**1. Activity Selection**

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.

**Note : The start time and end time of two activities may coincide.**

**Input:**  
The first line contains T denoting the number of testcases. Then follows description of testcases. First line is N number of activities then second line contains N numbers which are starting time of activies.Third line contains N finishing time of activities.

**Output:**  
For each test case, output a single number denoting maximum activites which can be performed in new line.

**Constraints:**  
1<=T<=50  
1<=N<=1000  
1<=A[i]<=100

**Example:**  
**Input:**  
2  
6  
1 3 2 5 8 5  
2 4 6 7 9 9  
4  
1 3 2 5  
2 4 3 6

**Output:**  
4  
4

CODE:

using namespace std;

bool comparator(pair<int,int> p1,pair<int,int> p2)

{

if(p1.second<p2.second)

return true;

if(p1.second==p2.second)

if(p1.first<p2.first)

return true;

return false;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int str[n];

int end[n];

vector<pair<int,int>>v;

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

cin>>str[i];

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

cin>>end[i];

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

{

v.push\_back(make\_pair(str[i],end[i]));

}

sort(v.begin(),v.end(),comparator);

int cnt=1;

int startt=v[0].first;

int endt=v[0].second;

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

{

if(endt<=v[i].first)

{

endt=v[i].second;

cnt++;

}

}

cout<<cnt<<endl;

}

return 0;

}

**2. N meetings in one room**

There is **one** meeting room in a firm. There are **N** meetings in the form of (S[i], F[i]) where S[i] is start time of meeting i and F[i] is finish time of meeting i.

What is the maximum number of meetings that can be accommodated in the meeting room?

**Input:**  
The first line of input consists number of the test cases. The description of T test cases is as follows:  
The first line consists of the size of the array, second line has the array containing the starting time of all the meetings each separated by a space, i.e., S [ i ]. And the third line has the array containing the finishing time of all the meetings each separated by a space, i.e., F [ i ].

**Output:**  
In each separate line print the order in which the meetings take place separated by a space.

**Constraints:**  
1 ≤ T ≤ 70  
1 ≤ N ≤ 100  
1 ≤ S[ i ], F[ i ] ≤ 100000

**Example:**  
**Input:**  
2  
6  
1 3 0 5 8 5  
2 4 6 7 9 9  
8  
75250 50074 43659 8931 11273 27545 50879 77924  
112960 114515 81825 93424 54316 35533 73383 160252

**Output:**  
1 2 4 5  
6 7 1

CODE:

using namespace std;

bool comparator(pair<int,int> p1,pair<int,int> p2)

{

if(p1.second<p2.second)

return true;

if(p1.second==p2.second)

if(p1.first<p2.first)

return true;

return false;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int s[n];

int e[n];

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

cin>>s[i];

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

cin>>e[i];

vector<pair<int,int>> v;

map<pair<int,int>,int> mp;

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

{

v.push\_back(make\_pair(s[i],e[i]));

mp[v[i]]=i;

}

sort(v.begin(),v.end(),comparator);

int start=v[0].first;

int end=v[0].second;

//debug

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

// {

// cout<<mp[v[i]]+1<<" - "<<v[i].first<<" "<<v[i].second<<endl;

// }

// cout<<"-------------------------------------------------"<<endl;

cout<<mp[v[0]]+1<<" ";

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

{

// cout<<v[i].first<<" - "<<v[i].second<<endl;

if(end<=v[i].first)

{

cout<<mp[v[i]]+1<<" ";

end=v[i].second;

}

}

cout<<endl;

}

return 0;

}

**3. Maximize Toys**

Given an array **arr** of length **N** consisting cost of toys. Given an integer **K** depicting the amount with you. The task is to Maximise the number of different toys you can have with K amount.

**Example 1:**

**Input:** N = 7, K = 50

arr = {1, 12, 5, 111, 200, 1000, 10}

**Output:** 4

**Explaination:** The costs of the toys are

1, 12, 5, 10.

**Example 2:**

**Input:** N = 3, K = 100

arr = {20, 30, 50}

**Output:** 3

**Explaination:** We can buy all types of

toys.

**Your Task:**  
You do not need to read input or print anything. Your task is to complete the function **toyCount()** which takes the value N, K and the array arr and returns the maximum count of toys.

