L19 : GreedyProblems

1-Tut : Min. Absolute Difference In Array

Send Feedback

Given an integer array A of size N, find and return the minimum absolute difference between any two elements in the array.

We define the absolute difference between two elements a_i and a_j (where i != j) as $|a_i - a_j|$.

Input format :

The first line of input contains an integer that denotes the number of test cases. Let us denote it by the symbol T.

Each of the test cases contain two lines. The first line of each test case contains an integer, that denotes the value of N. The following line contains N space separated integers, that denote the array elements.

Constraints :

1 <= T <= 10

2 <= N <= 10^5

0 <= arr[i] <= 10^8

Output Format :

You have to print minimum difference for each test case in new line.

Sample Input 1:

```
1
5
2 9 0 4 5
```

Sample Output 1:

```
1
```

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long int

int mod(int a)
{
        if(a<0)
        {
                return -a;
        }
        return a;
}
ll minAbsoluteDiff(ll * arr, int n) {
        sort(arr, arr+n);
        ll i=0;
        ll j=1;
        ll minimum=INT_MAX;
```

```
        while(j<n)
        {
                if(mod(arr[i]-arr[j])<minimum)
                {
                        minimum=mod(arr[i]-arr[j]);
                }
                i++;
                j++;
        }
        return minimum;
}

int main(){

    // write your code here
    int t;
    cin>>t;
    while(t--){
        int n;
        cin>>n;
        ll * arr = new ll[n];
        for(int i=0;i<n;i++){
            cin >> arr[i];
        }
        ll ans = minAbsoluteDiff(arr,n);
        cout<<ans<<endl;
    }
    return 0;
}
```

2-Tut : Nikunj and Donuts

Send Feedback

Nikunj loves donuts, but he also likes to stay fit. He eats n donuts in one sitting, and each donut has a calorie count, ci. After eating a donut with k calories, he must walk at least 2^j * k(where j is the number donuts he has already eaten) miles to maintain his weight.

Given the individual calorie counts for each of the n donuts, find and print a long integer denoting the minimum number of miles Nikunj must walk to maintain his weight. Note that he can eat the donuts in any order.

Input Format:

First line of input will contain T(number of test cases), each test case follows as.
The first line contains an integer, n, denoting the number of donuts.
The second line contains n space-separated integers describing the respective calorie counts of each donut, i.e ci.

Constraints

1 <= T <= 10^5

1 ≤ n ≤ 40

1 ≤ ci ≤ 1000

Output Format:

Print a long integer denoting the minimum number of miles Nikunj must walk to maintain his weight for each test case in new line

Sample Input 1

1

3

1 3 2

Sample Output 1

11

Explanation:

The minimum miles that Nikunj has to walk are 11 miles. He will first eat donut with calorie count 3, then the one with calorie count 2 and then the donut with calorie count as 1. The miles travelled are calculated as: 2^0*3 + 2^1*2 + 2^2*1.

```
#include<bits/stdc++.h>
using namespace std;

#define ll unsigned long long int

ll minimum_miles(ll *arr, ll n)
{
        ll total_miles=0;
        for(ll i=0; i<n; i++)
        {
                total_miles+=pow(2, i)*arr[n-1-i];
        }
        return total_miles;
}
int main()
{
    int t;
    cin>>t;

    while(t--){
       ll n;
                cin>>n;
                ll *arr=new ll [n];
                for(ll i=0; i<n; i++)
                {
                        cin>>arr[i];
```

```
                }
                sort(arr, arr+n);
                cout<<minimum_miles(arr, n)<<endl;
    }

}
```

3-Tut : Activity Selection

Send Feedback

You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

Input Format

The first line of input contains one integer denoting N.
Next N lines contains two space separated integers denoting the start time and finish time for the ith activity.

Constraints

1 ≤ N ≤ 10^6
1 ≤ ai, di ≤ 10^9

Output Format

Output one integer, the maximum number of activities that can be performed

Sample Input 1

6
1 2
3 4
0 6
5 7
8 9
5 9

Sample Output 1

4

Explanation:

The four activities will be done at following time intervals:
1. 1 2
2. 3 4
3. 5 7
4. 8 9

