0);
        int y=B.get(0);
        int l=0;
        int s1,s2;
        s1=s2=0;
        for(int i=1;i<A.size();i++){
            int m=A.get(i);
            int n=B.get(i);
            s1=Math.abs(m-x);
            s2=Math.abs(n-y);
            l+=Math.max(s1,s2);
            x=m;
            y=n;
        }
        return l;
    }
}
```

Add One To Number

Asked in: Google

Suggest Edit | Bookmark

Given a non-negative number represented as an array of digits,

add 1 to the number (increment the number represented by the digits).

The digits are stored such that the most significant digit is at the head of the list.

Example:

If the vector has `[1, 2, 3]`

the returned vector should be `[1, 2, 4]`

as `123 + 1 = 124` .

```
public class Solution {
    public ArrayList<Integer> plusOne(ArrayList<Integer> A) {
        ArrayList<Integer> res=new ArrayList<Integer>(A);
        int c=1;
        for(int i=A.size()-1;i>=0;i--){
            if(c==0){
                break;
            }
            else{
                int x=A.get(i)+1;
                A.set(i,x%10);
                c=x/10;
            }
        }
        if(c==1){
            A.add(0,1);
        }
        int index=0;
        while(index<A.size() && A.get(index)==0){
            A.remove(index);
        }
        return A;
    }
}
```

Max Sum Contiguous Subarray

Asked in: Facebook, Paypal, Yahoo, Microsoft, LinkedIn, Amazon, Goldman Sachs

Bookmark

Suggest Edit

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example:

Given the array `[-2,1,-3,4,-1,2,1,-5,4]`,

the contiguous subarray `[4,-1,2,1]` has the largest sum = 6.

For this problem, return the maximum sum.

See Expected Output

Seen this question in a real interview before Yes No

```
public class Solution {
    // DO NOT MODIFY THE LIST. IT IS READ ONLY
    public int maxSubArray(final List<Integer> A) {
        int sum=0;
        int max=A.get(0);
        for(int i=1;i<A.size();i++){
            sum+=A.get(i);
            if(sum>max){
                max=sum;
            }
            if(sum<0){
                sum=0;
            }
        }
        return max;
    }
}
```

Largest Number

Asked in: Amazon Goldman Sachs

Suggest Edit | Bookmark

Given a list of non negative integers, arrange them such that they form the largest number.

For example:

Given `[3, 30, 34, 5, 9]`, the largest formed number is `9534330`.

Note: The result may be very large, so you need to return a string instead of an integer.

See Expected Output

Seen this question in a real interview before Yes No

```
public class Solution {
    // DO NOT MODIFY THE LIST. IT IS READ ONLY
    public String largestNumber(final List<Integer> A) {
        String[] a=new String[A.size()];
        for(int i=0;i<A.size();i++){
            a[i]=String.valueOf(A.get(i));
        }
        Arrays.sort(a,new Comparator<String>(){
            public int compare(String a,String b){
                return (b+a).compareTo(a+b);
            }
        });


        StringBuilder sb=new StringBuilder();
        for(String s:a){
            sb.append(s);
        }

        while(sb.charAt(0)=='0' && sb.length()>1){
            sb.deleteCharAt(0);
        }

        return sb.toString();
    }
}
```

Next Permutation

Asked in: Microsoft Amazon

Suggest Edit | Bookmark

Implement the next permutation, which rearranges numbers into the numerically next greater permutation of numbers.

If such arrangement is not possible, it must be rearranged as the lowest possible order *ie, sorted in an ascending order.*

The replacement must be in-place, do not allocate extra memory.

Examples:

```
1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1

20, 50, 113 → 20, 113, 50
```

My idea is for an array:

1.Start from its last element, traverse backward to find the first one with index i that satisfy num[i-1] < num[i]. So, elements from num[i] to num[n-1] is reversely sorted.

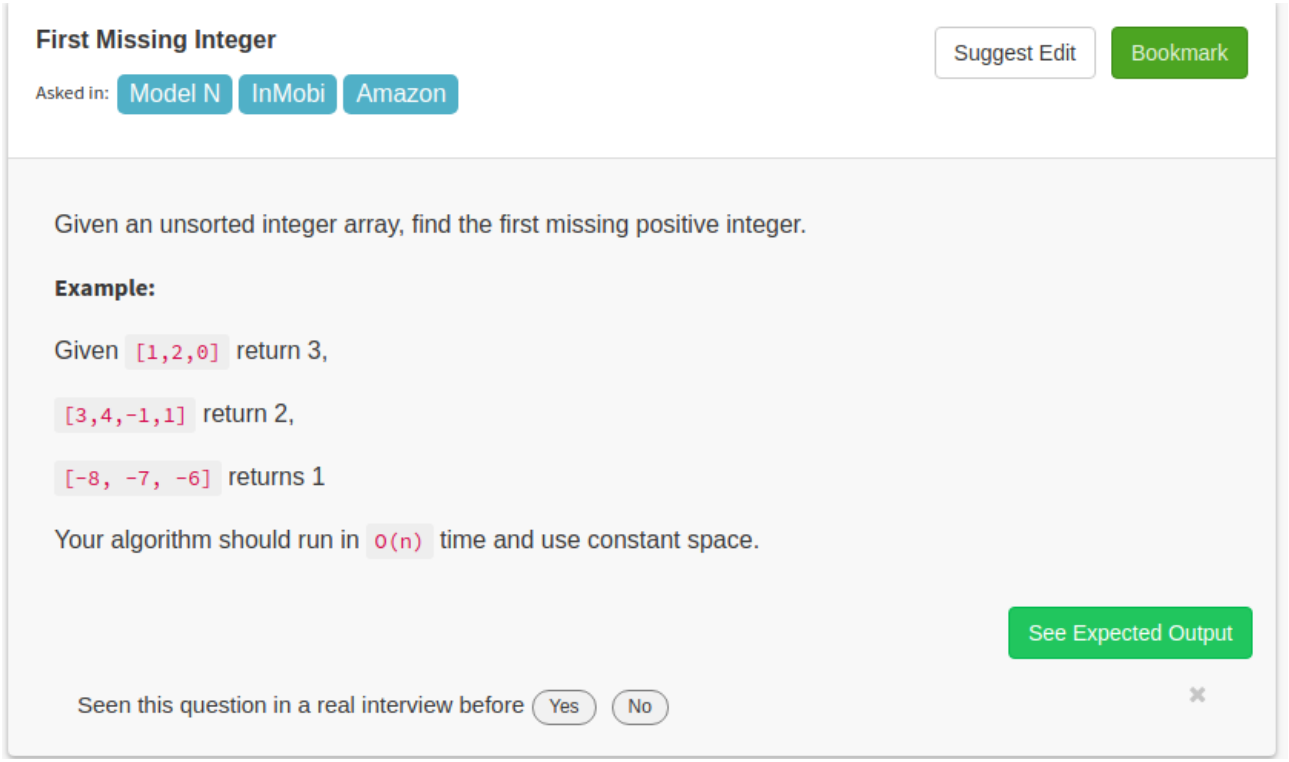
2.To find the next permutation, we have to swap some numbers at different positions, to minimize the increased amount, we have to make the highest changed position as high as possible. Notice that index larger than or equal to i is not possible as num[i,n-1] is reversely sorted. So, we want to increase the number at index i-1, clearly, swap it with the smallest number between num[i,n-1] that is larger than num[i-1]. For example, original number is 121543321, we want to swap the '1' at position 2 with '2' at position 7.

3.The last step is to make the remaining higher position part as small as possible, we just have to reversely sort the num[i,n-1]

