FRANCESCO DELLA VALLE M63001500

**PROBLEMA 1**

#include <iostream>

using namespace std;

int count(int array[], int low, int high, int value){

if ((low > high) || (low == high && array[low] != value)){

return 0;

}

>

if (low == high && array[low] == value){

return 1;

}

return count(array, low, (low + high) / 2, value) + count(array, 1 + (low + high) / 2, high, value);

}

int main(){

int tc, dim, value;

cin>> tc;

while(tc--){

cin>>value;

cin>> dim;

int array[dim];

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

cin>>array[i];

}

cout<<count(array, 0, dim-1, value);

}

return 0;

}

*Complessità temporale*: O(n logn)

**PROBLEMA 2**

#include <iostream>

using namespace std;

bool isPrime(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;

}

bool isSafe(int n, int \*output, int k){

if(!isPrime(n) ){

return false;

}

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

if(output[i] == n){

return false;

}

}

return true;

}

bool isASolution(int \*output, int N, int S){

int sum = 0;

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

sum += output[i];

}

if(sum == S){

return true;

}else{

return false;

}

}

void printSolution(int \*output,int N){

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

cout<<output[i]<<" ";

}

cout<<endl;

}

bool PrimeSum(int N, int P, int S, int \*output, int k,int number){

if( k == N ){

if(isASolution(output, N, S)){

printSolution(output, N);

return true;

}else{

return false;

}

}

bool res = false;

for(int i=number+1; i<=S ; i++ ){

if(isSafe(i,output,k)){

output[k]= i;

res = PrimeSum(N, P, S, output, k+1,i) || res;

output[k] = 0;

}

}

return res;

}

int main(int argc, const char \* argv[]) {

int S, N, P, tc;

int \*output;

cout<<"Inserisci numero di casi di test"<<endl;

cin>>tc;

int tc\_count = 0;

while(tc\_count < tc){

cin>> S;

cin>> N;

cin>> P;

output = new int[N];

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

output[i] = 0;

}

cout<<"CASO DI TEST "<<tc\_count+1<<endl;

PrimeSum(N, P, S, output, 0,P);

delete[] output;

tc\_count++;

}

return 0;

}

*Complessità temporale*: O(2^(S-P))