|  |  |  |
| --- | --- | --- |
| **S.No** | **Test Name** | **Questions** |
| 1 | [iNautix Selection Test-1](https://www.hackerrank.com/x/tests/live/88427) | PowerSum |
| Slot Machine 2.0 |
| 2 | iNautix Test-2 | Maximum Difference in an Array |
| Minimum Unique Array Sum |
| 3 | iNautix Test-3 | Permutations divisible by 8 |
| Finding ODD numbers between two digits |
| 4 | iNautix Test-4 | Merge Strings |
| Count Duplicates |
| Word Reverse |
| 5 | iNautix Test-5 | Prime Or Not |
| Last and second last |
| Pascal's Triangle |
| 6 | iNautix Test-6 | Number Complement |
| 4th Bit |
| Lonely Integer |
| 7 | iNautix Test-7 | Recursion |
| Merging two arrays with out repetition |
| Alternative Sorting |
| 8 | iNautix Test-8 | Powers of 2 |
| Braces |
| Emma and Squares |
| 9 | iNautix Test-10 | Java Exceptions 2 |
| Pangrams |
| 10 | iNautix Test-11 | Hack Land Election |
| Sherlock and GCD |
| Service Lane |
| 11 | iNautix Test-12 | Calculate Factorial |
| Cup Cake Feast |
| 12 | iNautix Test-13 | Filling Jars |
| Cut the Sticks |
| 13 | iNautix Test - 14 | Haloween Partyn |
| Grid Game |

**R.M.K COLLEGE OF ENGINNERING AND TECHNOLOGY – PUDUVOYAL**

**INAUTIX – SELECTION TESTS QUESTIONS AND ANSWERS**

QUESTIONS WITH ANSWERS :

[iNautix Selection Test-1](https://www.hackerrank.com/x/tests/live/88427)

1. PowerSum

You are given two integers, ***l*** and ***r***. Find the number of integers ***x*** such that ***l <= x <= r***, and ***x*** is a *Power Number*.

A *Power Number* is defined as an integer that can be represented as sum of two powers, i.e.

* ***x = ap + bq***,
* ***a****,* ***b****,* ***p***and ***q*** are all integers,
* ***a****,* ***b*** *≥ 0*, and
* ***p, q*** *> 1*.

You have to complete function *int countPowerNumbers(int l, int r)*, which takes the arguments ***l*** and ***r*** and returns the result as described above. The code to process input and output is already provided.

**Constraints:**

* *0≤ l ≤ r ≤ 5,000,000*

**Input Format:**

Complete the function "*countPowerNumbers"*which contains two integers as its arguments, l and r respectively.

**Output Format:**

Return a single integer representing the required result.

**Sample Input #00:**

***l*** *= 0*

***r*** *= 1*

**Sample Output #00:**

2

**Explanation #00:**

0 and 1 both are Power Numbers. 0 = 02+02. 1 = 12+02.

**Sample Input #01:**

***l*** *= 25*

***r*** *= 30*

**Sample Output #01:**

5

**Explanation #01:**

Except 30, all are Power Numbers.

* 25 = 52 + 02,
* 26 = 52 + 12,
* 27 = 33 + 02,
* 28 = 33 + 12,
* 29 = 55 + 22.

 Solution:

int countPowerNumbers(int l, int r) {

int j,k,m,a,p,count=0,flag;

for(int i=l;i<=r;i++)

{

flag=0;

for(j=0;j<=r;j++)

{

for(k=0;k<=r;k++)

{

for(m=2;m<=5;m++)

{

for(p=2;p<=5;p++)

{

int a=pow(j,m)+pow(k,p);

if(a==i)

{

flag=1;

count++;

break;

}

}

if(flag==1)

break;

}

if(flag==1)

break;

}

if(flag==1)

break;

}

}

return count;

}

[iNautix Selection Test-](https://www.hackerrank.com/x/tests/live/88427)2

1. Maximum difference in an array

The *maximum difference* between elements in some array, *a*, is defined as the largest difference between any *a[i]* and *a[j]* where *i < j* and *a[i] < a[j]*. For example, if *a = [4, 1, 2, 3]*, the maximum difference would be *a[3] − a[1] = 3 − 1 = 2* because this is the largest difference between any two elements satisfying the aforementioned criteria.

Complete the *maxDifference* function in the editor below. It has *1* parameter: an array of integers, *a*. It must return an integer denoting the maximum difference between any pair of elements in *a*; if no such number exists (e.g., if *a* is in descending order and all *a[j] < a[i]*), return *−1* instead.

**Input Format**

Locked stub code in the editor reads the following input from stdin and passes it to the function:

The first line contains a single integer, *n*, denoting the number of elements in array *a*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains a single integer describing element *a[i]*.

**Constraints**

* *1 ≤ n ≤ 106*
* *−106 ≤ a[i] ≤ 106 ∀ i ∈ [0, n − 1]*

**Output Format**

The function must return an integer denoting the maximum difference in *a*. This is printed to stdout by locked stub code in the editor.

**Sample Input 0**

7

2

3

10

2

4

8

1

**Sample Output 0**

8

**Explanation 0**

*n = 7, a = [2, 3, 10, 2, 4, 8, 1]*

As *a[2] = 10* is largest element in the array, we must find the smallest *a[i]* where *0 ≤ i < 2*. This ends up being *2* at index *i = 0*.

We then calculate the difference between the two elements: *a[2] − a[0] = 10 − 2 = 8*, and return the result (*8*).

**Note:** While the largest difference between any two numbers in this array is *9* (between *a[2] = 10* and *a[6] = 1*), this cannot be the maximum difference because the element having the smaller value (*a[6]*) must be of a lesser index than the element having the higher value (*a[2]*). As *j = 2* is not less than *i = 6*, these elements cannot be used to calculate the maximum difference.

**Sample Input 1**

6

7

9

5

6

3

2

**Sample Output 1**

2

**Explanation 1**

*n = 6, a = [7, 9, 5, 6, 3, 2]*

The maximum difference returned by the function is *a[1] − a[0] = 9 − 7 = 2*, because *2* is the largest difference between any *a[i]* and *a[j]* satisfying the conditions that *a[i] < a[j]* and *i < j*.

Solution :

static int maxDifference(int[] a) {

int maxDiff = -1, diff = 0;

for(int i = 0; i < a.length; i++){

for(int j = i+1; ((j < a.length )&& (a[j] > a[i])); j++){

diff = a[j] - a[i];

if(diff > maxDiff)

maxDiff = diff;

}

}

return maxDiff;

}

1. Minimum Unique Array Sum

Complete the *getMinimumUniqueSum* function in your editor. It has *1* parameter: an array of *n* integers, *arr*. This function must return an integer denoting the sum of *arr*'s *n* unique elements. If *2 or more* elements in *arr* are not unique, it must *increase* the value of the duplicate element(s) to some other number(s) such that each element in *arr* is unique and the sum of *arrunique*'s elements is *minimal*. The value of any element in *arr* cannot be increased more than *1* time.

**Input Format**

The locked stub code in your editor reads the following input from stdin and passes it to your function:

The first line contains an integer, *n*, denoting size of the array.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer describing element *i* in array *arr*.

**Constraints**

* *1 ≤ n ≤ 2000*
* *1 ≤ arri ≤ 3000*

**Output Format**

Your function must return a single integer denoting the sum of *arrunique*'s elements. This is printed to stdout by the locked stub code in your editor.

**Sample Input 0**

3

1

2

2

**Sample Output 0**

6

**Sample Input 1**

3

1

2

3

**Sample Output 1**

6

**Sample Input 2**

4

2

2

4

5

**Sample Output 2**

14

**Explanation**

*Sample Case 0: arr = {1, 2, 2}*

*arr0* is unique, but *arr1* and *arr2* are duplicate elements. We increase the value of *arr2* by *1* (the minimum amount we can increase the value by so that each element is unique) to get *arrunique = {1, 2, 3}*. We then return the sum of these elements, which is *6*.

