Prova in itinere – Michele Morgillo M63001467

**PROBLEMA 1**

#include <iostream>

using namespace std;

int count(int arr[], int first, int last, int val)

{

// caso di sotto-array non valido o valore non trovato

if ((first > last) || (first == last && arr[first] != val))

return 0;

// ritorna 1 se c'è almeno un solo elemento uguale a val

if (first == last && arr[first] == val)

return 1;

// divido l'array in due parti e conto le occorrenze di val in entrambe le parti

return count(arr, first,(first + last) / 2, val) + count(arr, 1 + (first + last) / 2, last, val);

}

int main()

{

int tc, dim, value, arr[500];

cin>>tc;

while(tc--){

cin>>value;

cin>>dim;

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

cin>>arr[i];

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

}

return 0;

}

**Complessità temporale:** *O(nlogn)*

**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 \*sequence, int k){

if( !isPrime(n) ){

return false;

}

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

if(sequence[i] == n){return false;}

}

return true;

}

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

int sum = 0;

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

sum += sequence[i];

}

if(sum == S){

return true;

}else{

return false;

}

}

void PrintSolution(int \*sequence,int N){

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

cout<<sequence[i]<<" ";

}

cout<<endl;

}

bool primeBacktrack(int N, int P, int S, int \*sequence, int k, int number){

if( k == N ){

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

PrintSolution(sequence, N);

return true;

}else{

return false;

}

}

bool res = false;

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

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

sequence[k]= i; //make move

res = primeBacktrack(N, P, S, sequence, k+1,i) || res; //backtracking call

sequence[k] = 0; //unmake move

}

}

return res;

}

int main()

{

int N, P, S;

int \*sequence;

int tc;

int tc\_counter = 0;

cin>>tc;

while(tc\_counter < tc){

cin>>S>>N>>P;

sequence = new int[N];

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

sequence[i] = 0;

}

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

primeBacktrack(N, P, S, sequence, 0,P);

delete[] sequence;

tc\_counter++;

}

return 0;

}

**Complessità temporale:** *O(2^(S-P))*