SDE - MODULE 2 - PROBLEM SOLVING 9 - HANDSON  
  
1.Given an array arr of positive integers sorted in a strictly increasing order, and an integer k. Return the kth positive integer that is missing from this array  
  
Program : (KthMissing)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** KthMissing {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[] = **new** **int**[n];

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

a[i] = sc.nextInt();

}

System.***out***.println("Enter k :");

**int** k = sc.nextInt();

**int** i = 0, current = 1, c= 0;

**while**(i < k){

**if** (i < n && a[i] == current) {

i++;

}

**else** {

c++;

**if** (c == k) {

System.***out***.println(current);

**break**;

}

}

current++;

}

}

}

Output :

Enter n:

5

Enter array :

2 3 4 7 11

Enter k :

5

9

2.Given a non-negative integer x, return the square root of x rounded down to the nearest integer. The returned integer should be non-negative as well.  
  
Program : (Sqrt)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** Sqrt {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

**int** i = 1;

**while**(i <= n) {

**int** ans = i \* i;

**if**(ans == n) {

System.***out***.println(i);

**break**;

}

**else** **if**(ans > n) {

i--;

System.***out***.println(i);

**break**;

}

**else**

i++;

}

}

}

Output :   
Enter n:

17

4

3.Given an array Arr of N positive integers. Your task is to find the elements whose value is equal to that of its index value ( Consider 1-based indexing ).  
  
Program : (ValueEqualIndex)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** ValueEqualIndex {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[] = **new** **int**[n+1];

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

a[i] = sc.nextInt();

}

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

**if**(a[i] == i) {

System.***out***.println(a[i]);

}

}

}

}

Output :   
Enter n:

5

Enter array :

15 2 45 12 7

2

4.Given two sorted arrays of distinct elements. There is only 1 difference between the arrays. The first array has one element extra added in between. Find the index of the extra element.  
  
Program : (IndexOfExtra)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** IndexOfExtra {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[] = **new** **int**[n];

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

a[i] = sc.nextInt();

}

System.***out***.println("Enter array :");

**int** b[] = **new** **int**[n-1];

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

b[i] = sc.nextInt();

}

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

**if**(a[i] != b[i]) {

System.***out***.println(i);

**break**;

}

}

}

}

Output :   
Enter n:

7

Enter array :

2 4 6 8 9 10 12

Enter array :

2 4 6 8 10 12

4

5.Given a sorted array of distinct integers 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 must write an algorithm with O(log n) runtime complexity.  
  
Program : (InsertPosition)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** InsertPosition {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[] = **new** **int**[n];

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

a[i] = sc.nextInt();

}

System.***out***.println("enter the target : ");

**int** t = sc.nextInt();

**int** l = 0, h =n-1 , mid = 0;

**boolean** flag = **false**;

**while** (l <= h) {

mid = (l+h)/2;

**if**(a[mid] == t) {

System.***out***.println(mid);

flag = **true**;

**break**;

}

**else** **if**(a[mid] < t) {

l = mid + 1;

}

**else**

h = mid - 1;

}

**if**(flag == **false**)

System.***out***.println(l);

}

}

Output :   
Enter n:

4

Enter array :

2 3 5 6

enter the target :

1

0

6.: Given an array arr of integers, check if there exist two indices i and j such that : i != j 0 <= i, j < arr.length arr[i] == 2 \* arr[j]  
  
Program : (DoubleExist)

**package** com.handson;

**import** java.util.Scanner;

**public** **class** DoubleExist {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[] = **new** **int**[n];

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

a[i] = sc.nextInt();

}

**boolean** flag = **false**;

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

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

**if**(a[i] == 2\*a[j]) {

flag = **true**;

**break**;

}

}

}

**if**(flag)

System.***out***.println("true");

**else**

System.***out***.println("false");

}

}

Output :   
Enter n:

4

Enter array :

3 1 7 11

false

7.You are given a sorted array ‘arr’ of ‘n’ numbers such that every number occurred twice in the array except one, which appears only once. Return the number that appears once.  
  
