**SAVIANO GAETANO M63001502**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROBLEMA 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

#include <iostream>

using namespace std;

#define N 10

void fill\_array(int arr[], int& n) {

cout << "Inserisci riempimento del vettore: ";

cin >> n;

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

cout << "Inserisci l'elemento n " << i << " :";

cin >> arr[i];

}

}

//funzione che trova le occorrenze di un dato elemento in un sottoarray

int counteggio(int arr[], int low, int high, int x)

{

//se low è più grande di high oppure l'elemento non è stato trovato

if ((low > high)

|| (low == high && arr[low] != x))

return 0;

// se ce ne sta soltanto uno uguale all'elemento da cercare

if (low == high && arr[low] == x)

return 1;

// Dividere l'array in due parti e poi andarlo a dividere volta per volta in sottoarray più piccoli pe rtrovare le occorrenze

return counteggio(arr, low,

(low + high) / 2, x)

+ counteggio(arr, 1 + (low + high) / 2,

high, x);

}

int main()

{

int arr[N];

int n;

int casi\_test;

cout << "Quanti casi di test vuoi fare?";

cin >> casi\_test;

for (int i = 0;i < casi\_test;i++) {

fill\_array(arr, n);

n = sizeof(arr) / sizeof(int);

int x;

cout << "le occorrenze di quale numero vuoi ricercare?";

cin >> x;

cout << counteggio(arr, 0, n - 1, x) << endl;

}

return 0;

}

**Studio della complessità del primo**

**T(n) = 2T(n/2) + O(1) possiamo quindi ricondurlo al primo caso del teorema dell’esperto**

**COMPLESSITA’ DELL’ALGORITMO : Ө(n) oppure O(nlogn).**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROBLEMA 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

#include <iostream>

using namespace std;

//prototipi delle funzioni

bool isPrime(int n);

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

bool primeBacktrack(int N, int P, int S, int\* output, int k, int h);

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

void StampaSoluzione(int\* output, int N);

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

int N, P, S;

int\* output;

int casi\_test;

int counter\_casi\_test = 0;

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

cin >> casi\_test;

while (counter\_casi\_test < casi\_test) {

cin >> S;

cin >> N;

cin >> P;

output = new int[N];

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

output[i] = 0;

}

cout << "CASO DI TEST " << counter\_casi\_test + 1 << endl;

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

delete[] output;

counter\_casi\_test++;

}

return 0;

}

//funzione che ci determina se il numero è un numero primo o meno

bool isPrime(int n) {

bool is\_prime = true;

if (n == 0 || n == 1) { //se un numero è 0 o è 1 non è un numero primo

is\_prime = false;

}

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

if (n % i == 0) {

is\_prime = false;

break;

}

}

return is\_prime;

}

//funzione che ci dice se lo step è safe

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;

}

//funzione di backtracking

bool primeBacktrack(int N, int P, int S, int\* output, int k, int h) {

if (k == N) {

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

StampaSoluzione(output, N);

return true;

}

else {

return false;

}

}

bool result = false;

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

if (isSafe(i, output, k)) { //se è safe va ad inserire l'elemento

output[k] = i;

result = primeBacktrack(N, P, S, output, k + 1, i) || result;

output[k] = 0;

}

}

return result;

}

//funzione che dice se è una soluzione o meno

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

int somma = 0;

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

somma += output[i];

}

if (somma == S) {

return true;

}

else {

return false;

}

}

//stampa della soluzione

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

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

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

}

cout << endl;

}

**la complessità dell’algoritmo del 2° problema è O(2^(S-P))**