**Q1. Take an input n which is size of array then n more inputs the values in array then find the square of special elements.**

**An element nums[i] of nums is called special if i divides n,** **i.e. n % i == 0.**

**Return *the sum of the squares of all special elements of* nums.**

**Example 1:**

**Input:**

**4**

**1 2 3 4**

**Output: 21**

**Explanation: There are exactly 3 special elements in nums: nums[1] since 1** **divides 4, nums[2] since 2 divides 4, and nums[4] since 4 divides 4.**   
**Hence, the sum of the squares of all special elements of nums is nums[1] \* nums[1] + nums[2] \* nums[2] + nums[4] \* nums[4] = 1 \* 1 + 2 \* 2 + 4 \* 4 = 21.**

**Example 2:**

**Input:**

**6**

**2 7 1 19 18 3**

**Output: 63**

**Explanation: There are exactly 4 special elements in nums: nums[1] since 1** **divides 6, nums[2] since 2 divides 6, nums[3] since 3 divides 6, and nums[6] since 6 divides 6.**   
**Hence, the sum of the squares of all special elements of nums is nums[1] \* nums[1] + nums[2] \* nums[2] + nums[3] \* nums[3] + nums[6] \* nums[6] = 2 \* 2 + 7 \* 7 + 1 \* 1 + 3 \* 3 = 63.**

**Constraints:**

* **1 <= nums.length == n <= 50**
* **1 <= nums[i] <= 50**

**Solution:**

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

**public class Main {**

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

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

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

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

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

**arr[i] = sc.nextInt();**

**}**

**System.out.println(sumOfSquares(arr));**

**}**

**public static int sumOfSquares(int[] nums) {**

**int ans = 0;**

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

**if (nums.length % (i + 1) == 0)**

**ans += (nums[i] \* nums[i]);**

**}**

**return ans;**

**}**

**}**

**Test Case:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **4**  **3 4 5 6** | **61** |
| **5**  **11 3 4 6 7** | **170** |
| **3**  **4 7 2** | **20** |
| **8**  **5 2 7 1 6 6 9 0** | **30** |
| **4**  **11 11 1 11** | **363** |

**Q2. Deepak has a limited amount of money that he can spend on his girlfriend. So, he decides to buy two roses for her. Since roses are of varying sizes, their prices are different. Deepak wishes to completely spend that fixed amount of money on buying roses for her.**  
**As he wishes to spend all the money, he should choose a pair of roses whose prices when summed up are equal to the money that he has.**  
**Help Deepak choose such a pair of roses for his girlfriend.**  
  
**NOTE: If there are multiple solutions print the solution that minimizes the difference between the prices i and j. After each test case, you must print a blank line.**

**Input Format**

**The first line indicates the number of available roses, N is given.**  
**The next line will have N integers, representing the price of each rose, a rose that costs less than 1000001.**  
**Then there is another line with an integer M, representing how much money Deepak has.**  
**There is a blank line after each test case.**

**Constraints**

**2 ≤ N ≤ 10000**  
**Price[i]<1000001**

**Output Format**

**For each test case, you must print the message: ‘Deepak should buy roses whose prices are i and j.’, where i and j are the prices of the roses whose sum is equal do M and i ≤ j. You can consider that it is always possible to find a solution. If there are multiple solutions print the solution that minimizes the difference between the prices i and j.**

**Sample Input**

**2**  
**40 40**  
**80**  
  
**5**  
**10 2 6 8 4**  
**10**

**Sample Output**

**Deepak should buy roses whose prices are 40 and 40.**  
**Deepak should buy roses whose prices are 4 and 6.**

**Explanation**

**Find two such kinds of roses which have sum up to equal to Deepak's Money.**

**Solution:**

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

**public class Main {**

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

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

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

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

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

**arr[j] = sc.nextInt();**

**}**

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

**int diff = Integer.MAX\_VALUE;**

**int a = 0;**

**int b = 0;**

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

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

**if (arr[i] + arr[j] == m && (Math.abs(arr[i] - arr[j]) < diff)) {**

**a = arr[i];**

**b = arr[j];**

**diff = Math.abs(arr[i] - arr[j]);**

**}**

**}**

**}**

**if (a < b) {**

**System.out.println("Deepak should buy roses whose prices are " + a + " and " + b + ".");**

**} else {**

**System.out.println("Deepak should buy roses whose prices are " + b + " and " + a + ".");**

**}**

**}**

**}**

**Testcase's:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **2**  **40 40**  **80** | **Deepak should buy roses whose prices are 0 and 0.** |
| **5**  **10 2 6 8 4**  **10** | **Deepak should buy roses whose prices are 4 and 6.** |
| **5**  **10 20 30 40 50**  **30** | **Deepak should buy roses whose prices are 10 and 20.** |
| **5**  **20 10 50 40 80**  **60** | **Deepak should buy roses whose prices are 20 and 40.** |
| **5**  **100 300 400 900 200**  **500** | **Deepak should buy roses whose prices are 200 and 300.** |