Program : (SingleNumber)

**package** com.handson;

**import** java.util.Scanner;

**public** **class** SingleNumber {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[] = **new** **int**[n];

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

a[i] = sc.nextInt();

}

**int** ans = a[0];

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

ans = ans ^ a[i];

}

System.***out***.println(ans);

}

}

Output :

Enter n:

5

Enter array :

1 1 3 5 5

3

8.You are given a sorted array 'ARR' and a number 'X'. Your task is to count the number of occurrences of 'X' in 'ARR'.   
  
Program : (Frequency)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** Frequency {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[] = **new** **int**[n];

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

a[i] = sc.nextInt();

}

System.***out***.println("enter x :");

**int** x = sc.nextInt();

**int** c = 0;

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

**if**(a[i] == x)

c++;

}

System.***out***.println(c);

}

}

Output :   
Enter n:

7

Enter array :

1 1 2 2 2 2 3

enter x :

2

4

9. Given a sorted array containing only 0s and 1s, find the transition point, i.e., the first index where 1 was observed, and before that, only 0 was observed.   
  
Program : (Transition)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** Transition {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[] = **new** **int**[n];

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

a[i] = sc.nextInt();

}

**int** x = -1;

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

**if**(a[i] == 1) {

System.***out***.println(i);

x = 1;

**break**;

}

}

**if**(x == -1)

System.***out***.println(x);

}

}

Output :   
Enter n:

4

Enter array :

0 0 0 0

-1

10.Given two integer arrays nums1 and nums2, sorted in non-decreasing order, return the minimum integer common to both arrays. If there is no common integer amongst nums1 and nums2, return -1.  
  
Program : (CommonValue)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** CommonValue {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[] = **new** **int**[n];

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

a[i] = sc.nextInt();

}

System.***out***.println("Enter n2: ");

**int** n2 = sc.nextInt();

System.***out***.println("Enter array :");

**int** b[] = **new** **int**[n2];

**for** (**int** i = 0; i < n2; i++) {

b[i] = sc.nextInt();

}

**int** x = -1;

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

**for**(**int** j = 0; j < n2; j++) {

**if**(a[i] == b[j]) {

System.***out***.println(a[i]);

x = 1;

**break**;

}

}

}

**if**(x == -1)

System.***out***.println(x);

}

}

Output :

Enter n:

3

Enter array :

1 2 3

Enter n2:

2

Enter array :

2 4

2

11. Given a matrix mat[][] of size N x M, where every row and column is sorted in increasing order, and a number X is given. The task is to find whether element X is present in the matrix or not  
  
Program : (SearchMatrix)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** SearchMatrix {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter m: ");

**int** m = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[][] = **new** **int**[n][m];

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

**for**(**int** j = 0; j < m; j++) {

a[i][j] = sc.nextInt();

}

}

System.***out***.println("Enter x :");

**int** x = sc.nextInt();

**int** ans = 0;

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

**for**(**int** j = 0; j < m; j++) {

**if**( x== a[i][j]) {

ans = 1;

}

}

}

System.***out***.println(ans);

}

}

Output :   
Enter n:

1

Enter m:

6

Enter array :

18 21 27 38 55 67

Enter x :

55

1

12.You are given an infinite array consisting of only ones and zeroes, in sorted order. You have to find the index of the first occurrence of 1.   
  
Program : (Infinite)  
package com.handson;

import java.util.Scanner;

public class InfiniteSorted {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter n: ");

int n = sc.nextInt();

System.out.println("Enter array :");

int a[] = new int[n];

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

a[i] = sc.nextInt();

}

int x = -1;

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

if(a[i] == 1) {

System.out.println(i);

x = 1;

break;

}

}

if(x == -1)

System.out.println(x);

}

}

Output :   
Enter n:

12

Enter array :

0 0 0 0 0 0 0 0 0 1 1 1

9

13.You have been given an array of ‘N’ distinct integers which is sorted in ascending order and then rotated to the left by an unknown which you don’t know beforehand. For a given integer ‘X’, your task is to find the index of ‘X’ in the given array if it exists. Please note that the sorted array A : [2, 3, 6, 8, 9, 11, 15] might become [6, 8, 9, 11, 15, 2, 3] after rotating it twice to the left.  
  