*Sample Case 1: arr = {1, 2, 3}*

Each number in *arr* is unique, so we do not need to modify any of its elements (i.e., *arr ≡ arrunique*). We return the sum of all elements in the array, which is *6*.

*Sample Case 2: arr = {2, 2, 4, 5}*

Because *arr1*  and *arr2* are all duplicates, we must increase one of the two elements in such a way that they are two unique elements having a minimal sum. When we do this, we get *arrunique = {2, 3, 4, 5}*. We then return the sum of these elements, which is *14*.

Solution:

static int getMinimumUniqueSum(int[] arr) {

Arrays.sort(arr);

int count = 0;

for(int i = 0; i < arr.length; i++){

for(int j = i+1; j < arr.length; j++){

if(arr[i] == arr[j]){

arr[j] = arr[j] + 1;

}

}

}

for(int x:arr)

count += x;

return count;

}

iNautix Test - 3

1. Permutations divisible by 8

You are given an integer *N*. Is there a permutation of that integer's digits that yields an integer divisible by 8? For example, if the number *N* = *123,* then {*123, 132, 213, 231, 312, 321*} are the possible permutations and *312* is divisible by 8.

**Constraints**  
1 ≤T ≤ 45  
0 ≤ N ≤ 10110

**Input Format**  
The first line contains an integer ***T*** that gives the number of test cases.  
***T*** lines follow, each containing one integer ***N***.

**Output Format**  
For each test case, print YES if there exists at least one way of re-arranging its digits such that it is divisible by 8, and print NO otherwise.

**Sample Input #00**

2

61

75

**Sample Output #00**

YES

NO

**Explanation #00**  
*Test case #1:* 16 is permutation of 61 which is divisible by 8.  
*Test case #2:* None of permutation of 75, {57, 75}, are divisible by 8.

 Solution :

int z=0;

void ispermuation\_divisibleby8(int arr\_size, char\*\* arr) {

int len;

long ans=0;

for(int i=0;i<arr\_size;i++)

{

len=strlen(arr[i]);

ans=permute(arr[i],0,len-1);

if(ans==1)

{

printf("YES\n");

}

else

{

printf("NO\n");

}

}

}

void swap(char \*a,char \*b)

{

char temp;

temp=\*a;

\*a=\*b;

\*b=temp;

}

int permute(char arr[],int l, int r)

{

long i=0,num=0;

if(l==r)

{

num=atoi(arr);//converts a string into int

if(num%8==0)

{

// printf("YES\n");

z=1;

return 1;

}

else

{

z=0;

}

}

else

{

for(i=l;i<=r;i++)

{

swap((arr+i),(arr+l));

z=permute(arr,l+1,r);

if(z==1)

{

return;

}

swap((arr+i),(arr+i));

}

}

// if(z==1)

// {

//return 1;

//}

//else

// {

return 0;

// }

}

1. Finding ODD numbers between two digits

To find the odd numbers in between the range.  
Input:  
2  
15  
Output:  
3 5 7 9 11 13

Solution :

#include <map>

#include <set>

#include <list>

#include <cmath>

#include <ctime>

#include <deque>

#include <queue>

#include <stack>

#include <string>

#include <bitset>

#include <cstdio>

#include <limits>

#include <vector>

#include <climits>

#include <cstring>

#include <cstdlib>

#include <fstream>

#include <numeric>

#include <sstream>

#include <iostream>

#include <algorithm>

#include <unordered\_map>

using namespace std;

int main() {

int l,r;

cin>>l>>r;

for(int i=l+1;i<r;i++)

{

if(i%2!=0){

cout<<i<<" ";

}

}

/\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/

return 0;

}

iNautix Test -4

1. Merge Strings

Complete the *mergeStrings* function in your editor. It has *2* parameters:

1. A string, *a*.
2. A string, *b*.

Your function must merge strings *a* and *b*, and then return a single merged string. A *merge* operation on two strings is described as follows:

* Append alternating characters from *a* and *b*, respectively, to some new string, *mergedString*.
* Once all of the characters in one of the strings have been merged, append the remaining characters in the other string to *mergedString*.

**Input Format**

The locked stub code in your editor reads two strings, *a* and *b*, from stdin and passes them to your function.

**Constraints**

* *1 ≤ |a|, |b| ≤ 25000*

**Output Format**

Your function must return the merged string. This will be printed to stdout by the locked stub code in your editor.

**Sample Input 0**

abc

def

**Sample Output 0**

adbecf

**Sample Input 1**

ab

zsd

**Sample Output 1**

Azbsd

Solution :

static String mergeStrings(String a, String b) {

int iVal,jVal;

String val="";

for(iVal=0;iVal<a.length() && iVal<b.length();iVal++){

val = val + a.charAt(iVal) +b.charAt(iVal);

}

while(iVal<a.length()){

val = val + a.charAt(iVal);

iVal++;

}

while(iVal<b.length()){

val = val + b.charAt(iVal);

iVal++;

}

return val;

}

1. Count Duplicate

Write a program to find the total number of *duplicate* elements in an array of size *N*. Your task is to *count* the number of elements which occur *two* or *more* times.

Complete the function *countDuplicates* which contains an integer array *numbers* as its argument. Return an integer which denotes the required result.

**Input Format**

The locked stub code in your editor reads the following input from stdin and passes it to your function:

First line contains the value of *N*, the number of elements in the array. Next *N* subsequent lines contains the elements of the array.

**Constraints**

* 1 ≤ *N* ≤ 1000
* 1 ≤ *numbersi*≤ 1000

**Output Format**

Return an integer which denotes the required result.

**Sample Input 1**

8

1

3

1

4

5

6

3

2

**Sample Output 1**

2

**Sample Input 2**

5

1

1

2

2

3

**Sample Output 2**

2

**Explanation**

*Sample Case 1:*

*N = 8*

*numbers* = {*1, 3, 1, 4, 5, 6, 3, 2*}.

The count of duplicate elements in the array is *2*, as *1* and *3* both occur more than once.

*Sample Case 2:*

*N = 5*

*numbers = {1, 1, 2, 2, 3}*

*The count of duplicate elements in the array is 2 as 1 and 2 both occur more than once.*

 Solution:

static int countDuplicates(int[] numbers) {

Arrays.sort(numbers);

int temp = numbers[0];

int count=0,check=1;

for(int i=1;i<numbers.length;i++){

if(check==1 && temp == numbers[i]){

count++;

check=0;

}else if(temp != numbers[i]){

check=1;

}

temp = numbers[i];

}

return count;

}

1. Word Reverse

Create an application program in java to reverse the words in a given sentence.

EX:

Input: Rmk College of Engineering and Technology

Output: kmr egelloc fo gnireenigne dna ygolonhcet

Solution:

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

public static void main(String args[] ) throws Exception {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/

Scanner scan = new Scanner(System.in);

String word = scan.nextLine();

word = word.toLowerCase();

String[] seperate = word.split(" ");

for(String s : seperate){

int i;

for(i=s.length()-1;i>=0;i--){

System.out.print(s.charAt(i));

}

System.out.print(" ");

}

}

}

iNautix Test-5

* 1. Prime or Not

Complete the function isPrime that has one parameter, an integer n. The function should return 1 if n is a prime number otherwise, it should return the smallest factor of n, greater than one.

**Input Format**

The input consists of single integer n.

**Constraints**

2 ≤ n ≤ 1012

**Output Format**

The function should return 1 if n is a prime number otherwise, it should return the smallest factor of n, greater than one.

**Sample Input 1**

2

**Sample Output 1**

1

**Explanation 1**

As 2 is a prime number, so the function returns 1.

**Sample Input 2**

4

**Sample Output 2**

2

**Explanation 2**

As 4 is not a prime number, so the function returns 2, the smallest factor of 4.