**Expected Time Complexity:** O(NlogN)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 ≤ N ≤ 1000  
1 ≤ K, arr[i] ≤ 10000

CODE:

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

//User function Template for C++

class Solution{

public:

int toyCount(int n, int k, int arr[])

{

sort(arr,arr+n);

int cnt=0;

int i=0;

while(k>=0 && i<n)

{

k=k-arr[i];

if(k>=0)

cnt++;

i++;

}

return cnt;

// code here

}

};

// { Driver Code Starts.

int main(){

int t;

cin>>t;

while(t--){

int N, K;

cin>>N>>K;

int arr[N];

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

cin>>arr[i];

Solution ob;

cout<<ob.toyCount(N, K, arr)<<endl;

}

return 0;

}

**4. Page Faults in LRU**

In operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needs to be replaced when the new page comes in. Whenever a new page is referred and is not present in memory, the page fault occurs and Operating System replaces one of the existing pages with a newly needed page. Given a sequence of pages and memory capacity, your task is to find the number of page faults using Least Recently Used (LRU) Algorithm.

**Input:**  
The first line of input contains an integer **T** denoting the number of test cases. Each test case contains n number of pages and next line contains space seaprated sequence of pages. The following line consist of the capacity of the memory.  
**Note:** Pages are referred in the order left to right from the array (i.e index 0 page is referred first then index 1 and so on). Memory is empty at the start.

**Output:**  
Output the number of page faults.

**Constraints:**  
1<=T<=100  
1<=n<=1000  
4<=capacity<=100

**Example:**  
**Input:**  
2  
9  
5 0 1 3 2 4 1 0 5  
4  
8  
3 1 0 2 5 4 1 2  
4

**Output:**  
8  
7

CODE:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

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

cin>>arr[i];

int k;

cin>>k;

map<int,int> tv;

map<int,int> vt;

int t=0;

int cnt=0;

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

{

if(vt.find(arr[i])==vt.end())

{

vt[arr[i]]=t;

tv[t]=arr[i];

t++;

cnt++;

while(vt.size()>k && tv.size()>k)

{

int x=tv.begin()->second;

tv.erase(tv.begin());

vt.erase(x);

}

}

else

{

tv.erase(vt[arr[i]]);

vt[arr[i]]=t;

tv[t]=arr[i];

t++;

}

}

cout<<cnt<<endl;

}

return 0;

}

**5. Largest number possible**

Given two numbers '**N**' and '**S**' , find the **largest number** that can be formed with '**N**' digits and whose sum of digits should be equals to '**S**'.

**Example 1:**

**Input:** N = 2, S = 9

**Output:** 90

**Explaination:** It is the biggest number

with sum of digits equals to 9.

**Example 2:**

**Input:** N = 3, S = 20

**Output:** 992

**Explaination:** It is the biggest number

with sum of digits equals to 20.

**Your Task:**  
You do not need to read input or print anything. Your task is to complete the function **findLargest()** which takes N and S as input parameters and returns the largest possible number. Return -1 if no such number is possible.

**Expected Time Complexity:** O(N)  
**Exepcted Auxiliary Space:** O(N)

**Constraints:**  
1 ≤ N ≤ 104  
1 ≤ S ≤ 105

CODE:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

int n,s;

cin>>n>>s;

if(s==0)

{

cout<<-1<<endl;

continue;

}

vector<int> v;

while(s>=0 && n>0)

{

if(s>9)

{

s=s-9;

v.push\_back(9);

}

else

{

v.push\_back(s);

s=0;

}

n--;

}

if(s!=0)

cout<<-1;

else

{

for(int i=0;i<v.size();i++)

cout<<v[i];

}

cout<<endl;

}

return 0;

}

**6. Minimize the heights**

Given an array **arr[]** denoting heights of **N** towers and a positive integer **K**, modify the heights of each tower either by increasing or decreasing them by **K** only once. Find out the minimum difference of the heights of shortest and longest modified towers.

**Example 1:**

**Input:**

K = 2, N = 4

Arr[] = {1, 5, 8, 10}

**Output:** 5

**Explanation:** The array can be modified as

{3, 3, 6, 8}. The difference between

the largest and the smallest is 8-3 = 5.

**Example 2:**

**Input:**

K = 3, N = 5

Arr[] = {3, 9, 12, 16, 20}

**Output:** 11

**Explanation:** The array can be modified as

{6 12 9 13 17}. The difference between

the largest and the smallest is 17-6 = 11.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **getMinDiff()** which takes the **arr[], n** and **k**as input parameters and returns an integer denoting the minimum difference.