Program : (RotatedSort)  
package com.handson;

import java.util.Scanner;

public class RotatedSort {

public static int search(int[] arr, int t) {

int l = 0;

int r = arr.length - 1;

while (l <= r) {

int mid = l + (r - l) / 2;

if (arr[mid] == t) {

return mid;

}

if (arr[l] <= arr[mid]) {

if (arr[l] <= t && t < arr[mid]) {

r = mid - 1;

} else {

l = mid + 1;

}

} else {

if (arr[mid] < t && t <= arr[r]) {

l = mid + 1;

} else {

r = mid - 1;

}

}

}

return -1;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the n : ");

int n = sc.nextInt();

System.out.println("Enter the array : ");

int a[] = new int[n];

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

a[i] = sc.nextInt();

}

System.out.println("Enter the target : ");

int t = sc.nextInt();

int index = search(a, t);

if (index != -1) {

System.out.println(index);

}

else {

System.out.println(" not found ");

}

}

}

Output :   
Enter the n :

7

Enter the array :

6 8 9 11 15 2 3

Enter the target :

6

0

14. You are given an array nums of nonnegative integers. nums is considered special if there exists a number x such that there are exactly x numbers in nums that are greater than or equal to x. Notice that x does not have to be an element in nums. Return x if the array is special, otherwise, return -1. It can be proven that if nums is special, the value for x is unique.   
  
Program : (SpecialArray)  
package com.handson;

import java.util.Scanner;

public class SpecialArray {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the n : ");

int n = sc.nextInt();

System.out.println("Enter the array : ");

int a[] = new int[n];

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

a[i] = sc.nextInt();

}

int ans = -1;

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

int cnt = 0;

for (int j : a) {

if (j >= i) {

++cnt;

}

}

if (cnt == i) {

ans = i;

break;

}

}

System.out.println(ans);

}

}

Output :   
Enter the n :

5

Enter the array :

0 4 3 0 4

3

16. Given a m x n matrix grid which is sorted in non-increasing order both row-wise and column-wise, return the number of negative numbers in grid.  
  
Program : (NegativeInMatrix)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** NegativeInMatrix {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter m: ");

**int** m = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[][] = **new** **int**[n][m];

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

**for**(**int** j = 0; j < m; j++) {

a[i][j] = sc.nextInt();

}

}

**int** c = 0;

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

**for**(**int** j = 0; j < m; j++) {

**if**(a[i][j] < 0)

c++;

}

}

System.***out***.println(c);

}

}

Output :   
Enter n:

4

Enter m:

4

Enter array :

4 3 2 -1

3 2 1 -1

1 1 -1 -2

-1 -1 -2 -3

8

17.Given two arrays: a1[0..n-1] of size n and a2[0..m-1] of size m, where both arrays may contain duplicate elements. The task is to determine whether array a2 is a subset of array a1. It's important to note that both arrays can be sorted or unsorted. Additionally, each occurrence of a duplicate element within an array is considered as a separate element of the set.   
  
Program : (Subset)  
**package** com.handson;

**import** java.util.\*;

**public** **class** Subset {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[] = **new** **int**[n];

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

a[i] = sc.nextInt();

}

System.***out***.println("Enter n2: ");

**int** n2 = sc.nextInt();

System.***out***.println("Enter array :");

**int** b[] = **new** **int**[n2];

**for** (**int** i = 0; i < n2; i++) {

b[i] = sc.nextInt();

}

Map<Integer,Integer> count = **new** HashMap<Integer,Integer>();

**for**(**int** i : a) {

count.put(0,count.getOrDefault(0, i)+1);

}

**int** c = 0;

**boolean** flag = **true**;

**for**(**int** i : b) {

**if** (!count.containsKey(i) || count.get(i) == 0) {

flag = **false**;

**break**;

} **else** {

count.put(i, count.get(i) - 1);

}

}

**if**(flag)

System.***out***.println("No");

**else**

System.***out***.println("Yes");

}

}

Output :   
Enter n:

7

Enter array :

1 2 3 4 4 5 6

Enter n2:

3

Enter array :

1 2 4

Yes

18. Alice and Bob have a different total number of candies. You are given two integer arrays aliceSizes and bobSizes where aliceSizes[i] is the number of candies of the ith box of candy that Alice has and bobSizes[j] is the number of candies of the jth box of candy that Bob has. Since they are friends, they would like to exchange one candy box each so that after the exchange, they both have the same total amount of candy. The total amount of candy a person has is the sum of the number of candies in each box they have. Return an integer array answer where answer[0] is the number of candies in the box that Alice must exchange, and answer[1] is the number of candies in the box that Bob must exchange. If there are multiple answers, you may return any one of them. It is guaranteed that at least one answer exists.   
  
Program : (FairCandy)

**package** com.handson;

**import** java.util.Scanner;

**public** **class** FairCandy {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n1:");

**int** n1 = sc.nextInt();

**int** alice[] = **new** **int**[n1];

System.***out***.println("Enter array:");

**for** (**int** i = 0; i < n1; i++){

alice[i] = sc.nextInt();

}

System.***out***.println("Enter n2:");

**int** n2 = sc.nextInt();

**int** bob[] = **new** **int**[n2];

System.***out***.println("Enter array:");

**for** (**int** i = 0; i < n2; i++){

bob[i] = sc.nextInt();

}

**int** c1 = 0, c2 = 0;

**for** (**int** i : alice)

c1 += i;

**for** (**int** i : bob)

c2 += i;

**for** (**int** i = 0; i < n1; i++) {

**for** (**int** j = 0; j < n2; j++) {

**if** ((c1 - alice[i] + bob[j]) == (c2 - bob[j] + alice[i])) {

System.***out***.println(alice[i] + " " + bob[j]);

**break**;

}

}

}

}

}

Output :   
Enter n1:

1

Enter array:

2

Enter n2:

2

Enter array:

1 3

2 3

19.Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.If target is not found in the array, return [-1, -1].   
  
Program : (FirstLast)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** FirstLast {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

System.***out***.println("Enter array :");

**int** a[] = **new** **int**[n];

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

a[i] = sc.nextInt();

}

System.***out***.println("enter target : ");

**int** t = sc.nextInt();

**int** ans[] = **new** **int**[2] ;

**int** x = 0;

**int** low = 0, high = n-1;

**while**(low <= high) {

**int** mid = (low + high) / 2;

**if**(a[mid] == t) {

ans[x] = mid;

**if**(mid - 1 >= 0 && a[mid - 1] == t)

high = mid - 1 ;

**else**

low = mid + 1;

x++;

}

**else** **if**(a[mid] < t)

low = mid+1;

**else**

high = mid-1;

}

System.***out***.print(ans[1] + "," + ans[0]);

}

}

Output :   
Enter n:

6

Enter array :

5 7 7 8 8 10

enter target :

8

3,4

20.Given an integer array nums, return the length of the longest strictly increasing subsequence   
  
Program : (IncreasingSubsequence)  
package com.handson;

import java.util.Scanner;

public class IncreasingSubsequence {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the n : ");

int n = sc.nextInt();

System.out.println("Enter the array : ");

int a[] = new int[n];

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

a[i] = sc.nextInt();

}

int[] dp = new int[n];

int len=0;

int max=0;

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

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

if(a[i]>a[j]){

if(dp[j]>len)

len=dp[j];

}

}

dp[i]=len+1;

len=0;

if(dp[i]>max){

max=dp[i];

}

}

System.out.println(max);

}

}

Output :   
Enter the n :

6

Enter the array :

0 1 0 3 2 3

4

21. You are given a 0-indexed array of positive integers nums. A subarray of nums is called incremovable if nums becomes strictly increasing on removing the subarray. For example, the subarray [3, 4] is an incremovable subarray of [5, 3, 4, 6, 7] because removing this subarray changes the array [5, 3, 4, 6, 7] to [5, 6, 7] which is strictly increasing. Return the total number of incremovable subarrays of nums. Note that an empty array is considered strictly increasing.  
  