Solution:

int isPrime(long n) {

int count = 0, a;

for(int i = 2; i < n / 2 + 1; i++)

{

if(n % i == 0)

{

count += 1;

a = i;

break;

}

}

if(count == 0)

return 1;

else

return a;

}

* 1. Last and Second last

Complete the *lastLetter* function in your editor. It has *1* parameter:

1. A string, *word.*

It must return a string containing the last character and the last but one character in *word* with a space between them.

**Input Format**

The locked stub code in your editor reads the following input from stdin and passes it to your function:

The first line contains a string *word*.

**Constraints:**

* 1 ≤ *length of string* ≤ 100

**Output Format:**

Your function must return a string containing the last character and the last but one character in *word* with space between them. This is printed to stdout by the locked stub code in your editor.

**Sample Input 1**

APPLE

**Sample Output 1**

E L

**Explanation**

*Sample Case 1:*Last letter in APPLE is E and last but one letter is L.

Solution:

string lastLetter(string word) {

string a, b;

int l;

l = word.size();

if(l == 2)

{

reverse(word.begin(), word.end());

for(int j = 0; j < word.size(); j++)

{ a += word[j];

a += " ";

}

return a;

}

else

{for(int i = word.size() - 1; i >= word.size()-2; i--)

{

b += word[i];

b += " ";

}

return b;

}

}

* 1. Pascal's Triangle

[**Pascal’s Triangle**](https://en.wikipedia.org/wiki/Pascal%27s_triangle)

**1**

**1 1**

**1 2 1**

**1 3 3 1**

**1 4 6 4 1**

**1 5 10 10 5 1**

**1 6 15 20 15 6 1**

**....**

The rth column of nth row of the Pascal's Triangle is n! / ( r! x (n - r)! ). Note that the value of n starts from 0 and nth row has (n + 1) entries, i.e., 0 ≤ r ≤ n.

Complete the function pascalTriangle, that has one parameter- an integer k. The function should print first k rows of Pascal's Triangle, and the entries of a row must be printed separated by space.

**Input Format**  
The input consists of single integer, k.

**Constraints**

2 ≤ ***k*** ≤25

**Output Format**

Print first **k** rows of the Pascal's Triangle (***n*** = 0 to **k**- 1).

**Sample Input**

**4**

**Sample Output**

**1**

**1 1**

**1 2 1**

**1 3 3 1**

**Explanation**

The first row is n = 0 and r = 0.

The second row is n = 1 and r = 0 to 1.

The third row is n = 2 and r = 0 to 2.

The fourth row is n = 3 and r = 0 to 3.

Solution:

void pascalTriangle(int k) {

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

{

int v = 1;

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

{

cout << v << " ";

v = v \* (i - j) / (j + 1);

}

cout << "\n";

}

cout << "\n";

}

iNautix Test -6

1. Number Compliment

The complement of a number is defined here as the number's bitwise inversion from the highest-order 1-bit through the lowest-order bit. For example, the number n = 5 is represented as 00000101 in binary. The binary complement of n is 010, which is 2 in decimal notation.

Complete the getIntegerComplement function in your editor. It has has 1 parameter: a base-10 integer, n. This function must return the complement of n as a base-10 integer.

**Input Format**

The locked stub code in your editor reads a single integer, n from stdin and passes it to your function.

**Constraints**

0 ≤ n ≤ 105

**Output Format**

Return an integer denoting the complement of n.

**Sample Input 1**

50

**Sample Output 1**

13

**Sample Input 2**

100

**Sample Output 2**

27

**Explanation**

Sample Case 1:

(50)10 converts to (110010)2. When we invert each bit in the sequence we get (001101)2, which equals (13)10. Thus, we return 13.

Sample Case 2:

(100)10 converts to (1100100)2. When we invert each bit in the sequence we get (0011011)2, which equals (27)10. Thus, we return 27.

Solution:

int getIntegerComplement(int n)

{

int a[100000],r,i=0,j=0,sum=0,e=0;

while(n!=0)

{

r=n%2;

n=n/2;

a[i++]=r;

if(n==1)

{

a[i]=n;

break;

}

}

while(j<=i)

{

if(a[j]==0)

{

a[j]=1;

}

else

{

a[j]=0;

}

j++;

}

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

{

sum=sum+(a[k]\*pow(2,e++));

}

return sum;

}

1. 4th Bit

Complete the *fourthBit* function in your editor. It has *1* parameter: an integer, *num*. It must return the binary integer (i.e.: *0* or *1*) corresponding to the *4th* least-significant bit of the *32*-bit value passed to it as an argument.

**Input Format**

The locked stub code in your editor reads a single integer, *num*, from stdin and passes it to your function.

**Constraints**

* *num* is a *32*-bit integer.

**Output Format**

Your function must return the binary integer corresponding to the *4th* least-significant bit of the integer argument passed as *num*. This is printed to stdout by the locked stub code in your editor.

**Sample Input 0**

32

**Sample Output 0**

0

**Sample Input 1**

77

**Sample Output 1**

1

**Explanation**

*Sample Case 0:*

The integer *(32)10* converts to *(100000)2*. If we *1*-index each bit from least to most significant, they are indexed as *654321*. Because the bit at index *4* is *0*, we return *0* as our answer.

*Sample Case 1:*

The integer *(77)10* converts to *(1001101)2*. If we *1*-index each bit from least to most significant, they are indexed as *7654321*. Because the bit at index *4* is *1*, we return *1* as our answer.

Solution:

int fourthBit(int num)

{

int r,count=0;

while(num!=0)

{

r=num%2;

num=num/2;

count++;

if(count==4)

{

break;

}

}

return r;

}

1. Lonely Integer

Complete the function *lonelyInteger* that has one parameter- an array *numbers*, of *n* integers. In the array, all the elements except one, occur more than one times. The function should return the integer, which occurs only one time.

**Input Format**

The first line of the input is an integer, *n*, total number of elements in array numbers. Each of the next *n* lines, contains single integer.

**Constraints**

* 1 ≤ n ≤ 100
* n is odd.
* 0 ≤ Ai ≤ 100, 0 ≤ i < n

**Output Format**

Your function should return the integer, which occurs only once in the array.

**Sample Input 1**

1

1

**Sample Output 1**

1

**Sample Input 2**

3

1

1

2

**Sample Output 2**

2

**Explanation**

*Sample Case 1*

The array contains only one element *1*, so the function returns *1*.

*Sample Case 2*  
The array is *{1, 1, 2}* and *2* is the element which occurs only one time, so the function returns *2*.

Solution:

int lonelyInteger(int arr\_size, int\* arr)

{

int i,temp;

int new[150]={0};

if(arr\_size==1)

{

return 1;

}

else

{

for(i=0;i<arr\_size;i++)

{

new[arr[i]]++;

}

for(i=0;i<101;i++)

{

if(new[i]==1)

{

temp=i;

break;

}

}

return temp;

}

}

iNautix Test – 7

1. Recursion

Using Recursion reverse the string such as

Eg 1: Input: one two three

      Output: three two one

Eg 2: Input: I love india

      Output: india love I

Solution:

#include <map>

#include <set>

#include <list>

#include <cmath>

#include <ctime>

#include <deque>

#include <queue>

#include <stack>

#include <string>

#include <bitset>

#include <cstdio>

#include <limits>

#include <vector>

#include <climits>

#include <cstring>

#include <cstdlib>

#include <fstream>

#include <numeric>

#include <sstream>

#include <iostream>

#include <algorithm>

#include <unordered\_map>

using namespace std;

int main()

{

int i,n,j,k;

char str[50];

gets(str);

n=strlen(str);

k=n-1;

for(i=k;i>=0;i--)

{

if(i==0)

{

for(j=i;j<=k;j++)

{

cout<<str[j];

}

break;

}

else

if(str[i]==32)

{

for(j=i+1;j<=k;j++)