```
#include<bits/stdc++.h>
using namespace std;

bool compare(pair<int, int> i1, pair<int, int> i2)
{
        return i1.second<i2.second;
}
```

```
int activities(pair<int, int>*arr, int n)
{
        int current_ending_time=arr[0].second;
        int count=1;
        for(int i=1; i<n; i++)
        {
                if(current_ending_time<=arr[i].first)
                {
                        count+=1;
                        current_ending_time=arr[i].second;
                }
        }
        return count;
}
int main()
{
        int n;
        cin>>n;
        pair<int, int> *arr=new pair<int, int>[n];
        for(int i=0; i<n; i++)
        {
                cin>>arr[i].first>>arr[i].second;
        }
        sort(arr, arr+n, compare);
        cout<<activities(arr, n)<<endl;
}
```

4-Tut : Fractional Knapsack

Send Feedback

You are given weights and values of N items. You have to select and put these selected items in a knapsack of capacity W. Select the items in such a way that selected items give the maximum total value possible with given capacity of the knapsack.

Note: You are allowed to break the items in parts.

Input Format:

The first line of input contains two space separated integers, that denote the value of N and W.
Each of the following N lines contains two space separated integers, that denote value and weight, respectively, of the N items.

Constraints:

1 <= N = 2*10^5
1 <= W <= 10^9
1 <= value, weight <= 10^5
Time Limit: 1 sec

Output Format:

Print the maximum total value upto six decimal places.

Sample Input 1:

```
4 22
6 4
6 4
4 4
4 4
```

Sample Output 1:

```
20.000000
```

Explanation:

The total weight of all the items is 16 units, which is less than the total capacity of knapsack, i.e 22 units. Hence, we will add all the items in the knapsack and total value will be 20 units.

```
#include <bits/stdc++.h>
using namespace std;

struct Item {
        int value;
    int weight;
};

bool cmp(struct Item a, struct Item b)
{
        double r1 = (double)a.value / a.weight;
        double r2 = (double)b.value / b.weight;
        return r1 > r2;
}

double fractionalKnapsack(Item * arr, int N, int size)
{
        sort(arr, arr + size, cmp);

        int curWeight = 0;

        double finalvalue = 0.0;

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

                if (curWeight + arr[i].weight <= N) {
                        curWeight += arr[i].weight;
                        finalvalue += arr[i].value;
                } else {
                        int remain = N - curWeight;
                        finalvalue += arr[i].value * ((double)remain / arr[i].weight);
                        break;
```

```
                }
            }

            return finalvalue;
}

int main()
{
            int n;
    long long int w;
    cin >> n >> w;

    Item * arr = new Item [n];

    for(int i=0;i<n;i++){
        cin>>arr[i].value>>arr[i].weight;
    }

            double ans = fractionalKnapsack(arr, w, n);
    cout.precision(6);
            cout << fixed << ans << endl;
            return 0;
}
```

5-Tut : Weighted Job Scheduling

Send Feedback

You are given N jobs where every job is represented as:

1. Start Time

2. Finish Time

3. Profit Associated

Find the maximum profit subset of jobs such that no two jobs in the subset overlap.

Input Format:

The first line of input contains an integer denoting N.
Next N lines contains three space separated integers denoting the start time, finish time and the profit associated with the ith job.

Constraints:

1 ≤ N ≤ 10^6
1 ≤ ai, di, p ≤ 10^6

Output Format:

Output one integer, the maximum profit that can be achieved.

Sample Input 1

4
3 10 20

```
1 2 50
6 19 100
2 100 200
```

Sample Output 1

```
250
```

```
#include<bits/stdc++.h>
using namespace std;

using namespace std;
// A job has start time, finish time and profit.
struct Job
{
    int start, finish, profit;
};
// A utility function that is used for sorting events
// according to finish time
bool myfunction(Job s1, Job s2)
{
    return (s1.finish < s2.finish);
}
// A Binary Search based function to find the latest job
// (before current job) that doesn't conflict with current
// job. "index" is index of the current job. This function
// returns -1 if all jobs before index conflict with it.
// The array jobs[] is sorted in increasing order of finish
// time.
int binarySearch(Job jobs[], int index)
{
// Initialize 'lo' and 'hi' for Binary Search
    int lo = 0, hi = index - 1;
// Perform binary Search iteratively
    while (lo <= hi)
    {
        int mid = (lo + hi) / 2;
        if (jobs[mid].finish <= jobs[index].start)
        {
            if (jobs[mid + 1].finish <= jobs[index].start)
                lo = mid + 1;
            else
                return mid;
        }
        else
            hi = mid - 1;
```