**Q3. You are provided two sorted arrays. You need to find the maximum length of bitonic subsequence. You need to find the sum of the maximum sum path to reach from beginning of any array to end of any of the two arrays. You can switch from one array to another array only at common elements.**

**Input Format**

**The first line contains the integer t which is the number of test case. For each test case, it contains two integers n and m which is the size of arrays and next two lines contain n and m space separated integers respectively.**

**Constraints**

**1<=t<=100 1<=n,m<=100000**

**Output Format**

**Print the maximum path.**

**Sample Input**

**1**  
**8 8**  
**2 3 7 10 12 15 30 34**  
**1 5 7 8 10 15 16 19**

**Sample Output**

**122**

**Explanation**

**122 is sum of 1, 5, 7, 8, 10, 12, 15, 30, 34**

**Solution:**

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

**public class Main {**

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

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

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

**while (t-- > 0) {**

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

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

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

**int[] brr = new int[m];**

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

**arr[i] = sc.nextInt();**

**}**

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

**brr[i] = sc.nextInt();**

**}**

**int i = 0;**

**int j = 0;**

**int p1 = 0;**

**int p2 = 0;**

**int ans = 0;**

**while (i < arr.length && j < brr.length) {**

**if (arr[i] < brr[j]) {**

**i++;**

**} else if (arr[i] > brr[j]) {**

**j++;**

**} else {**

**int sum1 = 0;**

**int sum2 = 0;**

**for (int k = p1; k <= i; k++) {**

**sum1 += arr[k];**

**}**

**for (int k = p2; k <= j; k++) {**

**sum2 += brr[k];**

**}**

**ans = ans + Math.max(sum1, sum2);**

**p1 = i + 1;**

**p2 = j + 1;**

**i++;**

**j++;**

**}**

**}**

**int s1 = 0;**

**while (p1 < arr.length) {**

**s1 += arr[p1];**

**p1++;**

**}**

**int s2 = 0;**

**while (p2 < brr.length) {**

**s2 += brr[p2];**

**p2++;**

**}**

**ans += Math.max(s1, s2);**

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

**}**

**}**

**}**

**Testcases:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **1**  **8 8**  **2 3 7 10 12 15 30 34**  **1 5 7 8 10 15 16 19** | **22** |
| **1**  **4**  **4**  **1 3 8 9**  **2 3 6 7** | **22** |

**Q4 . Ramu often uses public transport. The transport in the city is of two types: cabs and rickshaws. The city has n rickshaws and m cabs, the rickshaws are numbered by integers from 1 to n, the cabs are numbered by integers from 1 to m.**

**Public transport is not free. There are 4 types of tickets:**

**A ticket for one ride on some rickshaw or cab. It costs c1 rupees;**  
**A ticket for an unlimited number of rides on some rickshaw or on some cab. It costs c2 ruppees;**  
**A ticket for an unlimited number of rides on all rickshaws or all cabs. It costs c3 ruppees;**  
**A ticket for an unlimited number of rides on all rickshaws and cabs. It costs c4 ruppees.**

**Ramu knows for sure the number of rides he is going to make and the transport he is going to use. He asked you for help to find the minimum sum of ruppees he will have to spend on the tickets.**

**Input Format**

**Each Test case has 4 lines which are as follows:**

**The first line contains four integers c1, c2, c3, c4 (1 ≤ c1, c2, c3, c4 ≤ 1000) — the costs of the tickets.**  
**The second line contains two integers n and m (1 ≤ n, m ≤ 1000) — the number of rickshaws and cabs Ramu is going to use.**  
**The third line contains n integers ai (0 ≤ ai ≤ 1000) — the number of times Ramu is going to use the rickshaw number i.**  
**The fourth line contains m integers bi (0 ≤ bi ≤ 1000) — the number of times Ramu is going to use the cab number i.**