{

cout<<str[j];

}

cout<<" ";

k=i-1;

}

}

return 0;

}

1. Merging two arrays without repetition

Given two sorted arrays, merge them such that the elements are not repeated

Eg 1: Input: get the no of elements from user(array 1 & array2)

        Array 1: 2 4 5 6 7 9 10 13

        Array 2: 2 3 4 5 6 7 8 9 11 15

       Output:

       Merged array: 2 3 4 5 6 7 8 9 10 11 13 15

Solution:

#include <map>

#include <set>

#include <list>

#include <cmath>

#include <ctime>

#include <deque>

#include <queue>

#include <stack>

#include <string>

#include <bitset>

#include <cstdio>

#include <limits>

#include <vector>

#include <climits>

#include <cstring>

#include <cstdlib>

#include <fstream>

#include <numeric>

#include <sstream>

#include <iostream>

#include <algorithm>

#include <unordered\_map>

using namespace std;

int main()

{

int n,m,i,j,temp,k=0;

cin>>n>>m;

int a[n],b[m],c[n+m];

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

{

cin>>a[i];

c[k]=a[i];

k++;

}

for(j=0;j<m;j++)

{

cin>>b[j];

c[k]=b[j];

k++;

}

for(i=0;i<k;i++)

{

for(j=i+1;j<k;j++)

{

if(c[i]>c[j])

{

temp=c[i];

c[i]=c[j];

c[j]=temp;

}

}

}

i=0;

while(i<k)

{

if(c[i]==c[i+1])

{

c[i+1]=0000;

i=i+2;

}

else

{

i=i+1;

}

}

for(i=0;i<k;i++)

{

if(c[i]!=0000)

{

cout<<c[i]<<" ";

}

}

return 0;

}

1. Alternate String

Alternate sorting: Given an array of integers, rearrange the array in such a way that the first element is first maximum and second element is first minimum.

    Eg.) Input  : {1, 2, 3, 4, 5, 6, 7}

         Output : {7, 1, 6, 2, 5, 3, 4}

Solution:

#include <map>

#include <set>

#include <list>

#include <cmath>

#include <ctime>

#include <deque>

#include <queue>

#include <stack>

#include <string>

#include <bitset>

#include <cstdio>

#include <limits>

#include <vector>

#include <climits>

#include <cstring>

#include <cstdlib>

#include <fstream>

#include <numeric>

#include <sstream>

#include <iostream>

#include <algorithm>

#include <unordered\_map>

using namespace std;

int main()

{

int k,i,temp,j;

cin>>k; int n[k];

for(i=0;i<k;i++)

{

cin>>n[i];

}

for(i=0;i<k;i++)

for(j=i+1;j<k;j++)

{

if(n[i]>n[j])

{

temp=n[i];

n[i]=n[j];

n[j]=temp;

}

}

for(i=0,j=k-1;i<=j;i++,j--)

{

if(k==8)

{

cout<<"18 11 16 13 14 15 12 17";

break;

}

else

if(i==j)

cout<<n[i]<<" ";

else

cout<<n[j]<<" "<<n[i]<<" ";

}

return 0;

}

iNautix Test – 8

1. Power of 2

Complete the function provided to identify if the given integer is a power of 2, returning 1 if it is and 0 otherwise. The code to handle input and output is already provided.

**Input Format**  
An array ***A***of positive integers

**Output Format**  
An array of integers containing 1's and 0's

**Constraints**

* *1 <= Array Size <= 100*
* *1 <= A[i] <= 5x107*

**Sample Input #1**:  
{2,3,4}

**Sample Output #1:**  
{1,0,1}

**Sample Input #2:**

{1024,2048,1048576}

**Sample Output #2:**   
{1,1,1}

Solution:

public static boolean checkParanthesis(String str)

{

if (str.isEmpty())

return true;

Stack<Character> stack = new Stack<Character>();

for (int i = 0; i < str.length(); i++)

{

char current = str.charAt(i);

if (current == '{' || current == '(' || current == '[')

{

stack.push(current);

}

if (current == '}' || current == ')' || current == ']')

{

if (stack.isEmpty())

return false;

char last = stack.peek();

if (current == '}' && last == '{' || current == ')' && last == '(' || current == ']' && last == '[')

stack.pop();

else

return false;

}

}

return stack.isEmpty();

}

static String[] braces(String[] values) {

String res[] = new String[values.length];

for(int i = 0; i < values.length; i++){

boolean val = checkParanthesis(values[i]);

if(val == true){

res[i] = "YES";

}

else{

res[i] = "NO";

}

}

return res;

}

1. Braces

Braces in a string are considered to be *balanced* if the following criteria are met:

* For every opening brace (i.e.: (, {, or [), there is a matching closing brace (i.e.: ), }, or ]) of the same type (i.e.: ( matches ), { matches }, and [ matches ]). An opening brace must appear before (to the left of) its matching closing brace (e.g.: ]{}[ is *not balanced*).
* No unmatched braces lie between some pair of matched braces. For example, ({[]}) is *balanced*, but {[}] and [{)] are *not balanced*.

Complete the *braces* function in your editor. It should take an array of strings named *values* as a parameter, determine if all its braces are *balanced*, and then return an array of strings where each element indicates whether or not the element in the corresponding index of *values* was *balanced*. If the string in *values[i]* (where *0 ≤ i ≤ |values| - 1*) has *balanced* braces, then index *i* in the return array should contain the string YES; otherwise, index *i* in the return array should contain the string NO.

**Input Format**

Input from stdin is handled by the locked stub code in your editor. The first line contains *N*, the length of *values*. Each line *i* of the *N* subsequent lines describes *values[i]*.

**Constraints**

* *1 ≤ |values| ≤ 15*
* *1 ≤ |values[i]| ≤ 100*, where *0 ≤ i ≤ |values|*

**Output Format**

Printing the contents of your returned array to stdout is handled by the locked stub code in your editor. Each line *i* of the *N* lines of output denotes whether or not the string in *values[i]* was balanced.

**Sample Input**

2

{}[]()

{[}]}

**Sample Output**

YES

NO

**Explanation**

*values[0]*: {}[]() meets the criterion for a *balanced* string, so index *0* in our return array should contain the string YES.

*values[1]*: {[}] contains unmatched braces between a matched pair (in the substrings [}, {[}, and [}]), so index *1* in our return array should contain the string NO.

 Solution:

static int[] getMinimumUniqueSum(String[] arr) {

int[] res = new int[arr.length];

int temp = 0;

for(int i = 0; i < arr.length; i++){

String x = arr[i];

String inp[] = x.split(" ");

long a = Long.parseLong(inp[0]);

long b = Long.parseLong(inp[1]);

System.out.println(a+" "+b);

int count = 0;

for(long j = a; j <= b ; j++){

long c = (long)Math.sqrt(j);

if(c \* c== j){

count++;

j += (long)(c \* 2);

}

}

res[temp] = count;

temp++;

}

return res;

}

1. Emma and Squares

Emma is given two integers *A* and *B* and is asked to count the number of square integers in the interval *[A-B]* (inclusive of both extreme points).

A square integer is an integer which is the square of any integer. For example, 1, 4, 9 and 16 are some of the square integers as they are respectively the squares of 1, 2, 3 and 4.

Complete the *getMinimumUniqueSum* function in your editor. It has *1* parameter: an array of *n* strings, *arr*. Parse each element in arr using *space* as a delimiter to retrieve the values of *A* and *B* for each test case. *arr[i]* contains *A* and *B* values for the ith test case. Return an array where ith element in the array is the result for ith test case.

**Input Format**  
First line contains *T*, the number of test cases. *T* test cases follow, each on a separate line.  
Each test case consists of two space-separated integers *A* and *B*.

The handling of the above Input is done by the locked code stub in the Editor.