```
public class Solution {
        public void nextPermutation(ArrayList<Integer> num) {
            int n=num.size();
    if(n<2)
        return;
    int index=n-1;
    while(index>0){
```

```
        if(num.get(index-1)<num.get(index))
            break;
        index--;
    }
    if(index==0){
        reverseSort(num,0,n-1);
        return;
    }
    else{
        int val=num.get(index-1);
        int j=n-1;
        while(j>=index){
            if(num.get(j)>val)
                break;
            j--;
        }
        swap(num,j,index-1);
        reverseSort(num,index,n-1);

        return;
    }
}

public void swap(ArrayList<Integer> num, int i, int j){
    int temp=0;
    temp=num.get(i);
    num.set(i,num.get(j));
    num.set(j,temp);
}

public void reverseSort(ArrayList<Integer> num, int start, int end){
```

```
    if(start>end)
        return;
    for(int i=start;i<=(end+start)/2;i++)
        swap(num,i,start+end-i);
}
}
```




```
public class Solution {
    public int firstMissingPositive(ArrayList<Integer> A){
      boolean[] a=new boolean[A.size()+1];
      int n=0;
      for(int i=0;i<A.size();i++){
          if(A.get(i)>0 && A.get(i)<=A.size()){
              n++;
              a[A.get(i)]=true;
          }
      }
```

```
        if(n==0){
         return 1;
      }
      for(int i=1;i<=n;i++){
         if(!a[i]){
            return i;
         }
      }
     return n+1;
    }
}
```

Repeat Missing Number:

You are given a read only array of n integers from 1 to n.

Each integer appears exactly once except A which appears twice and B which is missing.

Return A and B.

Note: Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

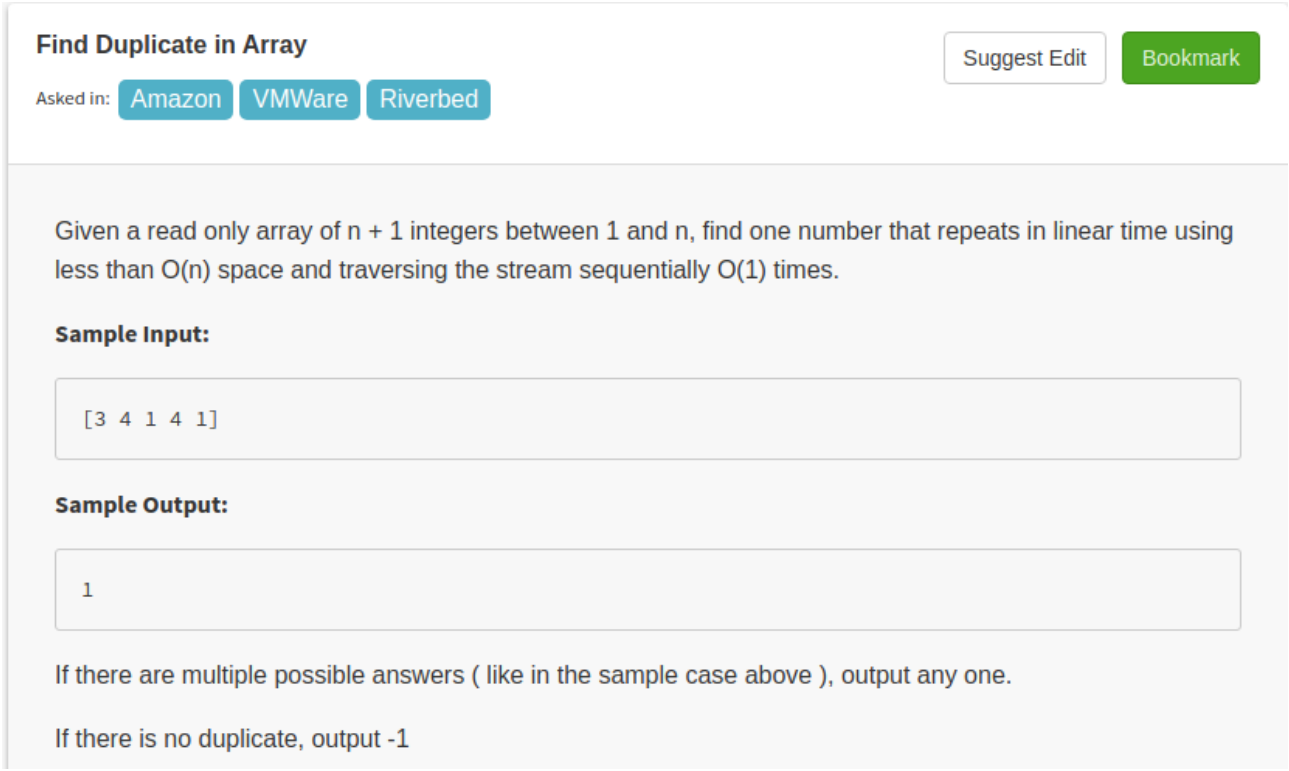
Note that in your output A should precede B.

Example:

```
Input:[3 1 2 5 3]

Output:[3, 4]

A = 3, B = 4
```

```
public class Solution {
    public ArrayList<Integer> repeatedNumber(final List<Integer> A) {
        ArrayList<Integer> result=new ArrayList<Integer>();
        int x=1;
        boolean a[]=new boolean[A.size()+1];
        for(int i=0;i<A.size();i++){
            if(a[A.get(i)]){
```

```
                result.add(A.get(i));
            }
            a[A.get(i)]=true;
        }
        for(int i=1;i<A.size()+1;i++){
            if(!a[i]){
                result.add(i);
                break;
            }
        }
        return result;
    }
}
```



Find Duplicate in Array

Asked in: Amazon VMWare Riverbed

Suggest Edit | Bookmark

Given a read only array of n + 1 integers between 1 and n, find one number that repeats in linear time using less than O(n) space and traversing the stream sequentially O(1) times.

Sample Input:

```
[3 4 1 4 1]
```

Sample Output:

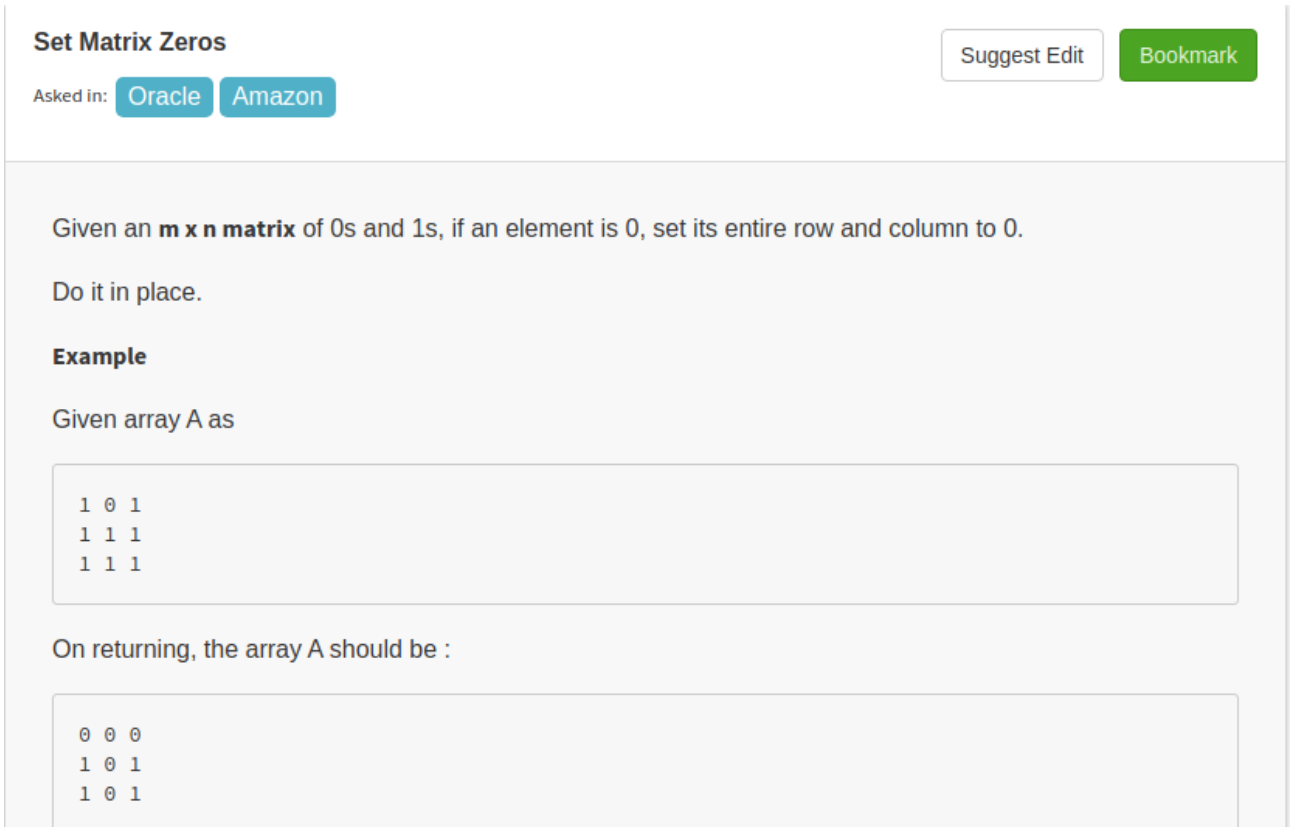
```
1
```

If there are multiple possible answers (like in the sample case above), output any one.

If there is no duplicate, output -1

```
public class Solution {
    // DO NOT MODIFY THE LIST
    public int repeatedNumber(final List<Integer> A) {
        int temp[]=new int[A.size()];
```

```
        for(int i=0;i<A.size();i++){
            if(temp[A.get(i)]==1){
                return A.get(i);
            }
            else
            {
                temp[A.get(i)]=1;
            }
        }
        return -1;
    }
}
```



Set Matrix Zeros

Asked in: Oracle Amazon

Suggest Edit | Bookmark

Given an **m x n matrix** of 0s and 1s, if an element is 0, set its entire row and column to 0.

Do it in place.

Example

Given array A as

```
1 0 1
1 1 1
1 1 1
```

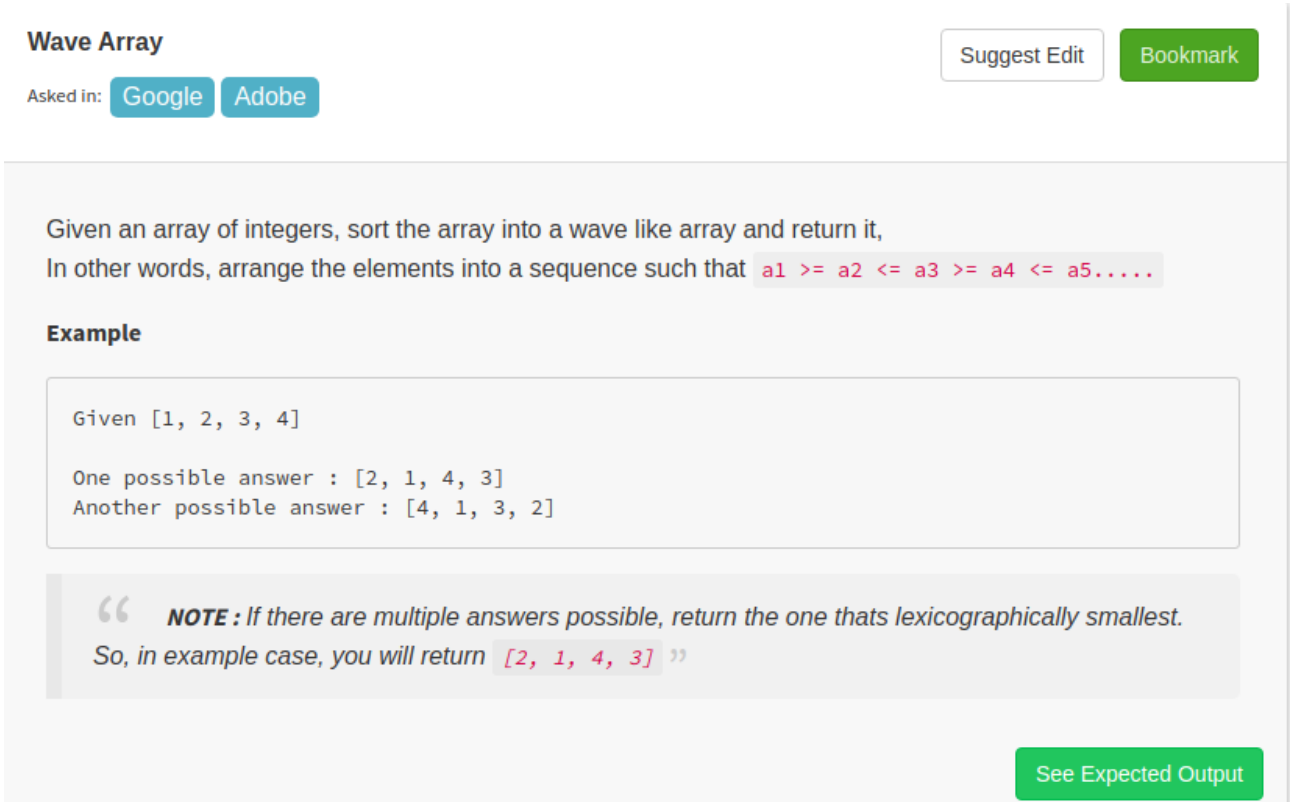
On returning, the array A should be :