**Constraints**

**1 <= T <= 1000 , where T is no of testcases**  
**1 ≤ c1, c2, c3, c4 ≤ 1000**  
**1 ≤ n, m ≤ 1000**  
**0 ≤ ai , bi ≤ 1000**

**Output Format**

**For each testcase, print a single number - the minimum sum of rupees Ramu will have to spend on the tickets in a new line.**

**Sample Input**

**1 3 7 19**  
**2 3**  
**2 5**  
**4 4 4**

**Sample Output**

**12**

**Example 2**  
**4 3 2 1**  
**1 3**  
**798**  
**1 2 3**

**Sample Output**  
**1**

**Explanation**

**For the first testcase ,**  
**The total cost of rickshaws = min( min(2 \* 1, 3) + min(5 \* 1, 3), 7) = min(5, 7) = 5**  
**The total cost of cabs = min( min(4 \* 1, 3) + min(4 \* 1, 3) + min(4 \* 1, 3) , 7) = min ( 9, 7) = 7**  
**Total final cost = min( totalCabCost + totalRickshawCost , c4) = min( 5 + 7, 19) = min ( 12, 19) = 12**  
**We print 12.**

**Solution:**

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

**public class Main {**

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

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

**int c1=sc.nextInt();**

**int c2=sc.nextInt();**

**int c3=sc.nextInt();**

**int c4=sc.nextInt();**

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

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

**int r[]=new int [tr];**

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

**r[i]= sc.nextInt();**

**}**

**int c[]=new int [tc];**

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

**c[i]= sc.nextInt();**

**}**

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

**System.out.print(isitpossible(c1,c2,c3,c4,r,c));**

**}**

**public static int isitpossible(int c1,int c2,int c3,int c4,int r[],int c[]) {**

**int a1=0;**

**int a2=0;**

**for(int i=0;i<=r.length-1;i++) {**

**a1+=Math.min(r[i]\*c1,c2);**

**}**

**a1=Math.min(a1,c3);**

**for(int i=0;i<=c.length-1;i++) {**

**a2+=Math.min(c[i]\*c1,c2);**

**}**

**a2=Math.min(a2,c3);**

**int ans = Math.min(a1+a2,c4);**

**return ans;**

**}**

**}**

**TestCases:**

|  |  |
| --- | --- |
| **Input** | **output** |
| **4 3 2 1** **1 3** **798** **1 2 3** | **1** |
| **1 3 7 19** **2 3** **2 5** **4 4 4** | **12** |
| **3 4 5 6**  **3 2**  **3 4 5**  **6 7** | **6** |
| **8 9 25 1**  **2 2**  **5 8**  **8 8** | **1** |
| **6 8 2 12**  **4 4**  **1 2 3 4**  **5 6 7 8** | **4** |

**Q5. Once upon a time, in a land of mathematical wonders, there lived a clever mathematician named Alice. She had an array of numbers laid out in front of her, and she was on a quest to find a magical solution.**

**Alice's array was no ordinary one; it contained a series of integers that held the power to reveal hidden truths. Each number in the array had a unique role to play, but Alice was after a special kind of magic. She wanted to discover, for each number in the array, the product of all other numbers, except the one at that very spot.**