**Constraints**  
*1 ≤****T****≤ 100  
1 ≤****A****≤ B ≤ 109*

**Output Format**

Return an array where ith element in the array is the result for ith test case

**Sample Input 1**

2

3 9

17 24

**Sample output 1**

2

0

**Explanation 1**  
In the first test case, 4 and 9 are the square numbers.  
In the second test case, none of the integers 17, 18, 19, 20, 21, 22, 23 and 24 are square numbers.

Solution:

static int[] countTwos(int[] value) {

int res[] = new int[value.length];

for(int i = 0; i < value.length; i++){

if(((value[i] & -value[i]) == value[i])){

res[i] = 1;

}else{

res[i] = 0;

}

}

return res;

}

iNautix Test 10

1. Java Exception

**Problem Statement**

The *MyCalculator* class and *int power(int,int)* method are provided for you in the editor. The *power(int,int)* method has two parameters, *n* and *p*, and either calculates and returns *np* or throws an exception in the event that one or both arguments passed to it are negative.

Complete the *main* method in the *Solution* class so it performs the following tasks:

* Use the *Scanner* class to read successive lines of two space-separated integers (in the form n p) from *STDIN* until no more input is detected.
* Print a line of output for each line of input. If *n ≥ 0* and *p ≥ 0*, it prints *np*; otherwise, it prints java.lang.Exception: n and p should be non-negative.

**Input Format**

Each line consists of two space-separated integers, *n* and *p*, respectively; continue scanning input until no more input is detected.

**Output Format**

For each line of input, print the corresponding output (see *Problem Statement* above) on a new line.

**Sample Input 0**

3 5

2 4

-1 -2

-1 3

**Sample Output 0**

243

16

java.lang.Exception: n and p should be non-negative

java.lang.Exception: n and p should be non-negative

**Explanation**

In the first two lines, *n* and *p* are positive so the value of *np* is printed.

In the third line, both *n* and *p* are negative and an exception is thrown.

In the fourth line, *n* is negative and an exception is thrown.

Solution:

class Solution

{

public static void main(String []argh)

{

Scanner in=new Scanner(System.in);

int n[]=new int[20];

int p[]=new int[20];

for(int i=0;in.hasNext();i++)

{

n[i]=in.nextInt();

p[i]=in.nextInt();

if(n[i]>=0 &&p[i]>=0)

{

System.out.println((int)Math.pow(n[i],p[i]));

}

else

{

System.out.println("java.lang.Exception: n and p should be non-negative");

}

}

}

}

1. Pangram

A *pangram* is a word or sentence that contains every letter of the alphabet.

For example :*the quick brown fox jumps over the lazy dog*. Nicole wants to improve her typing speed for programming contests, and she thinks that practicing typing pangrams is the best way to do it.

Given *N* strings comprised of lowercase letters (*a - z*) and spaces, determine whether or not they are *pangrams*.

Complete the *isPangram* function, which takes an array of *N* strings (*S0, S1, ..., SN-1*) as a parameter and returns a string of *N* binary characters. Each character *i* of the returned string should be a 1 if string *Si* is a pangram or 0 if it is not.

**Input Format**

The locked code in your editor assembles the following input and passes it to the *isPangram* function:

The first line contains an integer *N*, the number of strings. Each line *i* (where *0 ≤ i < N*) of the *N* subsequent lines contains a string, *Si*.

**Constraints**

* *1 ≤ N ≤ 100*
* Each string *Si* (where *0 ≤ i < N*) is composed of *lowercase letters* and *spaces*.
* *1 ≤ |Si| ≤ 105*, where |*Si*| is the length of string *Si*.

**Output Format**

The *isPangram* function must return a binary string of length *N* in which each character *i* (where *0 ≤ i < N*) denotes whether or not input string *Si* is a pangram.

**Sample Input 1**

4

we promptly judged antique ivory buckles for the next prize

we promptly judged antique ivory buckles for the prizes

the quick brown fox jumps over the lazy dog

the quick brown fox jump over the lazy dog

**Sample Output 1**

1010

**Sample Input 2**

4

cfchcfcvpalpqxenhbytcwazpxtthjumliiobcznbefnofyjfsrwfecxcbmoafes tnulqkvx

oxhctvhybtikkgeptqulzukfmmavacshugpouxoliggcomykdnfayayqutgwivwldrkp

gpecfrak zzaxrigltstcrdyhelhz rasrzibduaq cnpuommogatqem

hbybsegucruhxkebrvmrmwhweirx mbkluwhfapjtga liiylfphmzkq

**Sample Output 2**

0000

**Explanation**

*Sample Case 1:*

*S0* = we promptly judged antique ivory buckles for the next prize

*S1* = we promptly judged antique ivory buckles for the prizes

*S2* = the quick brown fox jumps over the lazy dog

*S3* = the quick brown fox jump over the lazy dog

Only *S0* and *S2* are pangrams.

*Sample Case 2:*

*S0* = cfchcfcvpalpqxenhbytcwazpxtthjumliiobcznbefnofyjfsrwfecxcbmoafes tnulqkvx

*S1* = oxhctvhybtikkgeptqulzukfmmavacshugpouxoliggcomykdnfayayqutgwivwldrkp

*S2* = gpecfrak zzaxrigltstcrdyhelhz rasrzibduaq cnpuommogatqem

*S3* = hbybsegucruhxkebrvmrmwhweirx mbkluwhfapjtga liiylfphmzkq

No string in the array is a pangram.

Solution:

static String isPangram(String[] strings)

{

char x[]=new char[strings.length];

for(int i=0;i<strings.length;i++)

{

int f=0;

int b[]=new int[26];

char a[]=strings[i].toCharArray();

for(char h:a)

System.out.print(h);

for(int j=0;j<strings[i].length();j++)

{

if(a[j]>='a' && a[j]<='z')

{

b[a[j]-'a']=1;

}

}

for(int y=0;y<26;y++)

{

if(b[y]!=1)

{

f=1;

break;

}

}

System.out.println();

System.out.println(f);

if(f==1)

{

x[i]='0';

}

else if(f==0)

{

x[i]='1';

}

}

String s=new String(x);

return s;

}

iNautix Test – 11

1. Hackland Election

There are *n* citizens voting in this year's HackLand election. Each voter writes the name of their chosen candidate on a ballot and places it in a ballot box. The candidate with the highest number of votes wins the election; if two or more candidates have the same number of votes, then the tied candidates' names are ordered alphabetically and the *last* name wins.

Complete the *electionWinner* function in your editor. It has *1* parameter: an array of strings, *votes*, describing the votes in the ballot box. This function must review these votes and return a string representing the name of the winning candidate.

**Input Format**

The locked stub code in your editor reads the following input from stdin and passes it to your function:

The first line contains an integer, *n*, denoting the size of the *votes* array.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) of strings contains a citizen's vote in the form of a candidate's name.

**Constraints**

* *1 ≤ n ≤ 104*

**Output Format**

Your function must return a string denoting the name of the winner. This is printed to stdout by the locked stub code in your editor.

**Sample Input 1**

10

Alex

Michael

Harry

Dave

Michael

Victor

Harry

Alex

Mary

Mary

**Sample Output 1**

Michael

**Explanation 1**

*votes = {"Alex", "Michael", "Harry", "Dave", "Michael", "Victor", "Harry", "Alex", "Mary", "Mary"}*

Alex, Harry, Michael, and Mary are all tied for the highest number of votes. Because Michael is alphabetically last, we return his name as the winner.

**Sample Input 2**

10

Victor

Veronica

Ryan

Dave

Maria

Maria

Farah

Farah

Ryan

Veronica

**Sample Output 2**

Veronica

**Explanation 2**

*votes = {"Victor", "Veronica", "Ryan", "Dave", "Maria", "Maria", "Farah", "Farah", "Ryan", "Veronica"}*

Veronica, Ryan, Maria, and Farah are all tied for the highest number of votes. Because Veronica is alphabetically last, we return her name as the winner.