```
0 0 0
1 0 1
1 0 1
```

```
public class Solution {
    public void setZeroes(ArrayList<ArrayList<Integer>> a) {
            int m=a.size();
            int n=a.get(0).size();
```

```
boolean fr[]=new boolean[m];
boolean fc[]=new boolean[n];
for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
        if(a.get(i).get(j)==0){
            fr[i]=true;
            fc[j]=true;
        }
    }
}
for(int i=0;i<m;i++){
    if(fr[i]){
        nullRow(a,i);
    }
}

for(int i=0;i<n;i++){
    if(fc[i]){
        nullCol(a,i);
    }
}



}
void nullRow(ArrayList<ArrayList<Integer>> a,int r){
    for(int i=0;i<a.get(0).size();i++){
        a.get(r).set(i,0);
    }
}
```

```
    void nullCol(ArrayList<ArrayList<Integer>> a,int c){
        for(int i=0;i<a.size();i++){
            a.get(i).set(c,0);
        }
    }
}
```



Wave Array

Asked in: Google Adobe

Suggest Edit | Bookmark

Given an array of integers, sort the array into a wave like array and return it,
In other words, arrange the elements into a sequence such that `a1 >= a2 <= a3 >= a4 <= a5.....`

Example

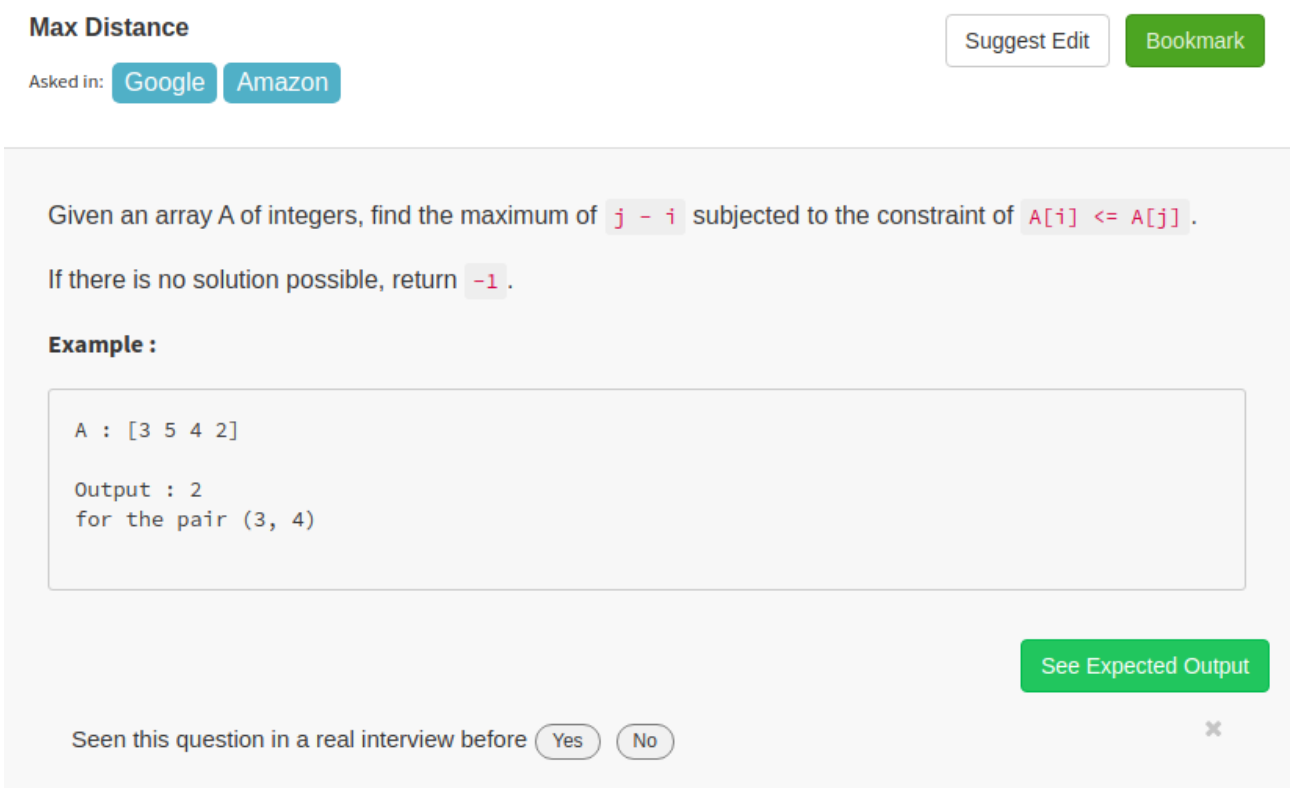
```
Given [1, 2, 3, 4]

One possible answer : [2, 1, 4, 3]
Another possible answer : [4, 1, 3, 2]
```

> ***NOTE :*** *If there are multiple answers possible, return the one thats lexicographically smallest.*
> *So, in example case, you will return* `[2, 1, 4, 3]`

See Expected Output

```
public class Solution {
    public ArrayList<Integer> wave(ArrayList<Integer> A) {
        Collections.sort(A);
        int i=0;
        while(i<=A.size()-2){
            int temp=A.get(i);
            A.set(i,A.get(i+1));
            A.set(i+1,temp);
            i+=2;
        }
        return A;
```

```
    }
}
```




```
public class Solution {
    // DO NOT MODIFY THE LIST. IT IS READ ONLY
    public int maximumGap(final List<Integer> A) {
        int n=A.size();
        int[] a=new int[n];
        int[] b=new int[n];
        a[0]=A.get(0);
        for(int i=1;i<n;i++){
            a[i]=Math.min(a[i-1],A.get(i));
        }
        b[n-1]=A.get(n-1);
        for(int i=n-2;i>=0;i--){
            b[i]=Math.max(b[i+1],A.get(i));
        }
```

```
    int i,j;
    i=j=0;
    int max=-1;
    while(i<n && j<n){
       if(a[i]<=b[j]){
          max=Math.max(max,j-i);
          j++;
       }
       else{
          i++;
       }
    }
    return max;
  }
}
```

Maximum Consecutive Gap

Asked in: Hunan Asset

Suggest Edit | Bookmark

Given an unsorted array, find the maximum difference between the successive elements in its sorted form.

Try to solve it in linear time/space.

Example :

```
Input : [1, 10, 5]
Output : 5
```

Return 0 if the array contains less than 2 elements.

1. You may assume that all the elements in the array are non-negative integers and fit in the 32-bit signed integer range.
2. You may also assume that the difference will not overflow.

See Expected Output

Seen this question in a real interview before Yes No