**Expected Time Complexity:** O(N\*logN)  
**Expected Auxiliary Space:** O(1)  
  
**Constraints**  
1 <= K <= 104  
1 <= N <= 105  
1 <= Arr[i] <= 105

CODE:

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

//User function template for C++

class Solution{

public:

int getMinDiff(int arr[], int n, int k) {

sort(arr,arr+n);

int diff=arr[n-1]-arr[0];

int small=arr[0]+k;

int big=arr[n-1]-k;

if(small>big)

swap(small,big);

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

{

int sub=arr[i]-k;

int add=arr[i]+k;

if(sub>=small || add<=big)

continue;

if(big-sub<=add-small)

small=sub;

else

big=add;

}

return min(diff,big-small);

// code here

}

};

// { Driver Code Starts.

int main() {

int t;

cin >> t;

while (t--) {

int n, k;

cin >> k;

cin >> n;

int arr[n];

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

cin >> arr[i];

}

Solution ob;

auto ans = ob.getMinDiff(arr, n, k);

cout << ans << "\n";

}

return 0;

}

**7. Minimize the sum of product**

You are given two arrays,**A** and **B**, of equal size **N**. The task is to find the minimum value of A[0] \* B[0] + A[1] \* B[1] +…+ A[N-1] \* B[N-1], where shuffling of elements of arrays A and B is allowed.

**Input:**  
The first line of input contains an integer **T** denoting the no of test cases. Then**T** test cases follow. Each test case contains three lines. The first line contains an integer **N** denoting the size of the arrays. In the second line are N space separated values of the array **A[]**, and in the last line are N space separated values of the array **B[]**.

**Output:**  
For each test case, print the minimum sum.

**Constraints:**  
1 <= T <= 100  
1 <= N <= 107  
1 <= A[] <= 1018

**Example:  
Input:**  
2  
3   
3 1 1  
6 5 4  
5  
6 1 9 5 4  
3 4 8 2 4  
**Output:**  
23   
80

CODE:

using namespace std;

#define ll long long

int main()

{

ll t;

cin>>t;

while(t--)

{

ll n;

cin>>n;

ll arr1[n];

ll arr2[n];

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

cin>>arr1[i];

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

cin>>arr2[i];

sort(arr1,arr1+n);

sort(arr2,arr2+n);

ll sum=0;

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

{

sum+=arr1[i]\*arr2[n-i-1];

}

cout<<sum<<endl;

}

return 0;

}

**8. Huffman Decoding-1**

he task is to implement [Huffman Encoding](https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/) and [Decoding](https://www.geeksforgeeks.org/huffman-decoding/).

**Input:**  
First line consists of T test cases. Only line of every test case consists of String S.

**Output:**  
Single line output, return the Decoded String.

**Constraints:**  
1<=T<=100  
2<=S<=1000

**Example:  
Input:**  
2  
abc  
geeksforgeeks  
**Output:**  
abc  
geeksforgeeks

CODE:

#include <bits/stdc++.h>

#define MAX\_TREE\_HT 256

using namespace std;

map<char, string> codes;

map<char, int> freq;

struct MinHeapNode

{

char data;

int freq;

MinHeapNode \*left, \*right;

MinHeapNode(char data, int freq)

{

left = right = NULL;

this->data = data;

this->freq = freq;

}

};

struct compare

{

bool operator()(MinHeapNode\* l, MinHeapNode\* r)

{

return (l->freq > r->freq);

}

};

void printCodes(struct MinHeapNode\* root, string str)

{

if (!root)

return;

if (root->data != '$')

cout << root->data << ": " << str << "\n";

printCodes(root->left, str + "0");

printCodes(root->right, str + "1");

}

void storeCodes(struct MinHeapNode\* root, string str)

{

if (root==NULL)

return;

if (root->data != '$')

codes[root->data]=str;

storeCodes(root->left, str + "0");

storeCodes(root->right, str + "1");

}

priority\_queue<MinHeapNode\*, vector<MinHeapNode\*>, compare> minHeap;

void HuffmanCodes(int size)

{

struct MinHeapNode \*left, \*right, \*top;

for (map<char, int>::iterator v=freq.begin(); v!=freq.end(); v++)

minHeap.push(new MinHeapNode(v->first, v->second));

while (minHeap.size() != 1)