Program : (Incremovable)  
package com.handson;

import java.util.Scanner;

public class Incremovable {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the n : ");

int n = sc.nextInt();

System.out.println("Enter the array : ");

int a[] = new int[n];

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

a[i] = sc.nextInt();

}

int count = 0;

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

int j = i + 1;

while (j < n && a[j] > a[j - 1]) {

j++;

}

int len = j - i;

count += (len \* (len + 1)) / 2;

i = j;

}

System.out.println(count);

}

}

Output :   
Enter the n :

4

Enter the array :

6 5 7 8

7

22.A row-sorted binary matrix means that all elements are 0 or 1 and each row of the matrix is sorted in non-decreasing order.  
  
Program : (LeftMost)  
package com.handson;

import java.util.Scanner;

public class LeftMost {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the m and n: ");

int m = sc.nextInt();

int n = sc.nextInt();

System.out.println("Enter the array : ");

int a[][] = new int[m][n];

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

for(int j =0; j < n; j++) {

a[i][j] = sc.nextInt();

}

}

int row = 0;

int col = n - 1;

int ans = -1;

while (row < m && col >= 0) {

if (a[row][col] == 1) {

ans = col;

col--;

}

else {

row++;

}

}

System.out.println(ans);

}

}

Output :   
Enter the m and n:

2

2

Enter the array :

0 0

0 1

1

23.Given an array of integers citations where citations[i] is the number of citations a researcher received for their ith paper and citations is sorted in ascending order, return the researcher's h-index.The h-index is defined as the maximum value of h such that the given researcher has published at least h papers that have each been cited at least h times.   
  
Program : (Citations)  
package com.handson;

import java.util.Scanner;

public class Citations {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the n : ");

int n = sc.nextInt();

System.out.println("Enter the array : ");

int a[] = new int[n];

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

a[i] = sc.nextInt();

}

int l = 0,r = n-1,h = 0;

while(l <= r){

int mid = (l + r)/2;

if(n-mid <= a[mid]){

h = n - mid;

r = mid-1;

}

else{

l = mid+1;

}

}

System.out.println(h);

}

}

Output :   
Enter the n :

5

Enter the array :

0 1 3 5 6

3

24.Koko can decide her bananas-per-hour eating speed of k. Each hour, she chooses some pile of bananas and eats k bananas from that pile. If the pile has less than k bananas, she eats all of them instead and will not eat any more bananas during this hour. Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return. Return the minimum integer k such that she can eat all the bananas within h hours.  
  
Program : (KokoBanana)

package com.handson;

import java.util.Scanner;

public class KokoBanana {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the n : ");

int n = sc.nextInt();

System.out.println("Enter the array : ");

int a[] = new int[n];

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

a[i] = sc.nextInt();

}

System.out.println("enter hours :");

int h = sc.nextInt();

int t = 0;

for (int i : a) {

t += i;

}

int speed = (int) Math.ceil((double) t / h);

System.out.println(speed);

}

}

Output :   
Enter the n :

4

Enter the array :

3 6 7 11

enter hours :

8

4

25. A conveyor belt has packages that must be shipped from one port to another within days days. The ith package on the conveyor belt has a weight of weights[i]. Each day, we load the ship with packages on the conveyor belt (in the order given by weights). We may not load more weight than the maximum weight capacity of the ship. Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within days days

Program : ()  
Output :   
  
26.You are given an integer array nums where the ith bag contains nums[i] balls. You are also given an integer maxOperations. You can perform the following operation at most maxOperations times: Take any bag of balls and divide it into two new bags with a positive number of balls.   
  
Program : (LimitOfBall)  
package com.handson;

import java.util.Scanner;

public class LimitOfBall {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the n : ");

int n = sc.nextInt();

System.out.println("Enter the array : ");

int a[] = new int[n];

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

a[i] = sc.nextInt();

}

System.out.println("enter maxoperation :");

int m = sc.nextInt();

int min = 0;

int max = 1000000000;

while (min + 1 < max) {

int mid = (max - min)/2 + min;

int count = 0;

for (int i: a) {

count += (i-1) / mid;

}

if (count <= m) {

max = mid;

} else {

min = mid;

}

}

System.out.println(max);

}

}

Output :   
Enter the n :

4

Enter the array :

2 4 8 2

enter maxoperation :

4

2

27.You are given an integer n indicating there are n specialty retail stores. There are m product types of varying amounts, which are given as a 0-indexed integer array quantities, where quantities[i] represents the number of products of the ith product type.