```
public class Solution {
    // DO NOT MODIFY THE LIST. IT IS READ ONLY
    public int maximumGap(final List<Integer> A) {
        int n=A.size();
        if(n<2){
            return 0;
        }
        Set<Integer> hs=new TreeSet<Integer>();
        for(int i=0;i<n;i++){
            hs.add(A.get(i));
        }

        int max=Integer.MIN_VALUE;
        int x=hs.iterator().next();
        for(Integer i:hs){
          if(i-x>max){
              max=i-x;
          }
          x=i;

        }
        return max;
    }
}
```

Merge Intervals

Asked in: Google

Suggest Edit | Bookmark

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

Example 1:

Given intervals `[1,3],[6,9]` insert and merge `[2,5]` would result in `[1,5],[6,9]`.

Example 2:

Given `[1,2],[3,5],[6,7],[8,10],[12,16]`, insert and merge `[4,9]` would result in `[1,2],[3,10],[12,16]`.

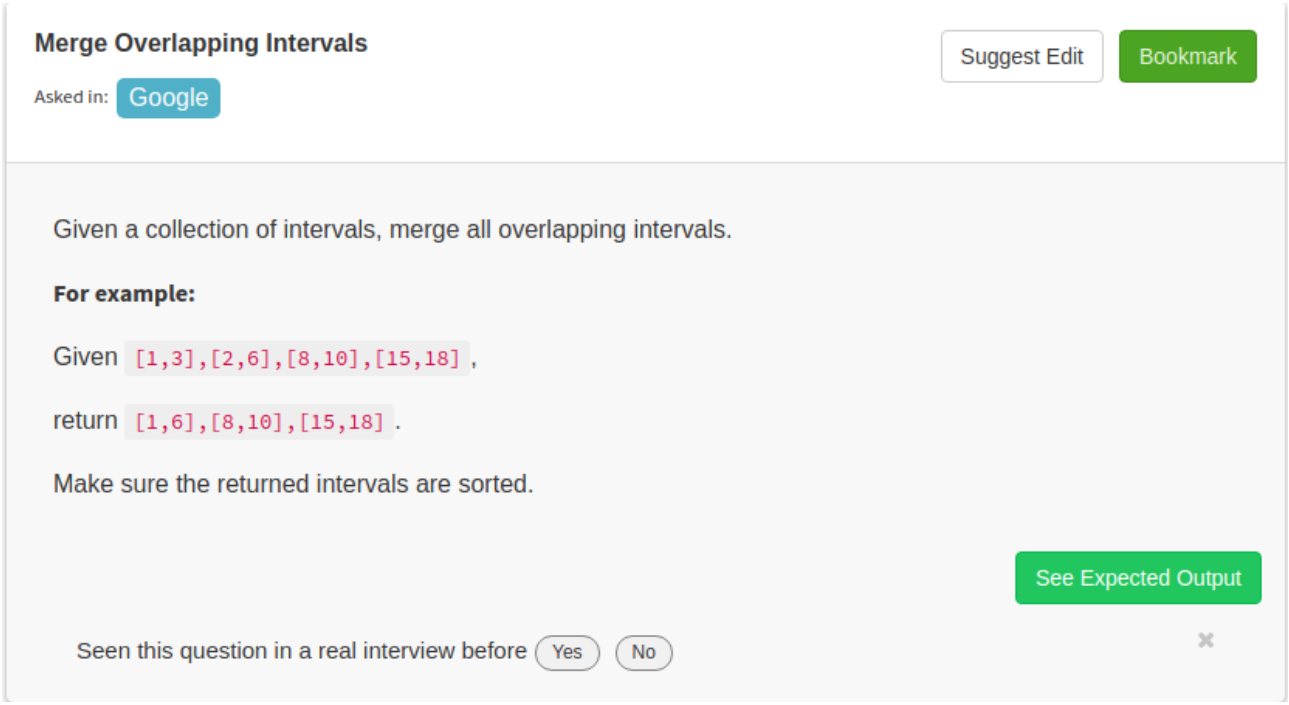
This is because the new interval `[4,9]` overlaps with `[3,5],[6,7],[8,10]`.

Make sure the returned intervals are also sorted.

See Expected Output

```
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public ArrayList<Interval> insert(ArrayList<Interval> intervals, Interval newInterval) {
        intervals.add(newInterval);
        ArrayList<Interval> res=new ArrayList<Interval>();
        Collections.sort(intervals, new Comparator<Interval>(){
```

```
        public int compare(Interval a, Interval b){
            if(a.start!=b.start){
                return a.start-b.start;
            }
            else
            {
                return a.end-b.end;
            }
        }});
        Interval prev=intervals.get(0);
        for(int i=1;i<intervals.size();i++){
            Interval next=intervals.get(i);
            if(next.start>prev.end){
                res.add(prev);
                prev=next;
            }
            else
            {
                Interval merge=new Interval(prev.start,Math.max(prev.end, next.end));
                prev=merge;
            }
        }
        res.add(prev);
        return res;
    }
}
```




```
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public ArrayList<Interval> merge(ArrayList<Interval> v) {
        ArrayList<Interval> res=new ArrayList<Interval>();
        Collections.sort(v, new Comparator<Interval>(){
            public int compare(Interval a, Interval b)
            {
                if(a.start!=b.start){
                    return a.start-b.start;
                }
```

```
            else
            {
                return a.end-b.end;
            }
        }});

        Interval pre=v.get(0);
        for(int i=1;i<v.size();i++){
            Interval cur=v.get(i);
            if(cur.start>pre.end){
                res.add(pre);
                pre=cur;
            }
            else
            {
                Interval merge=new Interval(pre.start,Math.max(pre.end, cur.end));
                pre=merge;
            }
        }
        res.add(pre);
  return res;