Solution:

string electionWinner(vector < string > votes) {

map<string,int>names;

int s=votes.size();

vector<string> arr(s);

string temp;

int max = 0;

for( int i=0;i<s;i++){

names[votes[i]]++;

}

for(int i=0;i<s;i++){

if(max<names[votes[i]])

max = names[votes[i]];

}

for(int i=0;i<s;i++){

if(names[votes[i]] == max)

arr.push\_back(votes[i]);

}

int ss = arr.size();

for(int i=0;i<ss-1;i++){

for(int j=0;j<ss-1-i;j++){

if(arr[j] > arr[j+1]){

temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

return arr[ss-1];

}

1. Sherlock and GCD

Sherlock is stuck. He has an array **A**=*{a1, a2, ..., aN}.* He wants to know if there exists a subset, ***B*** *= {****a****i1,* ***a****i2, … ,* ***a****ik}* where *1 ≤ i1 < i2 < … < ik ≤ N,* of this array which has the following properties

* *B* is non-empty.
* All elements of ***B*** are relatively prime, i.e. there exists no integer ***x*** *(****x*** *> 1)* that evenly divides all elements of ***B***.  
  Note that ***x*** may or may not be an element of ***A***.

**Constraints**  
*1 ≤ T ≤ 10  
1 ≤ N ≤ 100  
1 ≤ ai ≤ 105*

1 *≤ i ≤ N*

**Input Format**  
The first line contains a single integer ***T***, the number of test cases.

Each test case consists of two lines: a single integer ***N*** on the first line representing the size of the array.In the next line there are N space separated integers  *a1, a2, ..., aN* representing the elements of the array A.Na1,a

**Output Format**  
Print  YES if there exists any such subset, and or NO, if not.1≤i≤  
5 ∀1≤i≤N

**Sample input #00**

2

3

1 2 3

2

2 4

**Sample output #00**

YES

NO

**Explanation #00**  
In first test case, *{1},{1,2}, {1,3}, {2,3}* and *{1,2,3}* are the possible subsets where no integer greater than 1 divides the elements.  
In second test case, no non-empty subset exists which satisfies the given condition.

Solution:

#include <iostream>

using namespace std;

int gcd(int x,int y) {

if(y==0) return x;

return gcd(y,x%y);

}

int main() {

int test;

cin >> test;

while(test--){

int range,i,num,g=0;

cin >> range;

for(i=0;i<range;i++){

cin >> num;

g = gcd(g,num);

}

if(g==1) cout << "YES\n";

else cout << "NO\n";

}

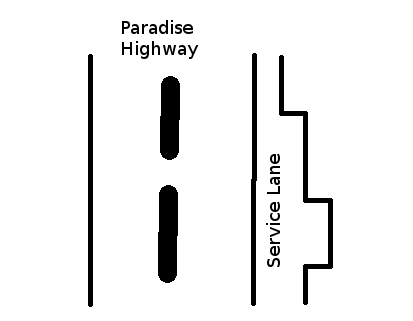
return 0;

}

1. Service Lane

Hobbes is driving his favorite vehicle on Paradise Highway. He notices that the "Check Engine" light of his vehicle is on, and he wants to service it immediately to avoid any risks. Luckily, a service lane runs parallel to the highway. The length of the highway (and the service lane) is ***N*** units. The service lane consists of ***N*** segments of unit length, and each segment can have a different width denoted by *1*, *2* or *3*. An array ***width****[]* of length ***N*** gives the widths of the different lanes, where ***width****[****k****]* represents the width of the ***k****th* segment of the service lane.

When Hobbes sees the "Check Engine" light, he is arriving at segment ***i***, and, the next exit that he needs to take to find a repair station is ***j***.  *Hobbes therefore must pass through all segments* ***i*** *to* ***j****, inclusive of both****i*** *and* ***j****.*



Hobbes has three types of vehicles - bike, car and truck, with widths *1*, *2* and *3* respectively. Given the entry and exit point of Hobbes' vehicle in the service lane, output the type of largest vehicle which can pass through the service lane (including the entry & exit segment). Assume that there is never more than *1000* segments on the path from *i* to *j* (both inclusive).

**Note**

1. If *width[k]* is 1, only the bike can pass through the *kth* segment.
2. If *width[k]* is 2, then both the bike and car can pass through the *kth* segment.
3. If *width[k]* is 3, then any of the bike, car or truck can pass through the ***k****th* segment.

**Input Format**  
The first line of input contains two integers - *N* and *T*, where *N* is the length of the freeway, and *T* is the number of test cases.

The next line has *N* space separated integers which represents the *width* array.

*T* test cases follow. Each test case contains two integers - *i* and *j*, where *i* is the index of segment through which Hobbes enters the service lane and *j* is the index of the lane segment where he exits.

**Output Format**  
For each test case, print (on a separate line) the number (*1*, *2* or *3*) that represents the largest vehicle type that can pass through the service lane.

**Note**  
*Hobbes has to pass through all segments from index i to index j (both inclusive).*

**Constraints**

* *1 ≤ T ≤1000*
* *2 ≤ N ≤ 105*
* *0 ≤ i < j < N*
* *j ≤****i*** *+ 999*
* *1 ≤ width[k] ≤ 3, where 0 <= k < N*

**Sample Input #00**

8 5

2 3 1 2 3 2 3 3

0 3

4 6

6 7

3 5

0 7

**Sample Output #00**

1

2

3

2

1

**Explanation #00**  
Below is the representation of lane.

**|HIGHWAY|Lane| -> Width**

**0: | |--| 2**

**1: | |---| 3**

**2: | |-| 1**

**3: | |--| 2**

**4: | |---| 3**

**5: | |--| 2**

**6: | |---| 3**

**7: | |---| 3**

1. (0, 3): Because *width[2]* = *1*, only the bike, represented as *1* can pass through this segment.
2. (4, 6): Here the largest vehicle that can pass through is the car, because  of the the 5th segment's width.
3. (6, 7): In this example, the vehicle enters at the 6th segment and exits at the 7th segment. Both segments allow even the truck to pass through, so *3* is the answer.
4. (3, 5): *width[3] = width[5] = 2*. While 4th segment allow the truck, the *3rd* and *5th* allow up to 2, the car width.
5. (0, 7): The bike is the only vehicle which can pass through the 2nd segment, which limits the capacity of the whole lane to 1.

Solution :

#include <stdio.h>

int main() {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/

long long int num,test;

long long int i,j,a[100000],m,n,min;

scanf("%d%d",&num,&test);

for(i=0;i<num;i++){

scanf("%d",&a[i]);

}

while(test--){

scanf("%d%d",&m,&n);

min = a[m];

for(i=m;i<=n;i++){

if(a[i]<min){

min = a[i];

}

}

printf("%d\n",min);

}

return 0;

}

iNautix Test – 12

1. Calculate Factorial

Complete the function ***factorial*** in your editor. It has *1* parameter:

1. An integer, *n*.

It must return the factorial of *n*.

**Input Format**  
The locked stub code in your editor reads the following input from stdin and passes it to your function:

The first line contains a single integer *N.*

**Constraints**

* 1 ≤ N ≤ 15

**Output Format**  
Your function must return the factorial of *n*. This is printed to stdout by the locked stub code in your editor.

**Sample Input 1**

2

**Sample Output 1**

2

**Sample Input 2**

3

**Sample Output 2**

6

**Sample Input 3**

4

**Sample Output 3**

24

**Explanation 3**

Factorial of 4 is 1\*2\*3\*4 = 24.

Solution:

int factorial(int n) {

if(n==1){

return 1;

}

else{

return n\*factorial(n-1);

}

}

1. Cup Cake Feast

Sarah loves going to her favorite bakery, Zillycakes, with *n* dollars to buy cupcakes. Each cupcake has a flat cost of *c* dollars, and the store has a promotion where they give you *1* free cupcake for every *m* cupcake wrappers you trade in.

