# 1)Maximum sum of sub Array

Sub Array is defined as continuous block of Element associated with I and j whose return maximum sum values

1<=i<=j<=N;

Number of Sub array possible nc2 + n

Algorithm:

// language is following 1 indexing

Time complexity o(n^3)

maxSum <-int\_min

For I in [1..n]

For j in [I …n]

Sum <-0

For k in [I …j]

Sum =sum +a[i];

maxSum=max(maxSum,sum)

Better Algorithm

Time complexity o(n^2)

1)we are basically finding sum of an Elements over and over..

For I in [1..n]:

Sum <-0;

For j in [i…n]:

Sum =sum +A[i] // Keep Adding Elements

maxSum =max(maxSum,sum) // keep updating maxSum

Time complexity o(n) (kadane`s algorithm)

Space complexity o(1)

How does algorithm work?

Ans: maximum sum of sub Array should be defined over an array of containing elements both positive and negative

Lets talk about observations of an array:

1)ignore those Elements whose maximum sum of sub array is negative (main thing of kadanes algorithm)

2)find out current sum of Every element return the maximum value among them

Current sum of ith Element defines the maximum sum of sub array with Ith Element

Ci <- ci+a[i] if ci-1 >0

A[i] otherwise

#include <bits/stdc++.h>

#define max(a,b) (a >b ? a : b)

#define endl "\n"

using namespace std;

int main() {

// your code goes here

int t;

cin>>t;

while(t--){

int n;

cin>>n;

int ms=INT\_MIN,cs=0;

int num=0;

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

cin>>num;

cs=max(cs+num,num);

ms=max(ms,cs);

}

cout<<ms<<endl;

}

return 0;

}

# Gcd using brute force technique

Time complexity o(min(a,b));

#include <bits/stdc++.h>

#define max(a,b) (a >b ? a : b)

#define min(a,b) (a >b ? b : a)

#define endl "\n"

using namespace std;

int main() {

// your code goes here

int t;

cin>>t;

while(t--){

int a,b;

cin>>a>>b;

int ans=1; // we know that every number is divisable by 1

for(int i=1;i<=min(a,b);i++){

if(a%i==0 and b%i==0)

ans=i;

}

cout<<ans<<endl;

}

return 0;

}

# GCD using Euclid Algorithm

#include <bits/stdc++.h>

#define max(a,b) (a > b ? a : b)

#define min(a,b) (a > b ? b : a)

using namespace std;

long gcd(long i,long j){

if(i==0)

return j;

return gcd(min(i,j%i),max(i,j%i));

}

int main() {

// your code goes here

int t;

cin>>t;

while(t--){

long int ans=1;

int n;

cin>>n;

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

int num;

cin>>num;

ans= (long(ans \* num))/gcd(ans,num);

}

cout<<ans<<endl;

}

return 0;

}

## Minimum absolute difference

Time complexity is o(n^2)

We need to take care about all pair of Element which could be done in order n^2 operations; have to use 2 for loops in order to achieve this task

Number of pairs of length n is n choose 2;

#include <bits/stdc++.h>

#define max(a,b) (a > b ? a : b)

#define min(a,b) (a > b ? b : a)

using namespace std;

int main() {

// your code goes here

int n;

cin>>n;

int a[n];

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

cin>>a[i];

int min=INT\_MAX;

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

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

min=min(min,abs(a[i]-a[j]));

}

}

cout<<min<<endl;

return 0;

}

## Better Algorithm

Since our Task Is to find out all pairs of given Array which is not going to take care about order

This kind of problems can be Solved where order is not important using sorting Technique.

#include <bits/stdc++.h>

#define min(a,b) (a > b ? b : a)

using namespace std;

int main() {

// your code goes here

int n;

cin>>n;

int a[n];

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

cin>>a[i];

int min=INT\_MAX;

sort(a,a+n);

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

min=min(min,abs(a[i]-a[i+1]));

}

cout<<min<<endl;

return 0;

}

## You are given a square, and you want to know the maximum number of regions you create by drawing n straight lines.

Recurrence relation

T(n)=T(n-1) + n;

T(1)=2;

#include <iostream>

using namespace std;

int main() {

// your code goes here

int t;

cin>>t;

while(t--){

int n;

cin>>n;

cout<< 1 + ((n \*(n+1)) /2)<<endl;

}

return 0;

}

Loss less compression -> if you are not losing any data and you are able represent data in less space then it is called loss less compression

Ex: Huffman coding..