    }
}
```

Hotel Bookings Possible

Asked in: Goldman Sachs

Suggest Edit | Bookmark

A hotel manager has to process `N` advance bookings of rooms for the next season. His hotel has `K` rooms. Bookings contain an arrival date and a departure date. He wants to find out whether there are enough rooms in the hotel to satisfy the demand. Write a program that solves this problem in time O(N log N) .

Input:

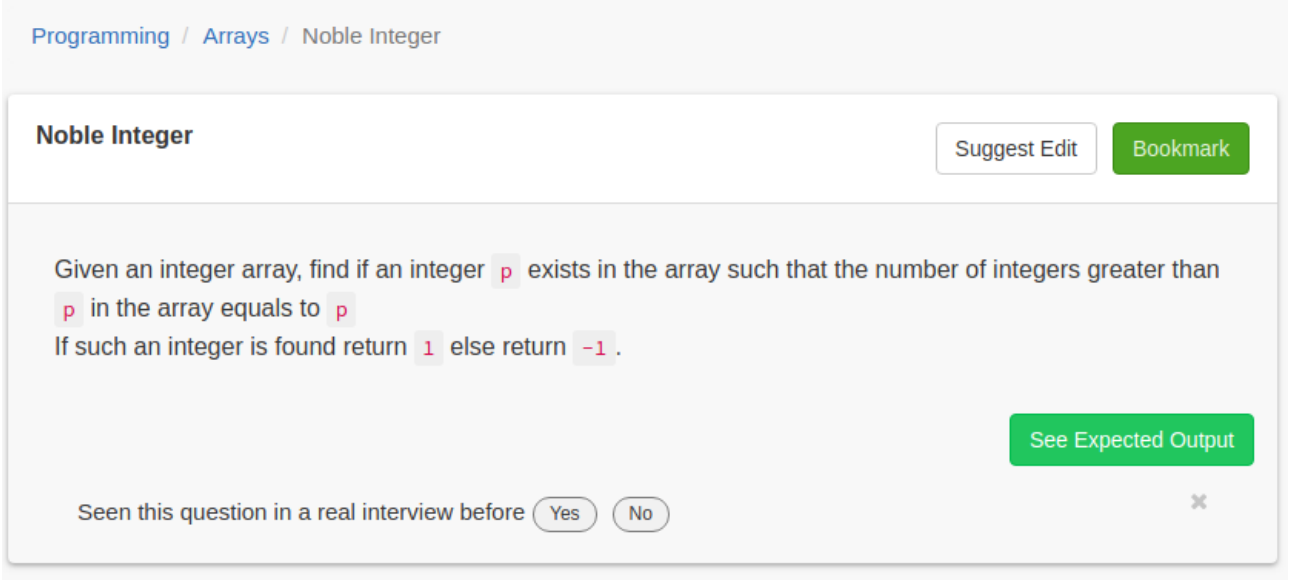
```
First list for arrival time of booking.
Second list for departure time of booking.
Third is K which denotes count of rooms.
```

Output:

```
A boolean which tells whether its possible to make a booking.
Return 0/1 for C programs.
0 -> No there are not enough rooms for N booking.
1 -> Yes there are enough rooms for N booking.
```

```
public class Solution {
    public boolean hotel(ArrayList<Integer> a, ArrayList<Integer> d, int K) {
        Collections.sort(a);
        Collections.sort(d);
        int temp,max,i=1,j=0;
        temp=max=1;
        int n=a.size();
        while(i<n && j<n){
            if(a.get(i)<d.get(j)){
                ++temp;
                if(temp>max){
                    max=temp;
                }
                i++;
            }
```

```
            else{
                --temp;
                j++;
            }
        }
        return K>=max;



    }
}
```




```
public class Solution {
    public int solve(ArrayList<Integer> A) {
        int n=A.size();
       Collections.sort(A);
       for(int i=0;i<n-1;i++){
           if(A.get(i)==A.get(i+1)){
               continue;
           }
```

```
        if(A.get(i)==n-i-1){
            return 1;
        }
    }
    if(A.get(n-1)==0){
        return 1;
    }
    return -1;
  }
}
```

Maximum Unsorted Subarray

Suggest Edit | Bookmark

You are given an array (zero indexed) of **N** non-negative integers, A_0, A_1 ,..., A_{N-1}.
Find the minimum sub array A_l, A_{l+1} ,..., A_r so if we sort(in ascending order) that sub array, then the whole array should get sorted.
If **A** is already sorted, output **-1**.

Example :

```
Input 1:

A = [1, 3, 2, 4, 5]

Return: [1, 2]

Input 2:

A = [1, 2, 3, 4, 5]

Return: [-1]
```

In the above example(Input 1), if we sort the subarray A_1, A_2, then whole array **A** should get sorted.

Solution:

1) Find the candidate unsorted subarray

a) Scan from left to right and find the first element which is greater than the next element. Let *s* be the index of such an element. In the above example 1, *s* is 3 (index of 30).

b) Scan from right to left and find the first element (first in right to left order) which is smaller than the next element (next in right to left order). Let *e* be the index of such an element. In the above example 1, e is 7 (index of 31).

2) Check whether sorting the candidate unsorted subarray makes the complete array sorted or not. If not, then include more elements in the subarray.

a) Find the minimum and maximum values in *arr[s..e]*. Let minimum and maximum values be *min* and *max*. *min* and *max* for [30, 25, 40, 32, 31] are 25 and 40 respectively.

b) Find the first element (if there is any) in *arr[0..s-1]* which is greater than *min*, change *s* to index of this element. There is no such element in above example 1.

c) Find the last element (if there is any) in *arr[e+1..n-1]* which is smaller than max, change *e* to index of this element. In the above example 1, e is changed to 8 (index of 35)

3) Print *s* and *e*.

```
public class Solution {
   public int[] subUnsort(int[] arr) {

      int n=arr.length;
      int s = 0, e = n-1, i, max, min;
      int[] res=new int[2];
      res[0]=-1;
      res[1]=-1;
      int[] nf={-1};

     // step 1(a) of above algo
     for (s = 0; s < n-1; s++)
     {
      if (arr[s] > arr[s+1])
        break;
     }
```

```
if (s == n-1)
{

 return nf;
}

// step 1(b) of above algo
for(e = n - 1; e > 0; e--)
{
 if(arr[e] < arr[e-1])
   break;
}

// step 2(a) of above algo
max = arr[s]; min = arr[s];
for(i = s + 1; i <= e; i++)
{
 if(arr[i] > max)
   max = arr[i];
 if(arr[i] < min)
   min = arr[i];
}

// step 2(b) of above algo
for( i = 0; i < s; i++)
{
 if(arr[i] > min)
 {
   s = i;
   break;
 }
```

```
    }

    // step 2(c) of above algo
    for( i = n -1; i >= e+1; i--)
    {
      if(arr[i] < max)
      {
        e = i;
        break;
      }
    }
    res[0]=s;
    res[1]=e;
    // step 3 of above algo
    return res;
  }
}
```

Search for a Range

Asked in: Google Microsoft

Suggest Edit | Bookmark

Given a sorted array of integers, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of `O(log n)`.

If the target is not found in the array, return `[-1, -1]`.

Example:

Given `[5, 7, 7, 8, 8, 10]`

and target value `8`,

return `[3, 4]`.

See Expected Output

Seen this question in a real interview before Yes No

```
public class Solution {
    // DO NOT MODIFY THE LIST
    public ArrayList<Integer> searchRange(final List<Integer> a, int b) {
        ArrayList<Integer> r=new ArrayList<Integer>();
        if(a.size()<=1){
            r.add(0);
            r.add(0);
            return r;
        }
        int x=findfirst(a,b);
        int y=findlast(a,b);
        r.add(x);
        r.add(y);
        return r;
    }
    public int findfirst(final List<Integer> a, int b){
        int index=-1;
        int start=0;
        int end=a.size()-1;
        int indx=-1;
        while(start<=end){
            int mid=start+(end-start)/2;
            if(a.get(mid)>=b){
                end=mid-1;
            }
            else
            {
                start=mid+1;
            }
            if(a.get(mid)==b){
                indx=mid;
```