For example, if *m = 2* and Sarah has *n = 4* dollars that she uses to buy *4* cupcakes at *c = 1* dollar apiece, she can trade in the *4* wrappers to buy *2* more cupcakes. Now she has *2* more wrappers that she can trade in for *1* more cupcake. Because she only has *1* wrapper left at this point and *1 < m*, she was only able to eat a total of *7* cupcakes.

Complete the *maximumCupcakes* function in your editor. It has *1* parameter: an array of strings named *trips*. Each string in *trips* contains the following three space-separated values:

1. An integer, *n*, denoting the amount of money (in dollars) Sarah can spend during that trip to Zillycakes.
2. An integer, *c*, denoting the cost of a cupcake in dollars.
3. An integer, *m*, denoting the number of wrappers that can be traded in for one new cupcake.

For each trip to Zillycakes in *trips*, your function must print an integer denoting the maximum number of cupcakes Sarah can eat on a new line.

**Input Format**

The locked stub code in your editor reads the following input from stdin and passes it to your function:

The first line contains an integer, *t*, denoting the number of trips Sarah makes to the store (i.e., the size of the *trips* array).

Each line *i* of the *t* subsequent lines contains three space-separated integers describing the respective *n*, *c*, and *m* values for one of Sarah's trips to the store. This corresponds to element *i* in *trips*.

**Constraints**

* *1 ≤ t ≤ 103*
* *2 ≤ n ≤ 105*
* *1 ≤ c ≤ n*
* *2 ≤ m ≤ n*

**Output Format**

For each trip in *trips*, your function must print a single integer on a new line denoting the maximum number of cupcakes Sarah can eat during that trip.

**Sample Input 1**

The following argument is passed to your function: *trips = ["10 2 5", "12 4 4", "6 2 2"]*

**Sample Output 1**

6

3

5

**Explanation 1**

Sarah makes the following three trips to Zillycakes:

1. She spends *10* dollars on *5* cupcakes at *2* dollars apiece. She then eats them and exchanges all *5* wrappers to get *1* more cupcakes. We print the total number of cupcakes she ate, which is *6*.
2. She spends *12* dollars on *3* cupcakes at *4* dollars apiece; however, she needs *4* wrappers to trade for her next cupcake. Because she only has *3* wrappers, she cannot purchase or trade for any more cupcakes. We print the total number of cupcakes she ate, which is *3*.
3. She spends *6* dollars on *3* cupcakes at *2* dollars apiece. She then exchanges *2* of the *3* wrappers for *1* additional cupcake. Next, she combines her third leftover cupcake wrapper from her initial purchase with the wrapper from her trade-in to do a second trade-in for *1* more cupcake. At this point she has *1* wrapper left, which is not enough to perform another trade-in. We print the total number of cupcakes she ate, which is *5*.

**Sample Input 2**

The following argument is passed to your function: *trips = ["8 4 2", "7 2 3"]*

**Sample Output 2**

3

4

**Explanation 2**

Sarah makes the following three trips to Zillycakes

1. She spends *8* dollars on *2* cupcakes at *4* dollars apiece. She then eats them and exchanges both of her wrappers for *1* more cupcake. At this point she has *1* wrapper left, which is not enough to perform another trade-in. We print the total number of cupcakes she ate, which is *3*.
2. She spends *6* of her *7* dollars on *3* cupcakes at *2* dollars apiece. She then eats them and exchanges all three of her wrappers for *1* more cupcake. At this point she has *1* wrapper left, which is not enough to perform another trade-in. We print the total number of cupcakes she ate, which is *4*.

Solution:

void maximumCupcakes(int trips\_size, char\*\* trips) {

int i=0,j,k=0,n,c,m,r=0,wraps,rem,flag=0;

char temp[10];

for(i=0;i<trips\_size;i++){

flag=0;

k=0;

for(j=0;trips[i][j]!='\0';j++){

if(trips[i][j]!=' '){

temp[k]=trips[i][j];

k++;

}

else{

temp[k]='\0';

if (flag==0){

n=atoi(temp);

k=0;

flag=1;

}

else{

c=atoi(temp);

k=0;flag=0;

}

}

}

temp[k]='\0';

m=atoi(temp);

r=n/c;

wraps=r;

while(wraps>=m){

r=r+(wraps/m);

rem=wraps%m;

wraps=(wraps/m)+rem;

}

printf("%d\n",r);

}

}

iNautix Test 13:

1. Filling Jars

Calvin has N empty candy jars, numbered from 1 to N, with infinite capacity. He performs M operations. Each operation is described by 3 integers a, b and k, where, a and b are indices of the jars, and k is the number of jelly beans to be added inside each jar whose index lies between a and b (inclusive of both a and b). Your task is to compute the average number of jelly beans per jar after M operations?

**Input Format**  
The first line contains two integers N and M separated by a single space.  
M lines follow. Each of the M lines contain the three integers described above, a, b and k, separated by a single space.

**Output Format**  
A single line containing the average number of candies across N jars, rounded down to the nearest integer.

(Rounded down means finding the greatest integer which is less than or equal to given number. Eg, 13.65 and 13.23 is rounded down to 13, while 12.98 is rounded down to 12.)

**Constraints**  
3 ≤ N ≤ 1071 ≤ M ≤ 1051 ≤ a ≤b ≤ N0 ≤ k ≤ 106

**Sample Input #00**

5 3

1 2 100

2 5 100

3 4 100

**Sample Output #00**

160

**Explanation**  
Initially each of the jar contains 0 candies

0 0 0 0 0

First operation

100 100 0 0 0

Second operation

100 200 100 100 100

Third operation

100 200 200 200 100

Total = 800, Average = 800/5 = 160

Solution:

#include <iostream>

#include<vector>

#include<math.h>

#include<array>

using namespace std;

int main() {

long n0,m0;

long answer0=0,t0;

cin>>n0>>m0;

t0=m0;

while(t0--){

long p,q,r;

cin>>p>>q>>r;

answer0=answer0+(abs(p-q)+1)\*r;

}

answer0=floor(answer0/n0);

cout<<answer0<<endl;

return 0;

}

1. Cut the Sticks:

Given an array, *lengths*, of *N* stick lengths (where each length is a positive integer), a *cut operation* reduces the length of each stick in the array by the length of the array's shortest stick. A stick can only be cut if it has a length *≥ 1*.

Complete the *cutSticks* function in your editor, It has *1* parameters:

1. An integer array of stick lengths, *lengths*(as well as the size of the array where required by the language).

Your function must perform *cut operations* on *lengths* until every stick length is reduced to *0* and return an integer array, where ith element of the array denotes the individual sticks cut during the ithoperation.

**Input Format**

The following input from stdin is handled for you by the locked stub code in your editor:

The first line contains an integer, *N*, the length of *lengths*. The *N* subsequent lines describe the elements in *lengths*.

**Constraints**

* *1 ≤ N ≤ 103*
* *1 ≤ lengths[i] ≤ 103*, where *0 ≤ i ≤ N−1*

**Output Format**

Your function must return an integer array, where ith element of the array denotes the individual sticks cut during the ith operation. This is printed to stdout by the locked stub code in your editor.

**Sample Input 1**

6

5

4

4

2

2

8

**Sample Output 1**

6

4

2

1

**Sample Input 2**

8

1

2

3

4

3

3

2

1

**Sample Output 2**

8

6

4

1

**Explanation**

*Sample Case 1:*

*|lengths| = 6, lengths = {5, 4, 4, 2, 2, 8}*

Cut Operation 1: The smallest length in *lengths* is *2*, so the cut length for this cut operation is *2*. After cutting (reducing) each stick in *lengths* by *2*, *lengths1 = {3, 2, 2, 0, 0, 6}* and we print *6* (the number of sticks cut during the cut operation) on a new line.