```
    }
    return -1;
}
// The main function that returns the maximum possible
// profit from given array of jobs
int findMaxProfit(Job arr[], int n)
{
    // Sort jobs according to finish time
    sort(arr, arr+n, myfunction);
    // Create an array to store solutions of subproblems. table[i]
    // stores the profit for jobs till arr[i] (including arr[i])
    int *table = new int[n];
    table[0] = arr[0].profit;
    // Fill entries in table[] using recursive property
    for (int i=1; i<n; i++)
    {
        // Find profit including the current job
        int inclProf = arr[i].profit;
        int l = binarySearch(arr, i);
        if (l != -1)
            inclProf += table[l];
        // Store maximum of including and excluding
        table[i] = max(inclProf, table[i-1]);
    }
    // Store result and free dynamic memory allocated for table[]
    int result = table[n-1];
    delete[] table;
    return result;
}
// Driver program
int main()
{
    int n;
    cin>>n;
    Job arr[n];
    for(int i=0;i<n;i++)
    {
        cin>>arr[i].start>>arr[i].finish>>arr[i].profit;
    }
    cout << findMaxProfit(arr, n);
    return 0;
}
```

Live Problem 1 : https://www.codechef.com/problems/SVENGY
Live Problem 2 : https://dmoj.ca/problem/coci13c1p4

6-Ass : Winning Lottery

Send Feedback

Harshit knows through his resources that this time the winning lottery number is the smallest number whose sum of the digits is S and the number of digits is D. You have to help Harshit and print the winning lottery number.

Input Format:

First line of input contians an integer T, representing the number of test cases.
Next T lines contains two space-separated integers: S,D

Constraints:

1 <= T <= 1000
1 <= D <= 1000
1 <= S <= 9*D
Time Limit: 1 second

Output Format

The output contains a single integer denoting the winning lottery number

Sample Input 1:

1
9 2

Sample Output 1:

18

Explanation

There are many other possible numbers like 45, 54, 90, etc with the sum of digits as 9 and number of digits as 2. The smallest of them is 18.

```
#include<bits/stdc++.h>
using namespace std;
void findSmallest(int m, int s)
{

    if (s == 0)
    {
        (m == 1)? cout << 0
                : cout << "Not possible";
        return ;
    }

    if (s > 9*m)
    {
        cout << "Not possible";
        return ;
```

```
    }

    int res[m];

    s -= 1;

    for (int i=m-1; i>0; i--)
    {

        if (s > 9)
        {
            res[i] = 9;
            s -= 9;
        }
        else
        {
            res[i] = s;
            s = 0;
        }
    }

    res[0] = s + 1;


    for (int i=0; i<m; i++)
        cout << res[i];
}


int main()
{
    int t;
    cin>>t;
    while(t--){

        int s, d;
                    cin >> s >> d;
                    findSmallest(d, s);
        cout<<endl;