```
        }
      }
      return indx;
    }
    public int findlast(final List<Integer> a, int b){
      int start=0;
      int end=a.size()-1;
      int indx=-1;
      while(start<=end){
        int mid=start+(end-start)/2;
        if(a.get(mid)<=b){
          start=mid+1;
        }
        else
        {
          end=mid-1;
        }
        if(a.get(mid)==b){
          indx=mid;
        }
      }
      return indx;
    }
}
```

Sorted Insert Position

Asked in: Yahoo

Suggest Edit | Bookmark

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

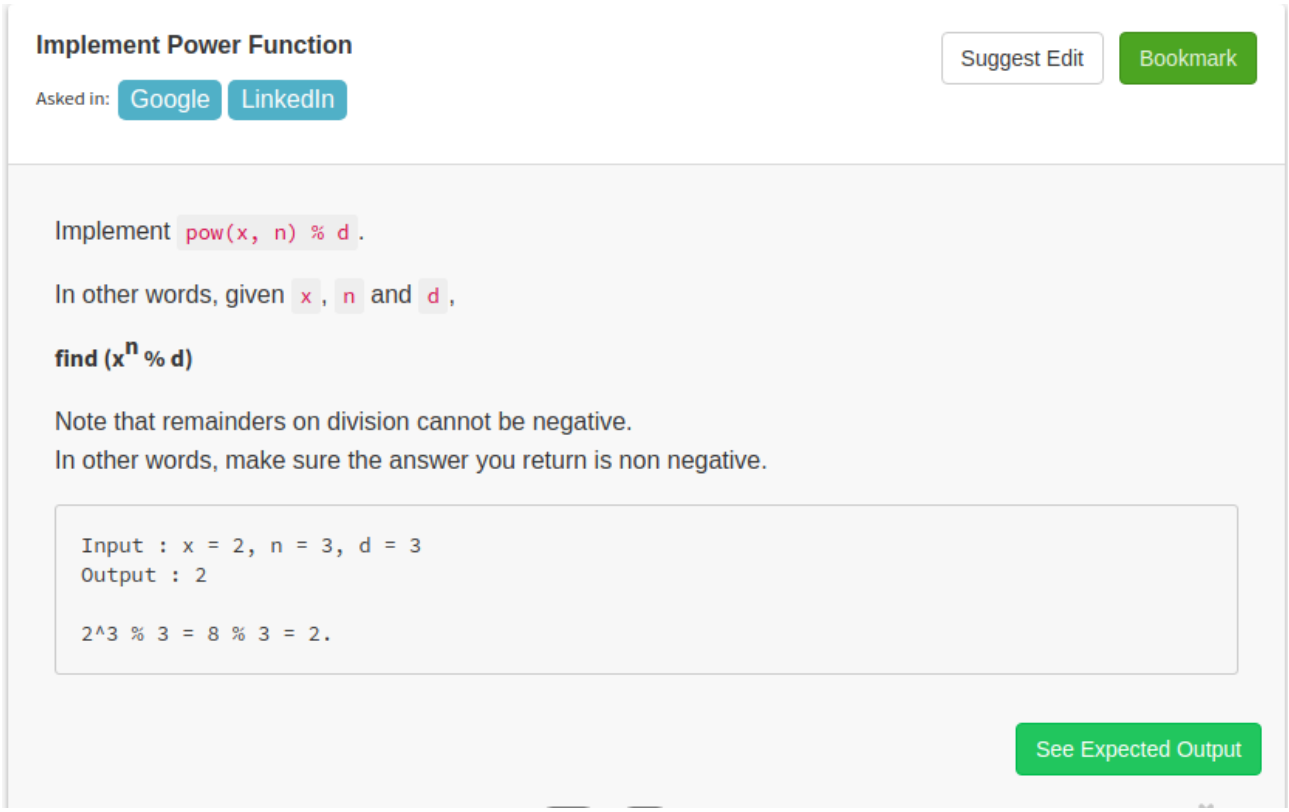
Here are few examples.

```
[1,3,5,6], 5 → 2
[1,3,5,6], 2 → 1
[1,3,5,6], 7 → 4
[1,3,5,6], 0 → 0
```

See Expected Output

```
public class Solution {
    public int searchInsert(ArrayList<Integer> a, int b) {
        if(b==0){
            return 0;
        }
        if(b>a.get(a.size()-1)){
            return a.size();
        }
        int start=0;
        int end=a.size();
        while(start<end){
            int mid=start+(end-start)/2;
            if(b>a.get(mid))
            {
                start=mid+1;
            }
            else
            {
```

```
            end=mid;
          }
         }
        return end;
    }
}
```



Implement Power Function

Asked in: Google LinkedIn

Suggest Edit | Bookmark

Implement `pow(x, n) % d`.

In other words, given `x`, `n` and `d`,

find (x^n % d)

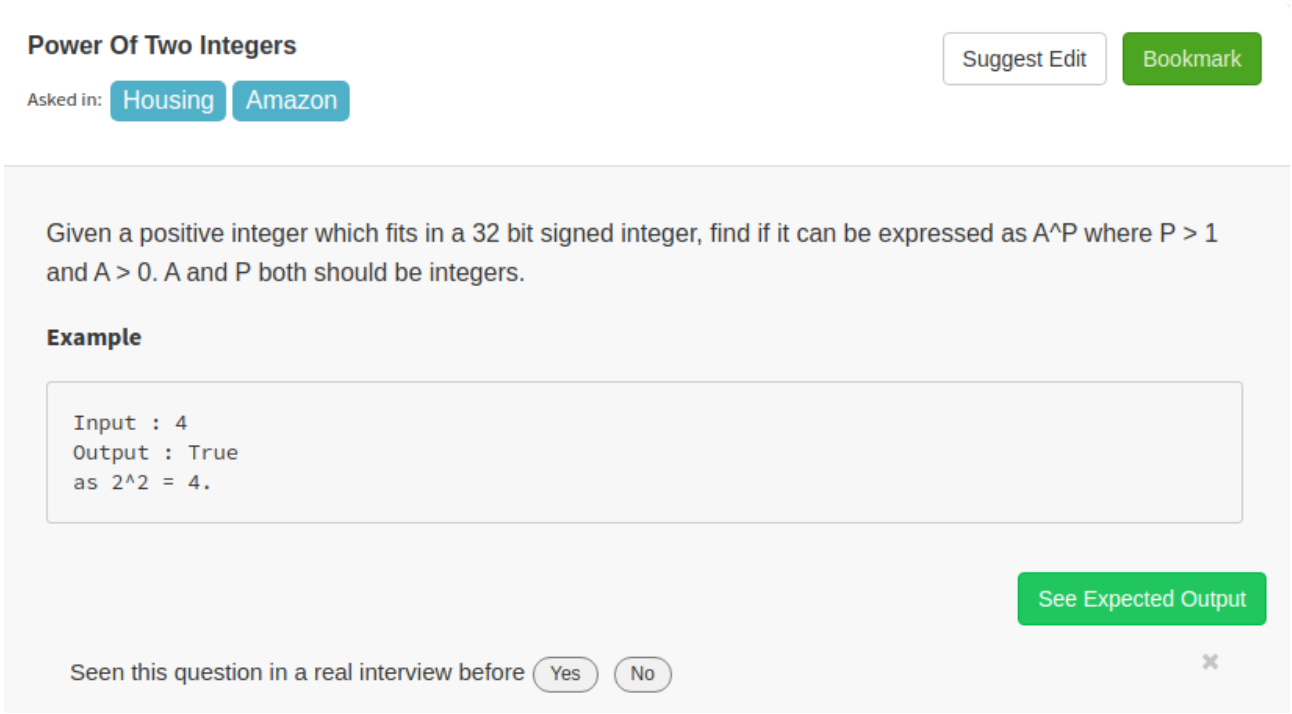
Note that remainders on division cannot be negative.
In other words, make sure the answer you return is non negative.