**It was a challenging problem, one that required her utmost mathematical skills and a touch of magic.**

**You must write an algorithm that runs in O(n) time and without using the division operation.**

**Example 1:**

**4**

**1 2 3 4**

**Output: 24 12 8 6**

**Example 2:**

**5**

**-1 1 0 -3 3**

**Output: 0 0 9 0 0**

**Constraints:**

**2 <= nums.length <= 105**

**-30 <= nums[i] <= 30**

**The product of any prefix or suffix of nums is guaranteed to fit in a 32-bit integer.**

**Solution :**

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

**public class Main {**

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

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

**long n = sc.nextLong();**

**int arr[] = new int[(int) n];**

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

**arr[i] = sc.nextInt();**

**}**

**Product(arr, n);**

**}**

**public static void Product(int[] arr, long n) {**

**long[] left = new long[(int) n];**

**left[0] = 1;**

**for (long i = 1; i < left.length; i++) {**

**left[(int) i] = left[(int) (i - 1)] \* arr[(int) (i - 1)];**

**}**

**long[] right = new long[(int) n];**

**right[(int) (n - 1)] = 1;**

**for (long i = right.length - 2; i >= 0; i--) {**

**right[(int) i] = right[(int) (i + 1)] \* arr[(int) (i + 1)];**

**}**

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

**System.out.print(left[(int) i] \* right[(int) i] + " ");**

**}**

**}**

**}**

**TestCases:**

|  |  |
| --- | --- |
| **Input** | **output** |
| **4**  **1 2 3 4** | **24 12 8 6** |
| **5**  **-1 1 0 -3 3** | **0 0 9 0 0** |
| **5**  **1 6 2 7 2** | **168 28 84 24 84** |
| **4**  **1 7 3 9 0** | **189 27 63 21** |
| **2**  **1 1** | **1 1** |

**Q 6. In a small, bustling town, there was a friendly competition every year. The townsfolk loved to gather in the central square and participate in the "Max Element Challenge." It was a contest that tested their skills in finding the largest number in a given set.**

**One sunny morning, as the townspeople prepared for the annual event, a young boy named Sam felt particularly eager to participate. He had been practicing his number skills and was determined to win this year's challenge.**

**The rules were simple: A large array of numbers was placed on a grand stage in the square. Each participant had one chance to step forward, examine the array, and pick the maximum number. The one who found the largest number would be crowned the champion.**

**As Sam's turn came, he confidently approached the stage. He carefully scanned the array of numbers, comparing them in his mind. He remembered his practice sessions and the importance of keeping track of the largest number he had seen so far**

**Input Format**

**First line contains integer n as size of array. Next n lines contains a single integer as element of array.**

**Constraints**

**N cannot be Negative. Range of Numbers can be between -1000000000 to 1000000000**

**Output Format**

**Print the required output.**

**Sample Input**

**4**  
**2**  
**8**  
**6**  
**4**

**Sample Output**

**8**

**Explanation**

**Arrays= {2, 8, 6, 4} => Max value = 8 .**

**Solution:**

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

**public class Main {**

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

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

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

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

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

**arr[i]= sc.nextInt();**

**}**

**System.out.println(max(arr));**

**}**

**public static int max(int [] arr){**

**int max = Integer.MIN\_VALUE;**

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

**if(arr[i]>max)**

**max=arr[i];**

**}**

**return max;**

**}**

**}**

**TestCases:**

|  |  |
| --- | --- |
| **Input** | **output** |
| **5**  **1 3 5 2 7** | **7** |
| **4**  **12 25 77 11** | **77** |
| **3**  **-11 –44 –15** | **-11** |
| **2**  **1 1** | **1** |
| **3**  **0 0 0** | **0** |

**Q 7. Take as input N, the size of an array. Take N more inputs and store that in an array. Take another number’s input as M. Write a function which returns the index on which M is found in an array, in case M is not found -1 is returned. Print the value returned.**

1. **It reads a number N.**  
   **2.Take Another N numbers as an input and store them in an Array.**
2. **Take another number M as an input.**
3. **If M is found in the Array the index of M is returned else -1 is returned and print the value returned.**

**Input Format**

**Constraints**

**N cannot be Negative. Range of Numbers can be between -1000000000 to 1000000000. M can be between -1000000000 to 1000000000.**

**Output Format**

**Sample Input**

**5**  
**2**  
**4**  
**6**  
**9**  
**17**  
**17**

**Sample Output**

**4**

**Explanation**

**Given array = {2, 4, 6, 9, 17}. Target number = 17. Index = 4.**

**Solution:**

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

**public class Main {**

**public static int index(int array[],int N,int M) {**

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

**if (array[i] == M) {**

**return i;**

**}**

**}**

**return -1;**

**}**

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

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

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

**int array []= new int[N];**

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

**array[i] = sc.nextInt();**

**}**

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

**index(array,N,M);**

**System.out.println(index(array,N,M));**

**}**

**}**

**TestCases:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **4**  **1 3 4 7**  **7** | **3** |
| **3**  **1 2 3**  **4** | **-1** |
| **5**  **55 7 222 444 1**  **222** | **2** |
| **1**  **1**  **1** | **1** |
| **7**  **1 5 12 23 5 16 –7**  **-7** | **6** |

**Q 8. Take as input N, the size of array. Take N more inputs and store that in an array. Take as input “target”, a number. Write a function which prints all pairs of numbers which sum to target.**