{

left = minHeap.top();

minHeap.pop();

right = minHeap.top();

minHeap.pop();

top = new MinHeapNode('$', left->freq + right->freq);

top->left = left;

top->right = right;

minHeap.push(top);

}

storeCodes(minHeap.top(), "");

}

void calcFreq(string str, int n)

{

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

freq[str[i]]++;

}

string decode\_file(struct MinHeapNode\* root, string s);

int main()

{

int t;

cin>>t;

while(t--){

codes.clear();

freq.clear();

minHeap=priority\_queue <MinHeapNode\*, vector<MinHeapNode\*>, compare>();

string str;

cin>>str;

string encodedString, decodedString;

calcFreq(str, str.length());

HuffmanCodes(str.length());

/\*cout << "Character With there Frequencies:\n";

for (auto v=codes.begin(); v!=codes.end(); v++)

cout << v->first <<' ' << v->second << endl;\*/

for (auto i: str)

encodedString+=codes[i];

//cout <</\* "\nEncoded Huffman data:\n" << \*/encodedString << endl;

decodedString = decode\_file(minHeap.top(), encodedString);

cout <</\* "\nDecoded Huffman Data:\n" << \*/decodedString << endl;

}

return 0;

}// } Driver Code Ends

/\*Complete the function below

Which contains 2 arguments

1) root of the tree formed while encoding

2) Encoded String\*/

string decode\_file(struct MinHeapNode\* root, string s)

{

string ans = "";

struct MinHeapNode\* curr = root;

for (int i=0;i<s.size();i++)

{

if (s[i] == '0')

curr = curr->left;

else

curr = curr->right;

// reached leaf node

if (curr->left==NULL and curr->right==NULL)

{

ans += curr->data;

curr = root;

}

}

// cout<<ans<<endl;

return ans+'\0';

}

**9. Minimum Spanning Tree**

Given a weighted, undirected and connected graph. The task is to find the sum of weights of the edges of the Minimum Spanning Tree.

**Input:**  
The first line of input contains an integer **T** denoting the number of testcases. Then T test cases follow. The first line of each testcase contains two integers V (starting from 1), E denoting the number of nodes and number of edges. Then in the next line are 3\*E space separated values a b w where a, b denotes an **edge** from **a** to **b** and **w** is the weight of the edge.

**Output:**  
For each test case in a new line print the sum of weights of  the edges of the Minimum Spanning Tree formed of the graph.

**User task:**  
Since this is a functional problem you don't have to worry about input, you just have to complete the function  **spanningTree()** which takes number of vertices **V**and the number of edges **E**anda graph **graph**as inputs and returns an integer denoting the sum of weights of the edges of the Minimum Spanning Tree.  
**Note:**Please note that input of graph is 1-based but the adjacency matrix is 0-based.

**Expected Time Complexity:**O(V2).  
**Expected Auxiliary Space:**O(V).

**Constraints:**  
1 <= T <= 100  
2 <= V <= 1000  
V-1 <= E <= (V\*(V-1))/2  
1 <= a, b <= N  
1 <= w <= 1000  
Graph is connected and doesn't contain self loops & multiple edges.

**Example:  
Input**:  
2  
3 3  
1 2 5 2 3 3 1 3 1  
2 1  
1 2 5

**Output**:  
4  
5

CODE:

#include <bits/stdc++.h>

using namespace std;

int spanningTree(int V, int E, vector<vector<int>> &graph);

// Driver code

int main() {

int t;

cin >> t;

while (t--) {

int V, E;

cin >> V >> E;

vector<vector<int> > graph(V, vector<int>(V, INT\_MAX));

int i=0;

while (i++<E) {

int u, v, w;

cin >> u >> v >> w;

u--, v--;

graph[u][v] = w;

graph[v][u] = w;

}

cout << spanningTree(V, E, graph) << endl;

}

return 0;

}

// } Driver Code Ends

// Function to construct and print MST for

// a graph represented using adjacency

// matrix representation, with V vertices.