```
Input : x = 2, n = 3, d = 3
Output : 2

2^3 % 3 = 8 % 3 = 2.
```

See Expected Output

```
public class Solution {
    public int pow(int x, int n, int d) {
        if(n==0){
            return x==0?0:1;
        }else if(n==1){
            x = x%d;
            if(x<0){
                x += d;
            }
            return x;
```

```
        }
        long smaller = pow(x,n/2,d);
        long result = (smaller*smaller)%d;
        if(result<0){
            result += d;
        }
        if(n%2!=0){
            result = (result*x)%d;
            if(result<0){
                result+=d;
            }
        }
        return (int)(result%d);
}
}
```




```
public class Solution {
```

```
    public int isPower(int A) {
        if(A==1){
            return 1;
        }
        int m=2;
        while(m<A){
            int n=2;
            while(n<A){
                int x=(int)Math.pow(m,n);
                if(x==A){
                    return 1;
                }
                else if(x>A){
                    break;
                }
                else{
                    n++;
                }
            }
            m++;
        }
        return 0;
    }
}
```


```
public class Solution {
    public int trailingZeroes(int n) {
      int i=5;
      int flag=0;
      while(n/i>=1){
          flag+=Math.floor(n/i);
          i*=5;
      }
      return flag;
    }
}
```

Great candidates usually are not about the money, they are about the opportunity, they will take the job as an opportunity to meet their personal life plans, ambitions, and goals. So when you talk to the right candidates, remember to give them an offer to grow at your company, and meet their goals and don't make it all about the money. Thoughts?

Grid Unique Paths

Asked in: Google Amazon Microsoft Adobe

Suggest Edit | Bookmark

A robot is located at the top-left corner of an **A x B grid** (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?

Note: A and B will be such that the resulting answer fits in a 32 bit signed integer.

Example :

```
Input : A = 2, B = 2
Output : 2

2 possible routes : (0, 0) -> (0, 1) -> (1, 1)
              OR  : (0, 0) -> (1, 0) -> (1, 1)
```

See Expected Output

```
public class Solution {
    public int uniquePaths(int A, int B) {
    int[][] r=new int[A][B];
    for(int i=0;i<A;i++){
        r[i][0]=1;
    }
    int i;
    for(i=0;i<B;i++){
```

```
        r[0][i]=1;
    }
    for(i=1;i<A;i++){
        for(int j=1;j<B;j++){
            r[i][j]=r[i-1][j]+r[i][j-1];
        }
    }
    return r[A-1][B-1];
    }
}
```

3 Sum

Asked in: Facebook Amazon Microsoft

Suggest Edit | Bookmark

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target.
Return the sum of the three integers.

Assume that there will only be one solution

Example:
given array `S = {-1 2 1 -4}`,
and `target = 1`.

The sum that is closest to the target is `2`. `(-1 + 2 + 1 = 2)`

See Expected Output

```
public class Solution {
    public int threeSumClosest(ArrayList<Integer> a, int b) {
        int r=0;
        int min=Integer.MAX_VALUE;
        Collections.sort(a);
        for(int i=0;i<a.size()-2;i++){
            int j=i+1;
            int k=a.size()-1;
```

```
            while(j<k){
                int sum=a.get(i)+a.get(j)+a.get(k);
                int d=Math.abs(sum-b);
                if(d==0){
                    return sum;
                }
                if(d<min){
                    r=sum;
                    min=d;

                }
                if(sum<b){
                   j++;
                }
                else{
                    k--;
                }
            }
        }
        return r;
    }
}
```

Minimize the absolute difference

Suggest Edit | Bookmark

Asked in: Microsoft

Problem Setter: ganeshk2 Problem Tester: dhruvi

Given three sorted arrays `A`, `B` and `C` of not necessarily same sizes.

Calculate the **minimum absolute difference** between the maximum and minimum number from the triplet a, b, c such that a, b, c belongs arrays `A`, `B`, `C` respectively.
i.e. minimize **| max(a,b,c) - min(a,b,c) |**.

Example :

Input:

```
A : [ 1, 4, 5, 8, 10 ]
B : [ 6, 9, 15 ]
C : [ 2, 3, 6, 6 ]
```

Output:

```
1
```

```
public class Solution {
    public int solve(ArrayList<Integer> a, ArrayList<Integer> b,
ArrayList<Integer> c) {
        int i=a.size()-1;
        int j=b.size()-1;
        int k=c.size()-1;


        int min_diff=Math.abs(Math.max(a.get(i),Math.max(b.get(j),c.get(k)))
        -Math.min(a.get(i),Math.min(b.get(j),c.get(k))));
        while(i!=-1&&j!=-1&&k!=-1){
            int cur_diff=Math.abs(Math.max(a.get(i),Math.max(b.get(j),c.get(k)))
        -Math.min(a.get(i),Math.min(b.get(j),c.get(k))));
           if(cur_diff<min_diff){
               min_diff=cur_diff;
           }
```

```
            int max_num=Math.max(a.get(i),Math.max(b.get(j),c.get(k)));
            if(max_num==a.get(i)){
                i--;
            }
            else if(max_num==b.get(j)){
                j--;
            }
            else{
                k--;
            }
        }
        return min_diff;
    }
}
```

Intersection Of Sorted Arrays

Asked in: Facebook Google

Suggest Edit | Bookmark

Find the intersection of two sorted arrays.

OR in other words,

Given 2 sorted arrays, find all the elements which occur in both the arrays.

Example :

```
Input :
    A : [1 2 3 3 4 5 6]
    B : [3 3 5]

Output : [3 3 5]

Input :
    A : [1 2 3 3 4 5 6]
    B : [3 5]

Output : [3 5]
```

```
public class Solution {
    // DO NOT MODIFY THE LIST. IT IS READ ONLY
```

```
    public ArrayList<Integer> intersect(final List<Integer> a, final
List<Integer> b) {
        ArrayList<Integer> result=new ArrayList<Integer>();
       int i=0;
       int j=0;
       while(i<a.size() && j<b.size()){
          if(a.get(i)<b.get(j)){
             i++;
          }
          else if(a.get(i)>b.get(j)){
             j++;
          }
          else{
             result.add(a.get(i));
             i++;
             j++;
          }
       }
       return result;

    }
}
```