    }

}
```

7-Ass : King Arthur and Excalibur

Send Feedback

King Arthur's land is in danger. He wants to fend off the Saxon invaders. In order to do so, he needs to find the fabled sword, Excalibur. He has a vague idea about its location, and has dispatched his best Knights and their Squires to acquire it. The Knights go to the location to find The Dark Forest. They set up tents and decided to split into groups to find Excalibur faster. The famous knight, Sir Lancelot was trying to form search parties, but ran into some trouble.

Most of the knights were untrained for quests, and sending them alone into the dangers of the quest would be a grave mistake. Each of the knights have been given a positive integer parameter by King Arthur, ti - their training score. Sir Lancelot decided that a knight with training t can join the group of t or more knights.

Some of the knights are very egotistical and can't work well with others. Hence Sir Lancelot decided it is not necessary to include all the knights, some can stay back and defend the tents from danger. Now Sir Lancelot needs to figure out how many search parties he can organize. Sir Lancelot is one of the most responsible knights and doesn't want to go back to King Arthur empty handed. He would rather die than fail. You need to help him in his quest.

Input Format:

First line of input will contain T(number of test cases), each test case follows as.
Line1 : will contain an integer N denoting the total number of explorer
Line2: Will contain N space-separated integers denoting the inexperience level

Ouput Format:

For each test case output the answer in a new line.

Constraints:

1 <= T <= 100
1 <= N <= 10^4
1 <= levels <= N

Sample Input:

```
2
3
1 1 1
5
2 3 1 2 2
```

Sample Output:

```
3

2
```

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4.
5. int main()
```

```
{

        int t;
    cin >> t;
        while(t--)
        {
                int n;
      cin >> n;
                vector<int> a(n+1);
                for(int i = 1 ; i <= n ; ++i) {
                        cin >> a[i];
                }
                sort(a.begin()+1, a.end());

                vector<int> dp(n+1, 0);

                int ans = 0;
                for(int i = 1 ; i <= n ; ++i) {
                        if(i - a[i] >= 0) {
                                dp[i] = max(dp[i-1], dp[i-a[i]] + 1);
                        }
                        else {
                                dp[i] = dp[i-1];
                        }
                }

                cout << dp[n] << endl;
        }
        return 0;
}
```

8-Ass : Call of War

Send Feedback

DecenTile Games have launched a new First Person Shooter soldier game, called the Call of War, that young gamers can play on their website.

Our protagonist, Ninja (played by you) is a soldier whose mission is to kill all the enemies plotting against you. Your enemies are standing in a circle, numbered clockwise from 1 to n. Initially, the i-th enemy has ai health.

You have only one pistol, and infinite bullet magazines. You have to shoot the enemies in order to kill them. Each bullet fired at the enemy decreases their health by 1 (deals 1 damage to it). When the health of some enemy i becomes 0 or less than 0, he dies and his grenade drops down and explodes, dealing bi damage to the next enemy (enemy i+1, if i<n, or enemy 1, if i=n). If the next enemy is already dead, then

nothing happens. If the frag from the grenade kills the next enemy, even he drops a grenade, damaging the enemy after him and possibly triggering another explosion, and so on.
You have to calculate the minimum number of bullets you need in order to kill all the enemies and win the game.

Input Format:
First line of input will contain T(number of test cases), each test case follows as.
Line 1: contain an integer N
Next N line contains two space separated integers a and b
Output Format:
For each test case print the output in a new line.
Constraints:
1 <= T <= 100
1 <= N <= 10^4
1 <= a, b <= 10^12
Sample Input:
1
3
7 15
2 14
5 3
Sample Output:

6

```
#include<bits/stdc++.h>
using namespace std;

#define ll long long int

int main()
{


        int t;
    cin >> t;
        while(t--)
        {
                int n;
      cin >> n;
      ll * a = new ll[n];
      ll * b = new ll[n];
      ll * d = new ll[n];

      ll * ans = new ll[n];
```

```
21.
22.         for(int i=0;i<n;i++){
23.             cin >> a[i]>> b[i];
24.         }
25.
26.         d[0] = a[0] - b[n-1];
27.         if(d[0]<0){
28.             d[0]=0;
29.         }
30.
31.         ll sumD = d[0];
32.
33.         for(int i=1;i<n;i++){
34.
35.             d[i] = a[i]-b[i-1];
36.             if(d[i]<0){
37.                 d[i]=0;
38.             }
39.             sumD+=d[i];
40.         }
41.
42.         ll minCost = LLONG_MAX;
43.         for(ll i=0 ; i<n ; i++){
44.
45.             ans[i] = a[i] + sumD - d[i];
46.             if(ans[i]<minCost){
47.                 minCost = ans[i];
48.             }
49.
50.         }
51.         cout<<minCost<<endl;
52.
53.             }
54.             return 0;
55. }
```

9-Ass : Minimum Number of Platforms

Send Feedback

You have been given two arrays, 'AT' and 'DT', representing the arrival and departure times of all trains that reach a railway station.