// graph[i][j] = weight if edge exits else INT\_MAX

int spanningTree(int V, int E, vector<vector<int>> &g) {

vector<pair<int,int> > adj[V];

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

{

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

{

if(g[i][j]!=0)

adj[i].push\_back({j,g[i][j]});

}

}

int value[V];

for(int i=0;i<V;i++) value[i]=INT\_MAX;

vector<bool> vis(V,false);

value[0]=0;

int res=0;

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

{

int u=-1;

int mi=INT\_MAX;

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

{

if(!vis[i])

{

if(mi>value[i]||u==-1)

{

mi=value[i];

u=i;

}

}

}

vis[u]=true;

res+=value[u];

for(auto v:adj[u])

{

if(!vis[v.first])

value[v.first]=min(value[v.first],v.second);

}

}

return res;

}

**10. Shop in Candy Store**

In a candy store there are **N** different types of candies available  and the prices of all the N different types of candies are provided to you.  
You are now provided with an attractive offer.  
You can buy a single candy from the store and get atmost **K** other candies ( all are different types ) for free.  
Now you have to answer two questions. Firstly, you have to tell what is the **minimum amount of money** you have to spend to buy all the**N**different candies. Secondly, you have to tell what is the **maximum amount of money** you have to spend to buy all the N different candies.  
In both the cases you must utilize the offer i.e. you buy one candy and get **K**other candies for free.

**Input**   
The first line of the input contains **T** the number of test cases. Each test case consists of two lines. The first line of each test case contains the values of **N** and **K** as described above.  Then in the next line **N** integers follow denoting the price of each of the**N** different candies.

**Output**  
For each test case output a single line containing **2** space separated integers , the first denoting the **minimum amount of money required to be spent**and the second denoting the **maximum amount of money to be spent**.  
Remember to output the answer of each test case in a new line.  
  
**Constraints**        
1 <= **T**<= 50  
1 <= **N**<= 1000  
 0 <= **K** <= N-1  
1 <= **Ai** <= 100

**Expected Time Complexity :**O(nlogn)

**Example:**  
**Input**     
 1  
 4  2  
 3 2 1 4

**Output**  
3 7

CODE:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

int n,k;

cin>>n>>k;

int arr[n];

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

cin>>arr[i];

sort(arr,arr+n);

int r;

if(n%(k+1)==0)

{

r=n/(k+1);

}

else

{

r=(n/(k+1))+1;

}

// cout<<"r="<<r<<endl;

int ans1=0;

int ans2=0;

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

{

if(i<r)

ans1+=arr[i];

if(i>=(n-r))

ans2+=arr[i];

}

cout<<ans1<<" "<<ans2<<endl;

}

return 0;

}

**11. Geek collects the balls**

There are two parallel roads, each containing N and M buckets, respectively. Each bucket may contain some balls. The buckets on both roads are kept in such a way that they are sorted according to the number of balls in them. Geek starts from the end of the road which has the bucket with a lower number of balls(i.e. if buckets are sorted in increasing order, then geek will start from the left side of the road).  
The geek can change the road only at the point of intersection(which means, buckets with the same number of balls on two roads). Now you need to help Geek to collect the maximum number of balls.

**Input:**  
The first line of input contains T denoting the number of test cases. The first line of each test case contains two integers N and M, denoting the number of buckets on road1 and road2 respectively. 2nd line of each test case contains N integers, number of balls in buckets on the first road. 3rd line of each test case contains M integers, number of balls in buckets on the second road.

**Output:**  
For each test case output a single line containing the maximum possible balls that Geek can collect.

**Constraints:**  
1<= T <= 1000  
1<= N <= 10^3  
1<= M <=10^3  
0<= A[i],B[i]<=10^6

**Example:**  
**Input:**  
1  
5 5  
1 4 5 6 8  
2 3 4 6 9

**Output:**  
29

CODE:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

int n,m;

cin>>n>>m;

int arr1[n];

int arr2[m];

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

cin>>arr1[i];

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

cin>>arr2[i];

int sum1=0;

int sum2=0;

int ans=0;

int i=0,j=0;

while(i<n && j<m)

{

if(arr1[i]==arr2[j])

{

int p=arr1[i];

while(arr1[i]==p && i<n)

{

sum1+=arr1[i];

i++;

}

while(arr2[j]==p && j<m)

{

sum2+=arr2[j];

j++;

}

ans+=max(sum1,sum2);

sum1=0;

sum2=0;

}

else if(arr1[i]<arr2[j])

{

sum1+=arr1[i];

i++;

}

else

{

sum2+=arr2[j];

j++;

}

}

while(i<n)

{

sum1+=arr1[i];

i++;

}

while(j<m)

{

sum2+=arr2[j];

j++;

}

ans+=max(sum1,sum2);

cout<<ans<<endl;

}

return 0;

}