Program : (Minimized)  
package com.handson;

import java.util.Scanner;

public class Minimized {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the n : ");

int n = sc.nextInt();

System.out.println("Enter the array : ");

int a[] = new int[n];

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

a[i] = sc.nextInt();

}

System.out.println("enter n stores :");

int m = sc.nextInt();

int low = 1 ;

int high = Integer.MIN\_VALUE ;

int result = 0 ;

for (int i : a)

high = Math.max(high, i) ;

while (low <= high) {

int mid = low + (high-low)/2 ;

int stores = 0 ;

for (int j : a) {

stores += j/mid ;

if (j % mid > 0)

stores++ ;

}

if (stores <= m) {

result = mid ;

high = mid-1 ;

}

else low = mid+1 ;

}

System.out.println(result);

}

}

Output :

Enter the n :

2

Enter the array :

11 6

enter n stores :

6

3

28. Given a m x n matrix mat and an integer threshold, return the maximum side-length of a square with a sum less than or equal to threshold or return 0 if there is no such square.

Program : (ThresholdSquare)  
package com.handson;

import java.util.\*;

public class ThresholdSquare {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of rows:");

int n = scanner.nextInt();

System.out.println("Enter the number of columns:");

int m = scanner.nextInt();

int[][] mat = new int[n][m];

System.out.println("Enter the matrix elements :");

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

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

mat[i][j] = scanner.nextInt();

}

}

System.out.println("Enter the threshold:");

int t = scanner.nextInt();

int ans = 0;

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

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

int s = 1, e = Math.min(n, m);

while (s <= e) {

int mid = s + (e - s) / 2;

if (mid + i > n || m < mid + j) {

e = mid - 1;

} else {

int sum = 0;

for (int x = i; x < i + mid; x++) {

for (int y = j; y < j + mid; y++) {

sum += mat[x][y];

}

}

if (sum <= t) {

s = mid + 1;

if (mid > ans) {

ans = mid;

}

} else {

e = mid - 1;

}

}

}

}

}

System.out.println(ans);

}

}

Output :   
Enter the number of rows:

5

Enter the number of columns:

5

Enter the matrix elements :

2 2 2 2 2

2 2 2 2 2

2 2 2 2 2

2 2 2 2 2

2 2 2 2 2

Enter the threshold:

1

0

29.Kavin owns an electronic shop. In the shop, Kavin has 'N' bulbs. To sell these bulbs, Kavin has to check if they are of good quality. To check bulbs, Kavin uses a unique technique. In this technique, in the first round he, turn on all the bulbs, then in the second round, he turns off every second bulb then in the third round, he chooses every third bulb and turns off it is on, or turn on if it is off and so no. He repeats this process 'N' times. The number of bulbs that are on after the end of 'N' rounds are of good quality. Your task is to help Kavin in finding the number of good bulbs  
  
Program : (BulbOn)  
**package** com.handson;

**import** java.util.Scanner;

**public** **class** BulbOn {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter n: ");

**int** n = sc.nextInt();

**int** i = 1;

**while**(i <= n) {

**int** ans = i \* i;

**if**(ans == n) {

System.***out***.println(i);

**break**;

}

**else** **if**(ans > n) {

i--;

System.***out***.println(i);

**break**;

}

**else**

i++;

}

}

}

Output :   
Enter n:

15

3

30.You are given two positive integers: ‘n,’ ‘maxVaccines’ denoting the number of days for which this vaccination drive will go on and the total number of vaccines available for the drive, respectively. You have to find the number of vaccines administered each day. You are also given a number ‘dayNumber,’ and we are interested to know the maximum number of vaccines that can be administered on ‘dayNumber’ th day.