Your task is to find the minimum number of platforms required for the railway station so that no train needs to wait.

Note :

1. Every train will depart on the same day and the departure time will always be greater than the arrival time. For example, A train with arrival time 2240 and departure time 1930 is not possible.

2. Time will be given in 24H format and colons will be omitted for convenience. For example, 9:05AM will be given as "905", or 9:10PM will be given as "2110".

3. Also, there will be no leading zeroes in the given times. For example, 12:10AM will be given as “10” and not as “0010”.

Input Format

The first line of input contains an integer 'T' representing the number of test cases.
The first line of each test case contains an integer 'N', representing the total number of trains.

The second line of each test case contains 'N' single-spaced separated elements of the array 'AT', representing the arrival times of all the trains.

The third line of each test case contains 'N' single-spaced separated elements of the array 'DT', representing the departure times of all the trains.

Output Format:

For each test case, return the minimum number of platforms required for the railway station so that no train needs to wait.

Note :

You don't need to print the output, it has already been taken care of. You just need to implement the given function.

Follow up :

Try to solve the problem in O(N) time and O(1) space.

Constraints:

1 <= T <= 10
1 <= N <= 50000
0 <= AT[i] <= DT[i] <= 2359
Where 'AT[i]' and 'DT[i]' are the elements of the arrival and the departure arrays respectively.
Time limit: 1 sec

Sample Input 1:

2
6
900 940 950 1100 1500 1800
910 1200 1120 1130 1900 2000
4
100 200 300 400
200 300 400 500

Sample Output 1:

3
2

Explanation of the Sample Output 1:

In test case 1, For the given input, the following will be the schedule of the trains:
Train 1 arrived at 900 on platform 1.
Train 1 departed at 910 from platform 1.
Train 2 arrived at 940 on platform 1.

Train 3 arrived at 950 on platform 2 (since platform 1 was already occupied by train 1).
Train 4 arrived at 1100 on platform 3 (since both platforms 1 and 2 were occupied by trains 2 and 3 respectively).
Train 3 departed at 1120 from platform 2 (platform 2 becomes vacant).
Train 4 departed at 1130 from platform 3 (platform 3 also becomes vacant).
Train 2 departed at 1200 from platform 1 (platform 1 also becomes vacant).
Train 5 arrived at 1500 on platform 1.
Train 6 arrived at 1800 on platform 2.
Train 5 departed at 1900 from platform 1.
Train 6 departed at 2000 from platform 2.

Thus, minimum 3 platforms are needed for the given input.

In test case 2, For the given input, the following will be the schedule of the trains:

Train 1 arrived at 100 on platform 1.
Train 2 arrived at 200 from platform 2 (as platform 1 is occupied by train 1).
Train 1 departed at 200 from platform 1.
Train 3 arrived at 300 on platform 1.
Train 2 departed at 300 from platform 2.
Train 4 arrived at 400 on platform 2.
Train 3 departed at 400 from platform 1.
Train 4 departed at 500 from platform 2.

Thus, 2 platforms are needed for the given input.

Sample Input 2:

```
2
2
900 1000
999 1100
3
1200 1300 1450
1310 1440 1600
```

Sample Output 2:

```
1
2
```

Explanation of the Sample Output 2:

In test case 1, For the given input, the following will be the schedule of the trains:

Train 1 arrived at 900 on platform 1.
Train 1 departed at 999 from platform 1.
Train 2 arrived at 1000 on platform 1.
Train 2 arrived at 1100 on platform 1.

Thus, only 1 platform is needed for the given input.

In test case 2, For the given input, the following will be the schedule of the trains:

Train 1 arrived at 1200 on platform 1.
Train 2 arrived at 1300 on platform 2. (since platform 1 was already occupied by train 1).
Train 1 departed at 1310 from platform 1.
Train 2 departed at 1440 from platform 2.
Train 3 arrived at 1450 on platform 1.
Train 3 departed at 1600 on platform 1.