Cut Operation 2: The smallest length in *lengths1 = {3, 2, 2, 0, 0, 6}* is *2*, so the cut length for this cut operation is *2*. After cutting (reducing) each stick in *lengths1* by *2*, *lengths2 = {1, 0, 0, 0, 0, 4}* and we print *4* (the number of sticks cut during the cut operation) on a new line.

Cut Operation 3: The smallest length in *lengths2 = {1, 0, 0, 0, 0, 4}* is *1*, so the cut length for this cut operation is *1*. After cutting (reducing) each stick in *lengths2* by *1*, *lengths3 = {0, 0, 0, 0, 0, 3}* and we print *2* (the number of sticks cut during the cut operation) on a new line.

Cut Operation 3: The smallest length in *lengths3 = {0, 0, 0, 0, 0, 3}* is *3*, so the cut length for this cut operation is *3*. After cutting (reducing) each stick in *lengths3* by *3*, *lengths4 = {0, 0, 0, 0, 0, 0}* and we print *1* (the number of sticks cut during the cut operation) on a new line.

At this point, there are no more sticks to be cut and we cease performing cut operations.

*Sample Case 2:*

lengths cut length count cuts

1 2 3 4 3 3 2 1 1 8

\_ 1 2 3 2 2 1 \_ 1 6

\_ \_ 1 2 1 1 \_ \_ 1 4

\_ \_ \_ 1 \_ \_ \_ \_ 1 1

\_ \_ \_ \_ \_ \_ \_ \_ DONE DONE

Solution:

static int[] cutSticks(int[] lengths) {

int mini=10000,count=0,ci=0;

List<Integer> counts=new ArrayList<Integer>();

while(true)

{

for(int i=0;i<lengths.length;i++)

{

if(lengths[i]<mini && lengths[i]!=0)

mini=lengths[i];

}

for(int j=0;j<lengths.length;j++)

{

if(lengths[j]!=0)

{

lengths[j]=lengths[j]-mini;

count++;

}

}

mini=10000;

counts.add(count);

count=0;

for(int k:lengths)

{

if(k==0)

ci++;

}

System.out.println(ci);

if(ci==lengths.length)

break;

ci=0;

}

int b[]=new int[counts.size()];

for(int l=0;l<counts.size();l++)

{

b[l]=counts.get(l);

}

return b;

}

iNautix Test -14

1. Haloween Party

Lenny is attending a Halloween party with his girlfriend Penny. At the party, Penny spots a giant rectangular chocolate bar. If the chocolate can be served as only 1 x 1 sized pieces, and Lenny can cut the chocolate bar exactly **K** times, what is the maximum number of chocolate pieces Lenny can cut and give Penny?

**Note**  
Chocolate must be served in size of 1 x 1 size pieces.  
Leonard can't move any of the pieces, including placing any piece on top of other.

Complete the max\_Chocolates function in your editor. It has 1 parameter:

  1. An integer array arr, where ai denotes the ith test case.

It must return an array, where ith element in the array denotes the result for the ith test case.

**Input Format**  
The locked stub code in your editor reads the following input from stdin and passes it to your function:

The first line contains an integer ***T***, the number of test cases. **T** lines follow.Each line contains an integer **K.**

**Constraints**  
1 ≤ T ≤ 10  
2 ≤ K ≤ 107

**Output Format**  
Your function must return an array, where ith element in the array denotes the result for the ith test case.

This is printed to stdout by the locked stub code in your editor.

**Sample Input 1**

4

5

6

7

8

**Sample Output 1**

6

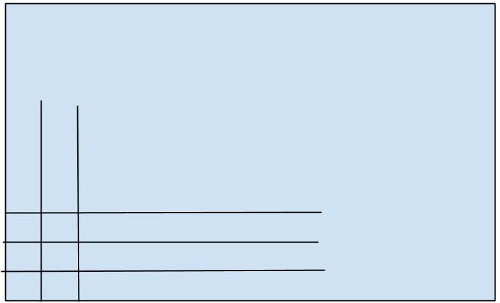
9

12

16

**Explanation**

Sample Case 1: The explanation below is for the first two test-cases. The rest of them follow a similar logic.

For the first test-case where K = 5, You need 3 Horizontal and 2 vertical cuts.  


For the second test-case where K = 6, You need 3 Horizontal and 3 vertical cuts.

Solution:

static int[] max\_Chocolates(int[] arr) {

int testcases=arr.length;

int[] max=new int[testcases];

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

{

int temp=arr[i];

for(int k=1;k<temp;k++)

{

int j=temp-k;

int tempo=(int)k\*j;

if(tempo>max[i])

{

max[i]=tempo;

}

}

/\*int tem1=arr[i];

int tem2=arr[i]/2;

int tem3=tem1-tem2;

int temp4=tem2\*tem3;

System.out.println(temp4);

max[i]=temp4;\*/

}

return max;

}

1. Grid Game

Julia is playing a game on an infinite 2-dimensional grid with the bottom left cell referenced as *(1, 1)*. All the cells contain a value of zero initially. The game consists of *n* steps. In each step, Julia is given two integers *a* and *b*. The value of each of the cells in the coordinate *(u, v)* satisfying *1 ≤ u ≤ a* and *1 ≤ v ≤ b*, is increased by *1*. After *n* such steps, if *x* is the largest number in any cell on the board, how many instances of *x*are there on the board?

Complete the function *countX* that has one parameter, a string array, *steps*, denoting the values of a and b for each of steps of the game. The function should return the total number of occurrences of greatest integer x in the grid after n steps.

**Input Format**  
The first line of input contains a single integer *n*. Each of the next n lines contains two space-separated integers, *a* and *b*.

**Constraints**

* *1 ≤****n****≤ 100*
* *1 ≤****a, b****≤ 10*

**Output Format**

The function should return the total number of occurrences of greatest integer x in the grid after n steps.

**Sample Input**

3

2 3

3 7

4 1

**Sample Output**

2

**Explanation**

Assume that the following board corresponds to cells (i, j) where 1 ≤ i ≤ 4 and 1 ≤ j ≤ 7.

At the beginning the board is in the following state:

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

After the first step we obtain:

0 0 0 0 0 0 0

0 0 0 0 0 0 0

1 1 1 0 0 0 0

1 1 1 0 0 0 0

After the second step we have:

0 0 0 0 0 0 0

1 1 1 1 1 1 1

2 2 2 1 1 1 1

2 2 2 1 1 1 1

Finally, after the last step the board will look like this:

1 0 0 0 0 0 0

2 1 1 1 1 1 1

3 2 2 1 1 1 1

3 2 2 1 1 1 1

So, the maximum number is 3 and there are exactly two cells which contain 3. Hence the answer is 2.

Solution:

static long countX(String[] steps) {

int length=steps.length;

int[][] matrix=new int[length][2];

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

{

String[] splitter=steps[i].split(" ");

matrix[i][0]=Integer.parseInt(splitter[0]);

matrix[i][1]=Integer.parseInt(splitter[1]);

}

int row\_max=0;

int column\_max=0;

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

{

if(matrix[i][0]>row\_max)

{

row\_max=matrix[i][0];

}

if(matrix[i][1]>column\_max)

{

column\_max=matrix[i][1];

}

}

int[][] new\_matrix=new int[row\_max][column\_max];

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

{

int temp\_row=matrix[i][0];

int temp\_column=matrix[i][1];

int row\_counter=temp\_row;

int column\_counter=temp\_column;

for(int k=(row\_max-1);row\_counter>0;k--)

{

for(int j=0;j<column\_counter;j++)

{

new\_matrix[k][j]=new\_matrix[k][j]+1;

}

row\_counter=row\_counter-1;

}

}

long ans\_counter=0;

for(int i=0;i<row\_max;i++)

{

for(int j=0;j<column\_max;j++)

{

if(new\_matrix[i][j]==length)

{

ans\_counter=ans\_counter+1;

}

}

}

return ans\_counter;

}