**Input Format**

**The first line contains input N. Next N lines contains the elements of array and (N+1)th line contains target number.**

**Constraints**

**Length of the arrays should be between 1 and 1000.**

**Output Format**

**Print all the pairs of numbers which sum to target. Print each pair in increasing order.**

**Sample Input**

**5**  
**1 3 4 2 5**  
**5**

**Sample Output**

**1 4**  
**2 3**

**Explanation**

**Find any pair of elements in the array which has sum equal to target element and print them.**

**Solution:**

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

**public class Main {**

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

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

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

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

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

**arr[i] = sc.nextInt();**

**}**

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

**Arrays.sort(arr);**

**Targetsum(arr, item);**

**}**

**public static void Targetsum(int arr[],int target) {**

**int low=0;**

**int end=arr.length-1;**

**while(low<end) {**

**if(arr[low]+arr[end]>target) {**

**end--;**

**}**

**else if(arr[low]+arr[end]==target) {**

**System.out.println(arr[low]+" and "+arr[end]);**

**low++;**

**end--;**

**}**

**else {**

**low++;**

**}**

**}**

**}**

**}**

**TestCases:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **3**  **1 4 2**  **3** | **1 2** |
| **5**  **1 3 4 2 5**  **9** | **4 5** |
| **7**  **1 2 3 4 5 6 7 8**  **15** | **7 8** |
| **2**  **1 2**  **3** | **1 2** |
| **5**  **1 5 8 2 3**  **6** | **1 5** |

**Q 9. Given an array nums of length n. We define a running sum of an array as for every index runningSum[i] = sum(nums[0]…nums[i]).**

**Return the running sum of array for each i (0 <= i < n).**

**Input Format**

**First line contains an integer n representing number of elements. Next line contains n integers denoting array elements.**

**Constraints**

**1 <= nums.length <= 1000**  
**-10^6 <= nums[i] <= 10^6**

**Output Format**

**An integer representing running sum array of the given array**

**Sample Input**

**4**  
**1 2 3 4**

**Sample Output**

**1 3 6 10**

**Explanation**

**Running sum is obtained as follows: [1, 1+2, 1+2+3, 1+2+3+4].**

**Solution:**

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

**public class Main {**

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

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

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

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

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

**arr[i]= sc.nextInt();**

**reverse(arr);**

**}**

**public static void reverse(int arr[]) {**

**int sum=0;**

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

**sum+=arr[i];**

**System.out.print(sum+" ");**

**}**

**}**

**}**

**TestCases:**

|  |  |
| --- | --- |
| **Input** | **output** |
| **3**  **1 2 3** | **1 3 6** |
| **4**  **3 2 1 4** | **3 5 6 10** |
| **2**  **4 5** | **4 9** |
| **5**  **1 2 3 4 5** | **1 3 6 10 15** |
| **1**  **5** | **5** |

**Q 10. Given an array and you need to find out the contiguous subarray that has the largest sum and return the sum of the subarray.**

**Input Format**

**A number representing n length of array. N lines representing the n elements of the array. The last line takes A number representing the target sum.**

**Constraints**

**The length of the array must be less than 10000**

**Output Format**

**Print the sum of the maximum subarray**

**Sample Input**

**9**  
**-****2 1 –****3 4 –****1 2** **1 –****5 4**

**Sample Output**

**6**

**Explanation**

**{-2,****1,-3,****4,-1,2,****1,-5,4}**  
**The resultant will be 6.**  
**{****4,-1,2,1}is the subarray that has the largest sum = 6.**

**Solution:**

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

**public class Main {**

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

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

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

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

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

**arr[i]= sc.nextInt();**

**}**

**System.out.println (max\_sum( arr));**

**}**

**public static int max\_sum(int[] arr) {**

**int ans = Integer.MIN\_VALUE;**

**int sum = 0;**

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

**sum += arr[i];**

**ans = Math.max(ans, sum);**

**if(sum<0)**

**sum=0;**

**}**

**return ans;**

**}**

**}**

**Testcase:**

|  |  |
| --- | --- |
| **Input** | **output** |
| **9**  **-2 1 –3 4 –1 2 1 –5 4** | **6** |
| **4**  **1 2 3 4** | **10** |
| **5**  **1 2 –4 5 6** | **11** |
| **3**  **2 –5 9** | **9** |
| **5**  **-1 –2 –3 –4 –5** | **-1** |