PROBLEMA 1: COMPLESSITA O(NlogN)

#include <bits/stdc++.h>

using namespace std;

int count(int arr[], int low, int high, int x);

int main(){

int test;

int k,n;

int j=0;

int occorrenze[100];

cin>>test;

while(test--){

cin>>k;

cin>>n;

int\* vett= new int[n];

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

cin>>vett[i];

}

occorrenze[j]=count(vett,0,n-1,k);

j++;

delete[] vett;

}

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

cout<<occorrenze[i]<<endl;

}

}

int count(int vett[], int inizio, int fine, int k){

//Se il sottoarray non è valido oppure l'elemento non è stato trovato

if ((inizio > fine) || (inizio == fine && vett[inizio] != k)){

return 0;

}

//Se c'è una singola occorrenza

if (inizio == fine && vett[inizio] == k){

return 1;

}

//Divisione del sottoproblema dividendo l'array in due sottoarray

//L'idea è quella di contare poi in entrambi

return count(vett, inizio,(inizio + fine)/2,k) + count(vett,1+(inizio+fine)/2,fine,k);

}

PROBEMA 2: COMPLESSITA’ O(1\*(S-num\_prec)^3)

#include <bits/stdc++.h>

using namespace std;

bool is\_prime(int n);

void trova\_primi(int n, int s, int p, int\* vett, int ind\_vett, int num\_prec, int tot);

int main(){

int test;

int s,n,p;

int\* seq;

int ind\_vett=0;

int numTest=1;

cin>>test;

while(test--){

cin>>s>>n>>p;

seq=new int[n];

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

seq[i]=0;

}

cout<<"CASO DI TEST "<<numTest<<endl;

trova\_primi(n,s,p,seq,ind\_vett,p+1,0);

numTest++;

}

}

void trova\_primi(int n, int s, int p, int\* vett, int ind\_vett, int num\_prec, int tot){

if(tot==s and ind\_vett==n){

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

cout<<vett[i]<<" ";

}

cout<<endl;

return;

}

if(tot>s or ind\_vett>n){

return;

}

for(int i=num\_prec; i<=s; i++){

if(is\_prime(i)){

//Aggiungo al vettore, aumento il totale

vett[ind\_vett]=i;

tot+=i;

ind\_vett++;

//Vado avanti

trova\_primi(n,s,p,vett,ind\_vett,i+1,tot);

//Undo

ind\_vett--;

tot-=i;

vett[ind\_vett]=0;

}

}

}

bool is\_prime(int n){

bool is\_prime=true;

if (n==0||n==1) {

is\_prime = false;

}

for(int i = 2; i <= n/2; ++i) {

if (n%i == 0) {

is\_prime = false;

break;

}

}

return is\_prime;

}