Thus, minimum 2 platforms are needed for the given input.

```
#include<bits/stdc++.h>
using namespace std;

int calculateMinPatforms(int at[], int dt[], int n) {
    // Write your code here.

    sort(at,at+n);
    sort(dt,dt+n);

    int plat_count=1;
    int result = 1;

    for(int i=1, j=0; i<n && j<n ;){
        if(at[i]<=dt[j]){
            plat_count++;
            i++;
        } else if(dt[j]<at[i]){
            plat_count--;
            j++;
        }


        if(plat_count>result){
            result = plat_count;
        }
    }

    return result;
}
```

10-Ass : Gas Tank

Send Feedback

You have a car with a gas tank of infinite capacity. There are ‘N’ gas stations along a circular route. Gas stations are numbered from 0 to N - 1. You begin the journey with an empty tank at one of the gas

stations. You want to travel around the circular route once in the clockwise direction. I.e if you start to travel from station ‘i’, then you will go to i + 1, i + 2, …, n - 1, 0, 1, 2, i - 1 and then return back to ‘i’.
You are given two integer arrays ‘gas’ and ‘cost’. ‘gas[i]’ gives the amount of gas available at station ‘i’ and cost[i] gives the amount of gas required to travel from station ‘i’ to next station i.e station ‘i’+1 (or 0 if the station is N - 1).
Return the starting gas station's index if it is possible to complete cycle of given circular route once in the clockwise direction. If there are multiple possible starting gas station’s indexes, then return the minimum of those gas station indexes. If there is no such gas station then return -1.

Input Format :

The first line of input contains a single integer T, representing the number of test cases or queries to be run, then the T test cases follow.

The first line of each test case contains a positive integer 'N' which represents the number of gas stations.

The second line of each test case contains 'N' space-separated integers representing the integer array 'gas'.

The third line of each test case contains 'N' space-separated integers representing the integer array 'cost'.

Output Format :

For each test case, print a single integer denoting the minimum index of the starting gas station if you are able to travel around the cycle once, otherwise print -1.

Note:

You do not need to print anything, it has already been taken care of. Just implement the given function.

Constraint :

1 <= T <=50
1 <= N <= 10 ^ 4
0 <= GAS[i] <= 10 ^ 5
0 <= COST[i] <= 10 ^ 5
Where GAS[i] represents the ith element of 'GAS' array,
COST[i] represents the ith element of 'COST' array.
Time Limit: 1 sec

Sample Input 1 :

```
1
2
1 2
2 1
```

Sample Output 1:

```
1
```

Explanation of Sample Input 1 :

Test Case 1:
If you start from index 0, you can fill in 1 unit of gas. Now your tank has 1 unit of gas. But you need 2 units of gas to travel to station 1. Thus you can not start at station 0.
If you start from index 1, you can fill in 2 units of gas. Now your tank has 2 units of gas. You need 1 unit of gas to get to station 0. So, you travel to station 0 and still have 1 unit of gas left. You fill in 1 unit of

additional gas, making your current gas = 2 unit. It costs you 2 amounts of gas to get to station 1, which you do and complete the cycle. Thus you can start at index 1.

Sample Input 2 :

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

Sample Output 2:

3

Explanation of Sample Input 2 :

Test Case 1:
If you start from index 3 and fill all available gas at each station, then you can reach station 4 with 3 units of gas, station 0 with 6 units of gas, station 1 with 4 units of gas, station 2 with 2 units of gas, and back to station 3 after consuming all the gas.

We can show that index 3 is the minimum index of the gas station from where the complete circular trip is possible

```
#include <iostream>
#include <vector>

using namespace std;


int minimumStartingIndex(vector<int> &gas, vector<int> &cost, int n)
{
    // Write your code here.


        int start = 0;
    int total = 0;
    int tank = 0;
    for(int i = 0; i < n ; i++) {
        tank = tank + gas[i] - cost[i];
        if(tank < 0) {
            start = i+1;
            total= total + tank;
            tank=0;
        }
    }
    if(total + tank < 0) {
        return -1;
    } else {
        return start;
    }
}
```