Data->data1 (even lesser space ) but you are losing some amount of data (Lossy compression)

Question arises :is it usefull method?

Since some tasks are required 70 % of data

Ex Image.

## Run length Encoding :

Run length Encoding is one the data compression technique loss less Technique in which a stream of incoming data in converted into an output stream of count of consecutive data values in a sequence..

For Example …RLE(“AAABBCCCC”)->A3B2C4

But this definition of run length Encoding need to be a loss less compression

We need to define the rules:

AAAABBCCCCC->A!4BBC!5

Explanation marks make sense we have number in string

Algorithm:

1->we are adding first character into output string;

By doing this step the solution becomes easier..

2->we are using count variable to keep track of number of times a variable is occurred.

#include <bits/stdc++.h>

using namespace std;

string RLE(string s){

string rle="";

rle+=s[0];

int count=1;

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

if(s[i]==s[i-1])

count++;

else{

rle+=to\_string(count);

rle+=s[i];

count=1;

}

}

rle+=to\_string(count);

return rle;

}

int main() {

int t;

cin>>t;

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

cout<<"case #"<< i<< ": ";

string s;

cin>>s;

cout<<RLE(s)<<endl;

}

return 0;

}

## Probem:

You are given two run length encoding string ,and you need to find the whether their string are same or not.

But you cannot decode it into original string because its original string might be large that is not fit in an array.so you are not allowed to do this task.if you are doing this .then there is noo point of RLE

#include <bits/stdc++.h>

using namespace std;

string RLE(string s){

string rle="";

rle+=s[0];

int count=1;

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

if(s[i]==s[i-1])

count++;

else{

rle+=to\_string(count);

rle+=s[i];

count=1;

}

}

rle+=to\_string(count);

return rle;

}

string optRLE(string s){

string opt="";

opt+=s[0];

string tempcnt="";

int cnt=0;

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

if('0'<=s[i] && s[i]<='9'){

tempcnt+=s[i];

}

else {

if(opt[opt.length()-1]==s[i]){

cnt +=stoi(tempcnt);

}else{

cnt+=stoi(tempcnt);

opt+=to\_string(cnt);

opt+=s[i];

cnt=0;

}

tempcnt="";

}

}

cnt+=stoi(tempcnt);

opt+=to\_string(cnt);

return opt;

}

int main() {

int t;

cin>>t;

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

cout<<"case #"<< i<< ": ";

string s;

cin>>s;

cout<<optRLE(s)<<endl;

}

return 0;

}

## Your task is to start from a point with least X value and highest y value to a point having with highest y value ,least x value ,in this process you can move only forward …

#include <bits/stdc++.h>

using namespace std;

#define pi pair<int ,int>

#define F first

#define S second

#define pb push\_back

#define mp make\_pair

#define FOR(i,a,b) for(int i=(a);i<(b);i++)

bool comp(const pi &a,const pi &b){

if(a.F == b.F) return a.S > b.S;

return a.F < b.F;

}

void solve(){

int n;

cin>>n;

vector<pi> v(n);

FOR(i,0,n) cin>>v[i].F>>v[i].S;

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

double ans=0.0;

FOR(i,1,n){

ans+=sqrt(

((v[i].F-v[i-1].F)\*(v[i].F - v[i-1].F)) + ((v[i].S - v[i-1].S)\*(v[i].S - v[i-1].S))

);

}

cout<<fixed<<setprecision(2)<<ans<<endl;

}

int main() {

// your code goes here

int t;

cin>>t;

while(t--){

solve();

}

return 0;

}

# Binary Exponentiation:

#include<iostream>

using namespace std;

long long power(int a,int b){

if(b==0){

return 1;

}

long int x=power(a,b/2);

if(b%2==0){

return x\*x;

}

else{

return x\*x\*a;

}

}

int main(){

int a,b;

cin>>a>>b;

long int result=power(a,b);

cout<<"result is"<<result;

return 0;

}